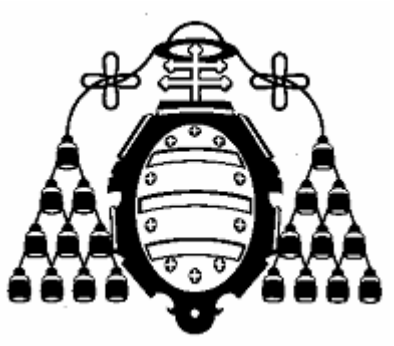


UNIVERSIDAD DE OVIEDO

ESCUELA POLITÉCNICA SUPERIOR DE INGENIERÍA DE GIJÓN

PROYECTO FIN DE CARRERA Nº 1052067

SHIFT Videogame

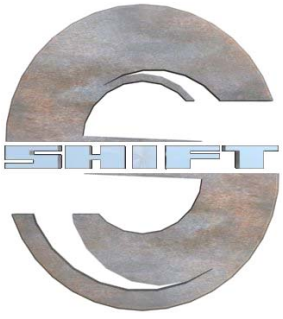
Autor
Alberto Carlos del Blanco Maraña
Marzo 2008

Director
Iván Fernández Lobo

ÍNDICE DE LA MEMORIA

Capítulo 1
Alcance

16 de marzo de 2008



Capítulo 1: Alcance

Tabla de apartados

Lista de figuras

1. Introducción (*alcance*)

Los orígenes de los videojuegos se remontan a 1948, cuando Thomas T. Goldsmith Jr. y Estle Ray Mann patentaron la idea de un dispositivo de entretenimiento mediante un tubo de rayos catódicos. La primera aplicación directa fue "*Tennis for Two*" de William Higinbotham en 1958 como un pequeño juego de tenis por medio de un osciloscopio.

Desde entonces los videojuegos se han extendido a múltiples dispositivos (ordenadores personales, videoconsolas, móviles, etc.) y han hecho uso de infinidad de áreas del conocimiento (informática, física, inteligencia artificial, redes, etc.)

Hoy en día la gran demanda de un mercado generoso para con esta disciplina ha hecho que el software de entretenimiento haya evolucionado de manera vertiginosa. En apenas dos décadas hemos pasado de los clásicos 2D dónde destacaba la jugabilidad, hasta los videojuegos 3D actuales que aportan mayor realismo.

Desde un punto de vista personal, el mundo de los videojuegos ha ganado en prestaciones, pero en muchos casos ha perdido la esencia de los clásicos: aquellos juegos que con una sencillez extrema podían aportar gran jugabilidad.

El presente proyecto pretende aunar las virtudes de los clásicos 2D con las capacidades 3D actuales. Se tratará de aportar **sencillez y jugabilidad** al mismo tiempo que **prestaciones y realismo**.

Este objetivo se plantea desde una óptica académica, tratando de ser en todo caso autocontenido: se incluye el desarrollo de todos los recursos necesarios, desde los modelos gráficos hasta el propio motor físico. Éste carácter integral obliga a limitar drásticamente las aspiraciones del proyecto, por lo que no pretende ser ambicioso en ninguno de los dos aspectos propuestos.

Antes de comenzar un análisis detallado, es conveniente definir el alcance lógico y técnico del proyecto.

2. Alcance lógico

SHIFT Videogame se propone como un videojuego de coches, donde la jugabilidad no se centra en la simulación, sino en los añadidos arcade.

En general, a diferencia de la gran mayoría de los juegos de este tipo, el propósito del juego no es completar un trazado sobre un circuito cerrado, sino que basándose en la simulación se deberán recoger diferentes elementos dentro de un tiempo determinado y sin ocasionar daños al vehículo.

La lógica del videojuego se construye en base a cuatro tipos de entidades:

- **Arenas** Las arenas definen el entorno y las propiedades del juego. El entorno se define por una serie de objetos fijos y móviles situados para poner a prueba la habilidad del jugador. Las propiedades de juego se caracterizan mediante una serie de parámetros cómo el tiempo, los objetivos, etc.
- **Vehículos** Los vehículos definen al jugador. Se podrá escoger entre diferentes tipos con propiedades claramente diferenciadas, tales como la aceleración, velocidad máxima, masa, vida, capacidad, etc.
- **Elementos** Los elementos definen cómo cambia el estado del juego en cada momento. Son objetos que aparecen aleatoriamente sobre la arena y que pueden capturarse con el vehículo.
- **Armas** Los vehículos podrán manipular armas, que pueden resultar útiles para perjudicar a los contrincantes o para facilitar su propia victoria.

En el “Capítulo 2: Análisis” se evaluará de un modo detallado cada una de las entidades lógicas aquí listadas.

3. Alcance técnico

SHIFT Videogame se desarrolla a bajo nivel, por lo que desde un punto de vista técnico supone la aplicación de varias áreas de conocimiento dentro de la informática, desde el modelado de los objetos hasta la implementación de un motor gráfico en su totalidad.

El software se desarrolla bajo el entorno C++.NET, haciendo uso únicamente de las librerías gráficas DirectX, sin ningún otro recurso adicional, por lo que es necesario diseñar los siguientes módulos gráficos:

- **Física** Para gestionar la geometría, texturas y parámetros físicos de los objetos.
- **Colisiones** Para tratar las colisiones entre los objetos físicos que tengan esta cualidad.
- **Partículas** Para incluir sistemas de partículas con usos diversos (humo, nieve, etc.).
- **Sonidos** Para manipular los buffer de sonido asociados a los objetos físicos.
- **Utilidades** Para mantener la cámara, luz, interfaz de usuario y un sistema de carga XML.
- **Lógico** Para integrar la lógica del videojuego dentro del motor desarrollado.

Además de ello y como parte más creativa que técnica, resulta necesario definir:

- **Modelado** Diseño gráfico 2D y 3D de los modelos, todos ellos construidos a medida para el proyecto.
- **Sonidos** En este caso, los sonidos del videojuego se han tomado de librerías con licencias copyleft.

En el “Capítulo 3: Diseño” se detallará cada uno de los módulos técnicos aquí listados.

4. Planificación

Desde el inicio del proyecto, en noviembre de 2005, se identificaron y planificaron las siguientes tareas para el desarrollo del mismo:

- **Análisis**
  - **Alcance**
    Obtener una definición formal del proyecto para fijar la propuesta.
  - **Formación**
    Formación en las diferentes soluciones disponibles para el desarrollo.
  - **Web**
    Elaboración de un soporte Web para informar sobre el estado del proyecto.
- **Diseño**
  - **Técnico**
    Definir los aspectos técnicos de la aplicación.
  - **Gráfico**
    Fase para la generación de los modelos gráficos del videojuego.
- **Implementación**
  - **Gráfico**
    Motor gráfico base sobre el que desarrollar el resto de módulos necesarios.
  - **Física**
    Implementación de la física del entorno.
  - **Colisiones**
    Implementación de las colisiones entre objetos.
  - **Partículas**
    Implementación de los sistemas de partículas.
  - **Sonidos**
    Implementación de los sonidos 3D.
  - **Utilidades**
    Implementación de diferentes utilidades para el motor gráfico.
  - **Lógico**
    Implementación de la lógica propia del videojuego.
- **Pruebas**
  Fase para probar el software bajo diferentes equipos y configuraciones.
- **Revisión documentación**
  Fase para ultimar la documentación del proyecto.
- **Presentación**
  Fase para resumir el proyecto en una breve presentación.

Estas tareas se iniciaron el 1 de diciembre de 2004, y se planificaron para ser combinadas con el último curso de la ingeniería informática, fijando su finalización a finales de octubre de 2005. La planificación propuesta seguía el siguiente diagrama de Gantt:

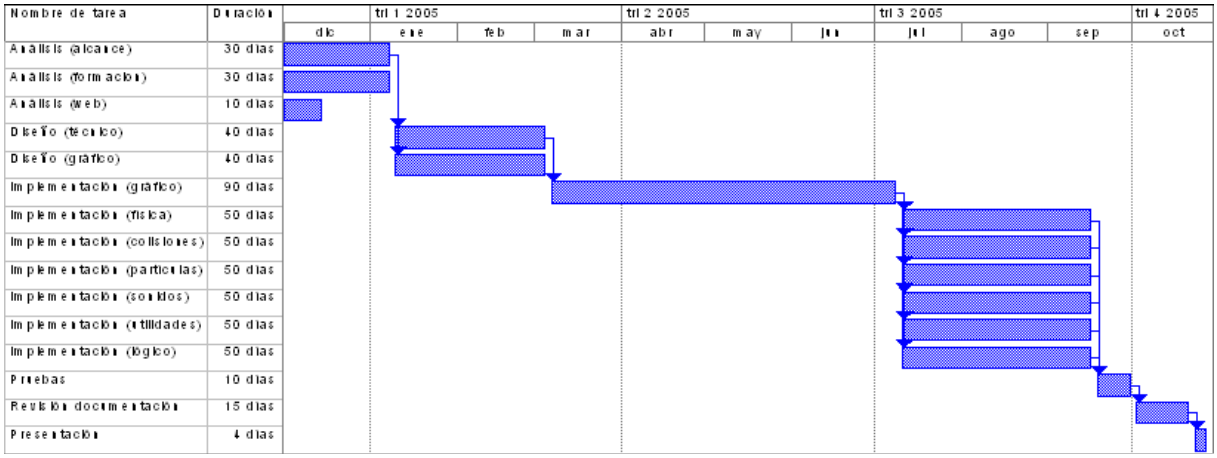


Figura 1: Planificación inicial [01-12-2004]

A finales de junio de 2005, el proyecto se ve interrumpido por motivos laborales durante algo más de dos años. Cuando se decide retomarlo, durante el verano de 2007, la planificación temporal fue reestructurada del siguiente modo:

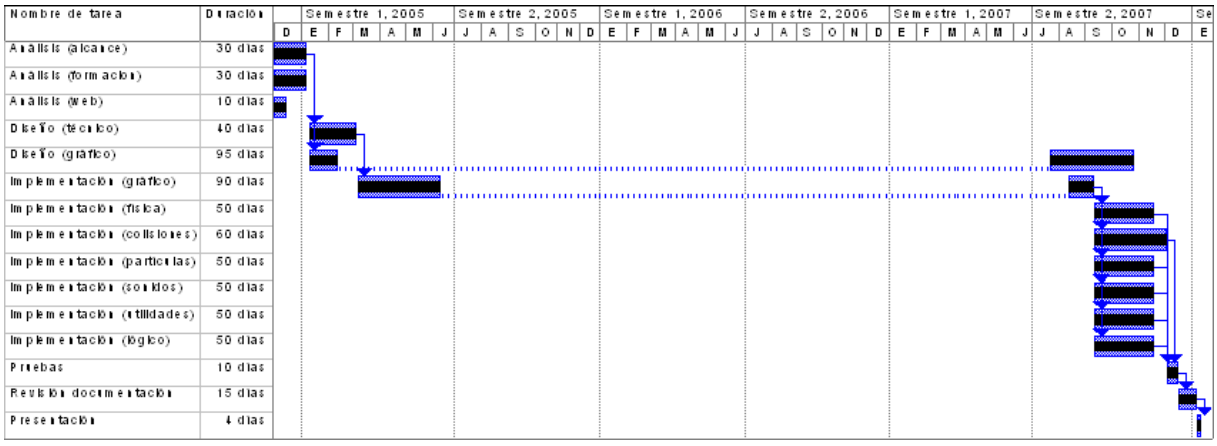


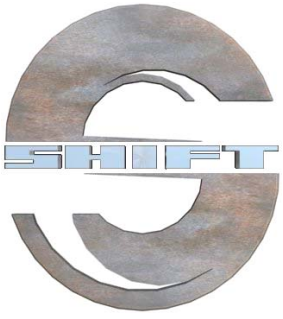
Figura 2: Planificación tras retraso por motivos laborales [01-07-2007]

El proyecto se finalizó a mediados de enero de 2008, tras más de tres años desde su inicio. Sin embargo, obviando la interrupción durante el desarrollo, las tareas planificadas no supusieron mayores retrasos sobre la planificación inicial.

Capítulo 2

Análisis

16 de marzo de 2008



Capítulo 2: Análisis

Tabla de apartados

Lista de figuras

5. Introducción (*análisis*)

La idea original del juego consiste en recoger ciertos elementos objetivo dentro de un tiempo fijado por un contador hacia atrás. En cada momento habrá un único elemento objetivo, situado aleatoriamente en el escenario y del que conoceremos su situación. Su captura provocará la aparición de otro elemento objetivo en otro punto diferente, aumentando ligeramente el contador de tiempo para la siguiente captura.

El jugador ganará cuando haya capturado un número determinado de objetivos y perderá si en algún momento el contador llega a cero o si agota la vida de su vehículo.

Como se podrá distinguir más adelante, tanto el aspecto gráfico de los elementos del juego como su funcionamiento han sido diseñados para lograr un compendio entre sencillez y realismo.

A continuación puede verse un boceto que resume la idea original del proyecto antes de su desarrollo (diciembre 2004) y una captura de la última versión del videojuego (enero 2008).

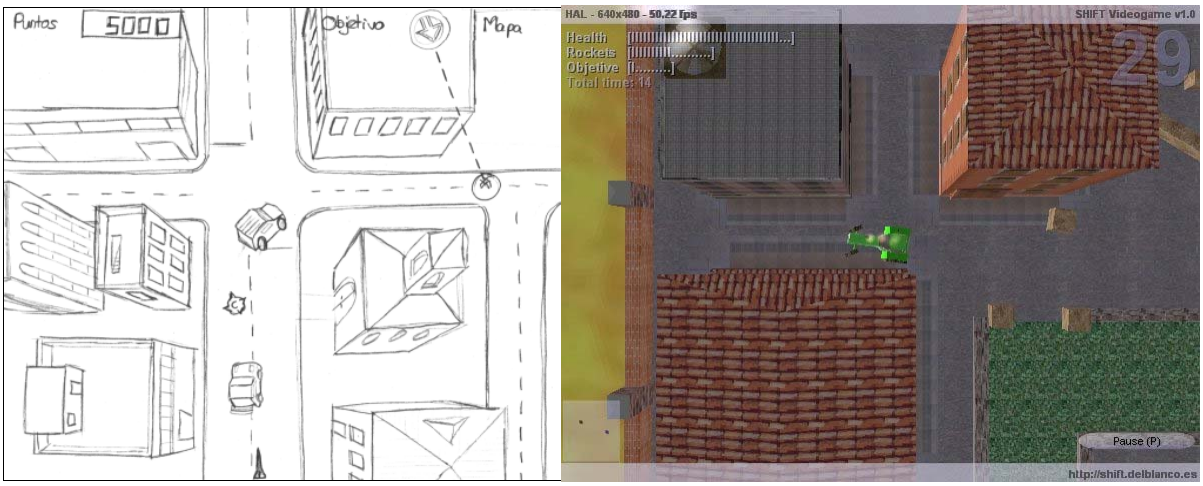


Figura 3: Idea inicial y final del juego

En los sucesivos apartados de este capítulo vamos a detallar los aspectos lógicos del videojuego, con el fin de analizar las entidades que intervienen, el funcionamiento del mismo y su manejo por parte del usuario.

6. Entidades del juego

En este apartado se tratarán de detallar los cuatro tipos de entidades lógicas que conforman el juego, dando a conocer algunos aspectos del funcionamiento del mismo:

- Arenas
- Vehículos
- Elementos
- Armas

6.1. Arenas

El juego se desarrollará en escenarios llamados **arenas**, dónde se podrá dirigir el vehículo hacia todas las direcciones. Estos escenarios contienen objetos fijos (edificios, barreras, etc.) y objetos móviles (bloques móviles, aspas giratorias, etc.) con el fin de poner a prueba la habilidad del jugador.

Dependiendo del tamaño y la distribución de objetos, cada arena tendrá asignados unos parámetros de juego diferentes, como son:

- Tiempo inicial para el contador hacia atrás
- Tiempo añadido por cada objetivo capturado
- Número de objetivos a capturar
- Factor de deslizamiento del suelo de la arena

El juego parte de un conjunto amplio de arenas, pero debe facilitar la creación e incorporación de otras nuevas, de forma que el programa sea flexible en este aspecto y permita ser ampliado incluso por el propio usuario.

A modo de ejemplo, veremos los dos tipos de arena que se incorporan inicialmente.

6.1.1. Arenas *Small_XX*

Son arenas pequeñas, con una geometría sencilla, diseñadas para que el usuario se adecue con el manejo del vehículo y el entorno de juego. Sin embargo, los tiempos de captura de objetivos son pequeños, por lo que la velocidad de juego es alta.

A continuación podemos ver varios ejemplos de estas arenas:

Figura 4: Arena *Small 11*

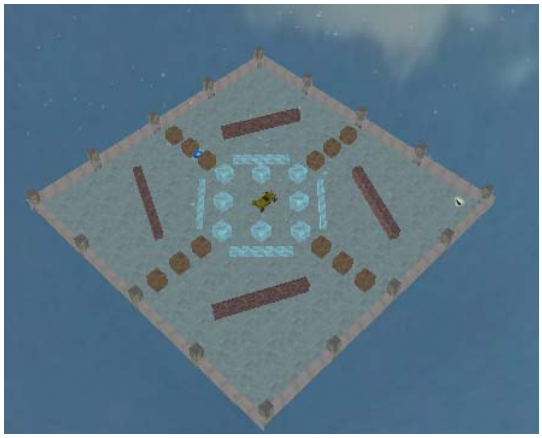


Figura 5: Arena *Small 12*

Figura 6: Arena *Small 13*

6.1.2. Arenas *Large_XX*

Son arenas más grandes, dónde la geometría será algo más compleja, con mayor número de objetos. Sin embargo estas arenas disponen de mayor tiempo de captura de objetivos, por lo que la velocidad de juego se reduce.

A continuación podemos ver varios ejemplos de estas arenas:

Figura 7: Arena *Large 21*

Figura 8: Arena *Large 22*

Figura 9: Arena *Large 23*

6.2. Vehículos

El usuario también podrá escoger entre diferentes tipos de **vehículos** con propiedades claramente diferenciadas en su comportamiento. Por ejemplo, se consideran vehículos rápidos pero frágiles (proto, sedan y classic) frente a vehículos lentos pero robustos (monster, truck y tractor).

Los parámetros que determinan la lógica del vehículo son:

- Masa total, que resulta importante de cara al comportamiento del vehículo en los choques
- Vida total con la que parte el vehículo
- Velocidad máxima que es capaz de alcanzar
- Aceleración
- Capacidad de giro
- Capacidad inicial y máxima para armamento

Combinando las anteriores características se consideran varios vehículos (inicialmente seis), que se resumen en la siguiente tabla y que serán comentados en detalle en los subapartados posteriores:

	Masa	Vida	Velocidad	Aceleración	Cap. de giro	Cap. armas
Proto	80	10	42	55	0.060	5
Sedan	80	15	40	50	0.070	10
Classic	90	20	38	50	0.065	12
Monster	120	25	32	45	0.080	15
Truck	150	30	30	40	0.065	17
Tractor	180	40	25	60	0.090	20

Al igual que con las arenas, se debe facilitar la creación e incorporación de nuevos vehículos, de forma que el videojuego permita ser ampliado incluso por el propio usuario.

6.2.1. Vehículo *Proto*

Es un vehículo rápido pero frágil y de poca capacidad, su uso está reservado para usuarios expertos, con habilidad para la conducción y cuidadosos ante la pérdida de vida, ya que sus ventajas en el juego pueden resultar difíciles de aprovechar.

Figura 10: Vehículo *Proto*

6.2.2. Vehículo *Sedan*

Es un vehículo algo más lento que el anterior, pero de similares características, también está reservado para los usuarios expertos, aunque en este caso su manejo es algo más sencillo dentro del juego.

Figura 11: Vehículo *Sedan*

6.2.3. Vehículo *Classic*

Es un vehículo de velocidad y capacidad media, aunque bastante robusto. Puede considerarse como un vehículo de propósito general para los usuarios medios.

Figura 12: Vehículo *Classic*

6.2.4. Vehículo *Monster*

Es un vehículo más robusto y pesado que el anterior, pero manteniendo cierta agilidad de movimiento. Puede considerarse como una elección intermedia y muy apta para jugadores principiantes, ya que ofrece un manejo sencillo en un vehículo robusto.

Figura 13: Vehículo *Monster*

6.2.5. Vehículo *Camión*

Es un vehículo lento pero robusto y con gran capacidad. Al igual que los vehículos rápidos hay que saber cuando conviene su uso y hay que descubrir sus ventajas dentro del juego.

Figura 14: Vehículo *Camión*

6.2.6. Vehículo *Tractor*

Es un vehículo realmente lento, pero con grandes cualidades, como su gran aceleración. Hay que saber manejarlo y en que momentos es adecuado su uso.

Figura 15: Vehículo *Tractor*

6.3. Elementos

Además de lo comentado, pueden aparecer **elementos** en los escenarios de forma aleatoria, y cada uno de ellos supondrá consecuencias diferentes dentro del juego. Los posibles elementos considerados son:

- **Objetivo**, cuya captura supone incrementar el número de objetivos capturados y el tiempo del contador regresivo.
- **Vida**, cuya captura incrementará la vida del jugador.
- **Cohetes**, cuya captura incrementará el número de cohetes del jugador. Ver el siguiente apartado *6.4. Armas* para más detalles sobre el armamento en los vehículos.
- **Minas**, cuya captura incrementará el número de minas del jugador. Ver el siguiente apartado *6.4. Armas* para más detalles sobre el armamento en los vehículos.
- **Interrogación**, que puede suponer de forma aleatoria cambios en el tiempo del contador regresivo, la vida, los cohetes o las minas del jugador.

Cada arena tendrá preestablecida una lista de posibles ubicaciones para los elementos anteriores, de manera que siempre aparezcan en sitios clave, o de difícil acceso.

En cada momento habrá sobre la arena, a lo sumo, un elemento objetivo y un elemento de cohetes, vida o interrogación.

6.3.1. Elemento *Objetivo*

Supondrá el incremento del contador de objetivos capturados, el incremento de cierta cantidad de tiempo al contador regresivo y la aparición de otro elemento objetivo en otra parte del escenario (desapareciendo el actual).

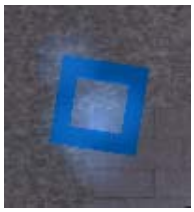


Figura 16: Elemento *Objetivo*

6.3.2. Elemento *Vida++*

Al cruzar con el vehículo sobre este objeto, aumentaremos la vida del mismo, siempre sin superar su vida máxima concreta. En cualquier caso, el objeto desaparecerá del escenario.

Figura 17: Elemento *Vida*

6.3.3. Elemento *Cohete*

Al cruzar con el vehículo sobre este objeto, y siempre que tengamos capacidad suficiente, nos aportará varios cohetes adicionales. En cualquier caso, tengamos o no capacidad suficiente, el objeto desaparecerá del escenario.

Figura 18: Elemento *Cohete*

6.3.4. Elemento *Mina*

Similar al anterior elemento, pero en este caso nos aportará varias minas adicionales. En cualquier caso, el objeto desaparecerá del escenario.

Figura 19: Objeto *Mina*

6.3.5. Elemento *Interrogación*

Al cruzar con el vehículo sobre este objeto supondrá uno de los siguientes eventos:

- Incrementar o decrementar el contador de tiempo regresivo
- Incrementar o decrementar la vida del vehículo
- Incrementar o decrementar el número de cohetes disponibles
- Incrementar o decrementar el número de minas disponibles

Al igual que los anteriores, el objeto desaparecerá del escenario tras tomarlo.

Figura 20: Objeto *Interrogación*

6.4. Armas

La última entidad lógica a considerar son las armas, que pueden utilizarse con diferentes finalidades, ya sea para perjudicar a los contrincantes o para facilitar el acceso a zonas complicadas sin dañar el vehículo.

6.4.1. Arma *Cohete*

Es un arma para actuar a distancia, una vez lanzado sigue una trayectoria recta y es efectivo en el momento que colisione con algún objeto. No se podrá lanzar más de un cohete al mismo tiempo, por lo tanto hay que esperar a que colisione cada cohete antes de poder lanzar el siguiente.

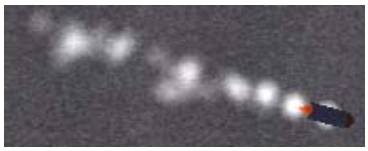


Figura 21: Arma *Cohete*

6.4.2. Arma *Mina*

Es un arma que se sitúa en una posición fija y se hace efectiva cuando algún objeto colisiona con ella. Al igual que con los cohetes, sólo podrá haber una mina por jugador en cada momento, por lo que antes de poner la siguiente habrá tenido que colisionar la anterior.

Figura 22: Arma *Mina*

7. Modos de juego

Tras conocer las cuatro entidades lógicas, podemos detallar ahora el funcionamiento del juego. Se puede escoger entre uno o dos jugadores, por lo que en cada caso tendremos unas posibilidades diferentes.

7.1. Un jugador

Para un único jugador tan sólo es necesario escoger una arena de juego y un vehículo. Al inicio del juego partiremos con:

- Cierta vida y armas (depende del vehículo)
- Un contador regresivo a partir de ciertos segundos (depende de la arena)
- Un marcador de objetivos vacío

En cada momento habrá un único elemento objetivo situado aleatoriamente en la arena. Si no lo vemos, podemos conocer su situación en el mini-mapa (más detalles sobre el GUI en el apartado *8.2. GUI durante el juego*)

La idea es capturar tan rápido cómo sea posible el elemento en pista, de forma que cada vez que capturemos uno:

- Aparecerá el siguiente en otro lugar de la arena
- Incrementaremos el contador regresivo en varios segundos (depende de la arena)
- Incrementaremos el marcador de objetivos en uno

El juego consiste en completar el marcador de objetivos sin que el contador regresivo llegue a cero y sin agotar la vida del vehículo. Es decir, el jugador ganará cuando haya capturado todos los objetivos prefijados de la arena y perderá si el contador regresivo llega a cero o si agota la vida de su vehículo.

En este modo de juego las armas pierden su significado, ya que no existe competencia entre jugadores. Para compensarlo se incluyen ranking de los mejores tiempos por pista, de forma que quedan registrados los jugadores que consiguieron completar cada arena en el menor tiempo posible.

7.2. Dos jugadores

Para dos jugadores es necesario escoger una arena de juego y dos vehículos (uno para cada jugador). En este caso permitimos dos formas de juego diferentes, según se resume a continuación:

7.2.1. Dos jugadores en modo objetivos

Al inicio del juego partiremos con:

- Cierta vida y armas **para cada jugador** (depende del vehículo)
- Un contador regresivo a partir de ciertos segundos (depende de la arena)
- Un marcador de objetivos vacío **para cada jugador**

Este modo de juego sigue la idea anterior para un jugador. Ganará el jugador que antes complete el marcador de objetivos o perderá el jugador que antes agote la vida de su vehículo, con el añadido de las armas disponibles para cada uno.

7.2.2. Dos jugadores en modo lucha

Al inicio del juego partiremos con:

- Cierta vida y armas **para cada jugador** (depende del vehículo)

En este modo de juego desaparecen los conceptos de *marcador de objetivos* y *contador regresivo*, por lo que el único objetivo es acabar con la vida del vehículo oponente, utilizando para ello el armamento que aparezca en pantalla.

8. Interfaz de usuario

El interfaz de usuario tratará de ser lo más simple posible, ofreciendo diferentes opciones cuando se está dentro de una partida (GUI durante el juego) y cuando no se está jugando (GUI general), tal y como se detallan en los subapartados posteriores.

8.1. GUI general

Al inicio del juego, y en general siempre que no estemos en una partida, se mostrará un menú con diferentes opciones según el siguiente mapa de navegación:

- **Nueva partida**
  - Seleccionar uno o dos jugadores
  - Seleccionar el tipo de juego (sólo en el caso de dos jugadores)
  - Seleccionar la arena
  - Seleccionar el vehículo o vehículos (dependiendo si es uno o dos jugadores)
  - Instrucciones de la arena
  - Comienza la partida (ver GUI durante el juego en el siguiente apartado)

- **Opciones**
  - **Video** para configurar los siguientes aspectos del video:
    Tipo de configuración (prefijada o personalizada)
    Pantalla completa (si o no)
    Estadísticas (si o no)
    Resolución (se toman las posibles del dispositivo)
    Filtros de texturas (punto, lineal, anisotrópica)
    Midmapping (si o no)

  - **Sonido** para configurar los siguientes aspectos del sonido:
    Volumen de los efectos de sonido

  - **Juego** para configurar los siguientes aspectos del juego:
    Nombre del jugador
    Controles del juego

- **Salir** Para finalizar el programa.

8.2. GUI durante el juego

El interfaz de usuario durante el juego puede variar en función del tipo de juego elegido, pero por lo general se mostrarán los siguientes marcadores:

1. **Marcador de vida** — Aparecerá un marcador de este tipo por cada jugador en pista. Indica la vida restante respecto del total posible del vehículo. Según se va agotando la vida del vehículo el marcador cambia su color de blanco a rojo.
2. **Marcador de cohetes** — Aparecerá un marcador de este tipo por cada jugador en pista. Indica el número de cohetes restantes respecto la capacidad total del vehículo.
3. **Marcador de minas** — Aparecerá un marcador de este tipo por cada jugador en pista. Indica el número de minas restantes respecto la capacidad total del vehículo.
4. **Marcador de objetivos** — Aparecerá un marcador de este tipo por cada jugador en pista. Indica el número de objetivos capturados y los restantes necesarios para completar el escenario.
5. **Tiempo total** — Indica el tiempo total de juego desde el inicio de la partida. Este dato es importante ya que será prioritario a la hora de clasificar el ranking de los mejores jugadores de la arena.
6. **Contador regresivo** — Indica el tiempo disponible para capturar los objetivos restantes.
7. **Minimapa** — Muestra la geometría de la arena y la situación actual de cada jugador (puntos grises) y del siguiente objetivo (punto azul).
8. **Pausa** — Botón para pausar el juego.

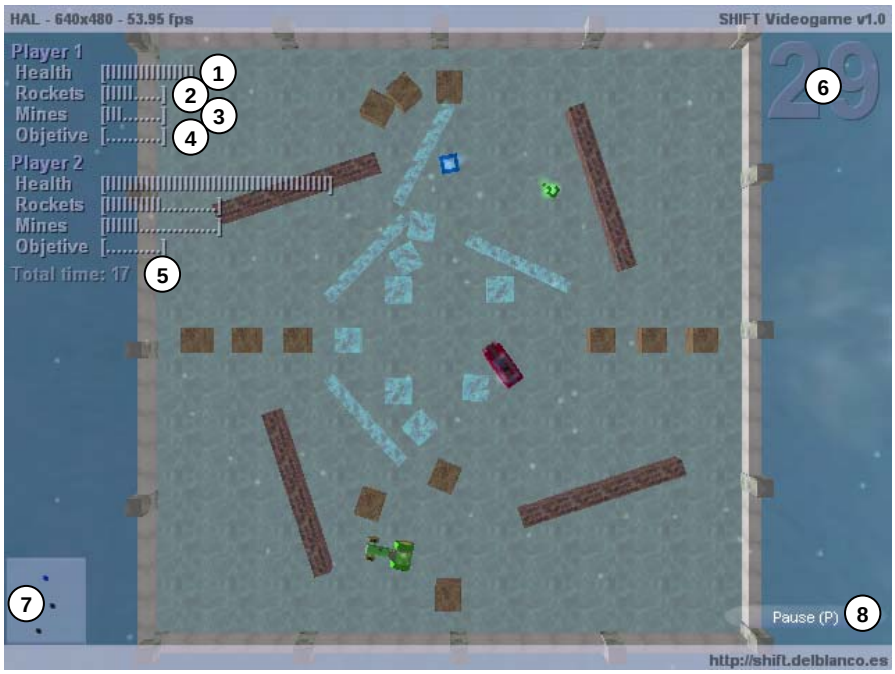


Figura 23: GUI durante el juego

Si durante el juego pulsamos sobre el botón de pausa o presionamos la tecla ESCAPE o 'P', accederemos al menú del modo pausa con las siguientes opciones:

- **Opciones**
  - **Video** para configurar los siguientes aspectos del video:
    Tipo de configuración (prefijada o personalizada)
    Pantalla completa (si o no)
    Estadísticas (si o no)
    Resolución (se toman las posibles del dispositivo)
    Filtros de texturas (punto, lineal, anisotrópica)
    Midmapping (si o no)

  - **Sonido** para configurar los siguientes aspectos del sonido:
    Volumen de los efectos de sonido

  - **Juego** para configurar los siguientes aspectos del juego:
    Nombre del jugador
    Controles del juego

- **Salir** Para finalizar la partida e ir al menú general, pidiendo antes confirmación.

Figura 24: Menú en el modo pausa

9. Controles

Los controles para el GUI y para el juego son los clásicos de cualquier juego de PC. Además, estos controles son fijos, ya que su simplicidad no obliga a parametrizarlos.

9.1. Manejo del GUI

Tanto en el GUI general, como en el GUI durante el juego, las opciones se podrán escoger por medio del ratón y del teclado:

- Con el ratón podemos navegar pulsando cada opción con el botón izquierdo.
- Con el teclado se puede navegar con los cursores y seleccionar mediante la tecla INTRO.

9.2. Manejo del juego

Durante el juego sólo podremos tomar el control del coche por medio del teclado.

Para el **jugador 1**:

- Cursor arriba — Para acelerar el vehículo
- Cursor abajo — Para utilizar el freno de pie, que actúa sobre las cuatro ruedas
- Cursor derecha — Para girar el vehículo hacia la derecha
- Cursor izquierda — Para girar el vehículo hacia la izquierda
- Barra espaciadora — Para utilizar el freno de mano, que actúa sobre las ruedas traseras. Hace que el coche se vuelva inestable, pero lo ladea con rapidez.
- Control derecho — Para lanzar un cohete
- Alt derecho — Para lanzar una mina

Para el **jugador 2**:

- Tecla W — Para acelerar el vehículo
- Tecla S — Para utilizar el freno de pie, que actúa sobre las cuatro ruedas
- Tecla D — Para girar el vehículo hacia la derecha
- Tecla A — Para girar el vehículo hacia la izquierda
- Bloq. Mayúsculas — Para utilizar el freno de mano.
- Control izquierdo — Para lanzar un cohete
- Alt izquierdo — Para lanzar una mina

Controles **comunes**:

- Tecla ‘C’ — Para cambiar la vista de la cámara
- ESC o tecla ‘P — Para pausar el juego y lanzar el menú de juego.

9.3. Vistas de la cámara

Si durante el juego pulsamos la tecla 'C' cambiaremos la vista de la cámara entre las diferentes posibles que se permiten en el juego.

La vista por defecto es la superior, dónde podemos ver el juego según la idea original. La orientación es pareja a la del mini-mapa y la cámara se sitúa siempre en la misma posición del vehículo.

Figura 25: Vista de cámara superior

La siguiente cámara es una vista superior pero que abarca toda la arena. En este sentido la cámara permanece fija, ya que no es necesario que se sitúe en la misma posición que el vehículo. Esta es la única vista permitida para el caso de dos jugadores.

Figura 26: Vista de cámara superior completa

La vista interior del vehículo permite situar la cámara justo encima del capó. Aunque en este caso se pierde la propia naturaleza arcade del juego.

Figura 27: Vista de cámara interior

Otra cámara es la exterior desde un punto fijo, que permite un mayor campo visual, aunque puede resultar incómoda de cara al juego.

Figura 28: Vista de cámara desde un punto fijo

Por último tenemos varias cámaras persecutorias a diferentes alturas, que siguen al vehículo a cierta distancia.

Figura 29: Vista de cámara persecutoria (sin edificios antepuestos)

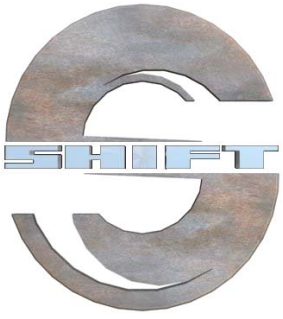
La persecución se implementa de un modo elástico, por lo que al girar, en ocasiones puede suceder que el vehículo quede oculto tras edificios. Para solucionar este problema, en este tipo de vista los objetos altos se hacen semitransparentes, mostrando así el vehículo tras el edificio.

Figura 30: Vista de cámara persecutoria (con edificios antepuestos)

Capítulo 3

Diseño

16 de marzo de 2008



Capítulo 3: Diseño

Tabla de apartados

Lista de figuras

10. Introducción (*diseño*)

SHIFT Videogame se desarrolla a bajo nivel, utilizando únicamente las librerías gráficas DirectX, por lo que la mayor dificultad reside en el diseño y construcción de un motor gráfico.

Según el diseño propuesto para el videojuego, se requiere un planteamiento técnico que aporte realismo, pero sin perder en eficiencia o sencillez. En definitiva, los criterios básicos a considerar son:

- **Eficiente**
  Los videojuegos son una rama del software con grandes exigencias de procesamiento, por lo que la eficiencia será uno de los aspectos mas complicados a tener en cuenta. Debemos construir algoritmos computacionalmente sencillos, optimizando las zonas de código críticas y cuidando el uso de la memoria.

- **Sencillo**
  Sin embargo, en algunas ocasiones será necesario sacrificar la eficiencia en pro de una mayor sencillez del código, más aún considerando que la orientación del proyecto es más académica que profesional.

- **Comprensible**
  Pero aparte de construir un motor gráfico sencillo, deberá ser comprensible. De esta forma, el proyecto podrá aprovechar de los beneficios que le aporta estar publicado bajo una licencia copyleft. Por tanto, se debe garantizar a nivel de diseño, programación y documentación, que el proyecto pueda ser retomado por terceras personas.

- **Genérico**
  Pero además, los aspectos técnicos deben basarse en un diseño genérico, de forma que no haya dependencias o cualidades específicas para este videojuego en cuestión.

- **Independiente**
  De forma similar, se debe garantizar un aislamiento entre el diseño técnico y el diseño del videojuego, de forma que el motor gráfico sea independiente de las cualidades de SHIFT como videojuego.

- **Modular**
  Se debe respetar la modularidad en todos los elementos que integren el motor gráfico, por lo que una de las claves del diseño va a ser la orientación a objetos.

- **Reutilizable**
  Acorde con los criterios anteriores, el motor gráfico debe diseñarse pensando en su posible reutilización o escalabilidad.

Teniendo en cuenta estos criterios, la propuesta de este proyecto debe pensarse cómo un motor gráfico (parte técnica) y una aplicación para utilizarlo (videojuego SHIFT) y en todo caso su separación debe quedar patente durante todo su diseño.

Acorde a la modularidad del sistema, el motor gráfico se dividirá en varios elementos más sencillos para tratar cada uno se los aspectos principales del videojuego. En general tendremos:

- **Motor físico**
  Para gestionar la geometría, texturas y parámetros físicos de los objetos.

- **Motor de colisiones**
  Para gestionar las colisiones entre los objetos físicos que tengan esta cualidad.

- **Motor de sonidos**
  Para manipular los sonidos asociados a los objetos físicos.

- **Motor de utilidades**
  Para mantener la cámara, la luz, la configuración de la aplicación, interfaz de usuario y un sistema de entrada/salida mediante XML.

- **Motor de partículas**
  Para generar sistemas de partículas que modelen diferentes efectos en la escena.

- **Motor lógico**
  Es el módulo adicional que implementará la lógica propia del videojuego, siendo esté la única parte no genérica para el motor físico.

A continuación, en los siguientes apartados veremos la interacción y descripción en detalle de cada uno de estos módulos que componen el motor gráfico del videojuego.

11. Esquema general

En el siguiente diagrama se muestra una primera visión modular del motor gráfico, dónde se puede tener una primera impresión del ámbito de cada módulo y las interacciones que tienen entre ellos:

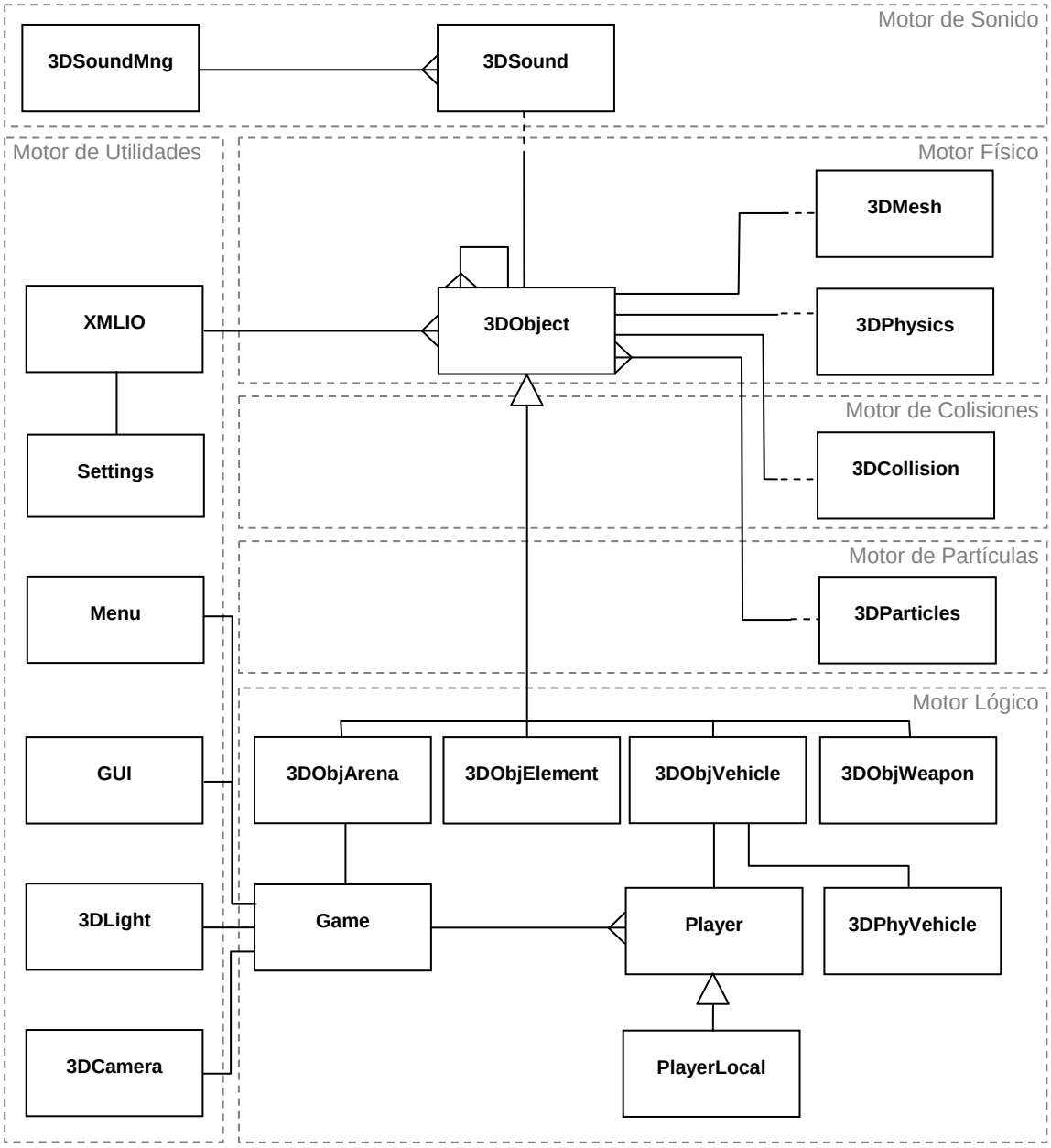


Figura 31 : Esquema modular del motor gráfico

12. Detalle de los módulos

12.1. Motor físico

Puede considerarse el módulo principal del motor gráfico. Permite gestionar los objetos en sí, manteniendo su geometría, texturas y parámetros físicos.

Se compone a su vez de las siguientes clases:

- **3DObject**
  Clase para abstraer el concepto de objeto dentro del motor gráfico. Es la clase que centraliza todas las propiedades técnicas y lógicas de cada elemento del videojuego

- **3DMesh**
  Permite gestionar la geometría y texturas del objeto.

- **3DPhysics**
  Permite mantener los datos físicos del objeto, como son posición, escalado, rotación velocidades, aceleraciones y fricciones lineales y angulares.

12.2. Motor de colisiones

Para gestionar las colisiones entre los objetos físicos que tengan esta cualidad, para ello se abstraen las colisiones con la clase:

- **3DCollision**
  Clase que permite implementar toda la lógica de colisiones entre objetos, como son la detección, recolocación y el intercambio de parámetros físicos. Con el fin de simplificar el motor gráfico, todas las colisiones se reducirán al caso rectángulo-rectángulo en dos dimensiones.

12.3. Motor de sonidos

Para manipular los sonidos, situándolos en el entorno 3D bajo la misma ubicación que los objetos a los que están asociados.

Se compone a su vez de las siguientes clases:

- **3DSoundMng**
  Gestor de todos los elementos de sonido de la aplicación.

- **3DSound**
  Clase para abstraer el buffer de cada uno de los sonidos de la aplicación.

12.4. Motor de utilidades

El motor de utilidades permite mantener la cámara, la luz, la configuración de la aplicación, interfaz de usuario y un sistema de carga externa de modelos mediante XML.

Se compone a su vez de las siguientes clases:

- **XMLIO**
  Permite cargar y almacenar de forma persistente datos y estructuras de la aplicación con formato XML. Se usará para los modelos, la configuración, el ranking, etc.

- **Settings**
  Esta clase implementa la gestión de parámetros personalizables por parte de los usuarios, tales como la configuración de video, sonidos, estilos de juego, etc.

- **Menu**
  Esta clase tan sólo hace llamada a las instancias apropiadas en cada momento, en función de la pantalla en la que se encuentre el interfaz de usuario.

- **GUI**
  Esta clase implementa el interfaz de usuario, incluyendo toda la lógica interna de navegación de menús y opciones de usuario.

- **3DLight**
  Clase para definir parámetros de luz ambiente, especular y difusa dentro de la aplicación. Para facilitar la programación no se implementan focos direccionales de luz, por lo que sólo se encuentra configurada la luz global de la escena.

- **3DCamera**
  Clase que abstrae el concepto de cámara para el dispositivo gráfico, con todos sus parámetros de foco, posición y rotación. Dentro de esta clase se implementan los algoritmos de seguimiento de objetos, cambios de vista, etc.

12.5. Motor de partículas

Permite gestionar sistemas de partículas en la escena. Utilizaremos estos sistemas para modelar diferentes fenómenos con mayor realismo, tales como el humo de los vehículos, nieve, chimeneas, explosiones, etc.

- **3DParticles**
  Gestiona sistemas de partículas con varios parámetros físicos que permitirán adaptarlos a los resultados necesarios en cada momento.

12.6. Motor lógico

Adicionalmente el sistema constará de un módulo que implementará la lógica del videojuego, siendo esta la parte personalizada al juego dentro del motor gráfico.

Se compone a su vez de las siguientes clases:

- **3DObjArena**
  Tipo de objeto con características especiales para contener una arena concreta, incluyendo gran cantidad de parámetros de juego que implementan la mayor parte de la lógica.

- **3DObjVehicle**
  Tipo de objeto con parámetros específicos para implementar los vehículos, incluyendo física y sonidos adicionales para ellos.

- **3DObjElement**
  Tipo de objeto con características especiales para contener los elementos aleatorios que aparecen en la arena, tales como cohetes, vida, interrogación y el elemento objetivo.

- **3DObjWeapon**
  Tipo de objeto con parámetros específicos para implementar las armas del juego, tales como minas o cohetes.

- **3DPhyVehicle**
  Clase que modela el comportamiento físico del vehículo, ya que dicho comportamiento es diferente al del resto de los objetos de la escena.

- **Game**
  Clase que gestiona toda la información referente a una partida en curso, desde el tipo de juego, hasta la arena seleccionada y los jugadores que participan.

- **Player**
  Clase que gestiona todos los datos de cada uno de los jugadores de la partida en curso.

- **PlayerLocal**
  Tipo de jugador local en el equipo. Esta clase hereda de la anterior, permitiendo así implementar otra clase paralela PlayerRemote que gestione jugadores remotos en el caso de ampliar el juego con un modo multijugador y motor de red.

13. Bucle principal

Como hemos visto, el motor gráfico se compone de diferentes clases que implementan cada uno de los aspectos relevantes del mismo, tales como la física, las colisiones, los sonidos, la red, etc.

Cada una de estas clases compartirá un framework de cuatro operaciones básicas, que serán implementadas con las tareas adecuadas en cada caso:

- **Initialize** Esta operación sólo se hace una vez antes de utilizar cada instancia. Las operaciones habituales dentro de esta función son las de inicialización, carga externa y registro de la instancia en el motor, tras haberlo hecho el objeto puede entrar a formar parte del motor gráfico.
- **Update** Esta operación se hace en cada iteración del bucle principal del motor gráfico. En ella se calcula el nuevo estado de cada instancia respecto la última iteración y el tiempo que pasó desde entonces. Es decir, si un objeto se encontraba en movimiento y pasaron 0,05 segundos respecto la anterior iteración, se deberá modificar su ubicación considerando ese tiempo y sus parámetros físicos de velocidad, dirección, rozamiento, etc.

  La idea es que el entramado de clases anterior interactuará entre sí para ir conformando el nuevo estado de la escena respecto el estado anterior.
- **Render** Esta operación se hace tras la anterior, en cada iteración del bucle principal del motor gráfico. En ella, cada instancia comunicará al dispositivo de video las modificaciones necesarias para representar la escena en la siguiente imagen renderizada.
- **Release** Función contraria a la de inicialización, para dar de baja una instancia del motor gráfico. Sólo se ejecutará una vez, para destruir el objeto y decrementar el contados de referencias interno del dispositivo.

Gráficamente, el diagrama de flujo refleja el uso de las anteriores funciones para cada una de las instancias que haya en la escena del motor gráfico:

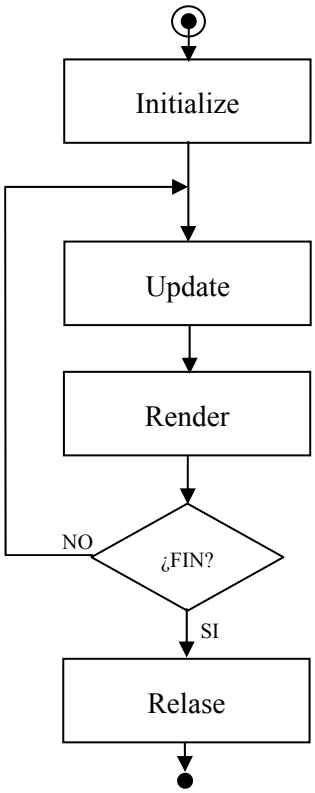


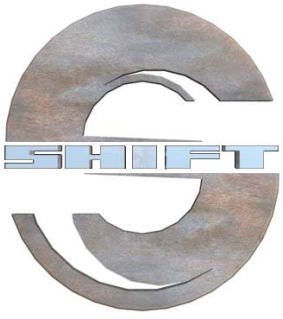
Figura 32 : Diagrama de flujo de cada elemento del motor gráfico

Desde un punto de vista simplista, el motor gráfico es un gran bucle que ejecutará las tareas de actualización y dibujado con la mayor rapidez posible.

Capítulo 4

Diseño del motor físico

16 de marzo de 2008



Capítulo 4: Diseño del motor físico

Tabla de apartados

Lista de figuras

14. Introducción (*motor físico*)

El módulo físico puede considerarse el módulo principal del motor gráfico. Permite gestionar los objetos en sí, definiendo su jerarquía de construcción, ubicación, geometría, texturas y parámetros básicos.

Dentro del esquema general del motor gráfico, el módulo físico queda limitado por el ámbito rayado:

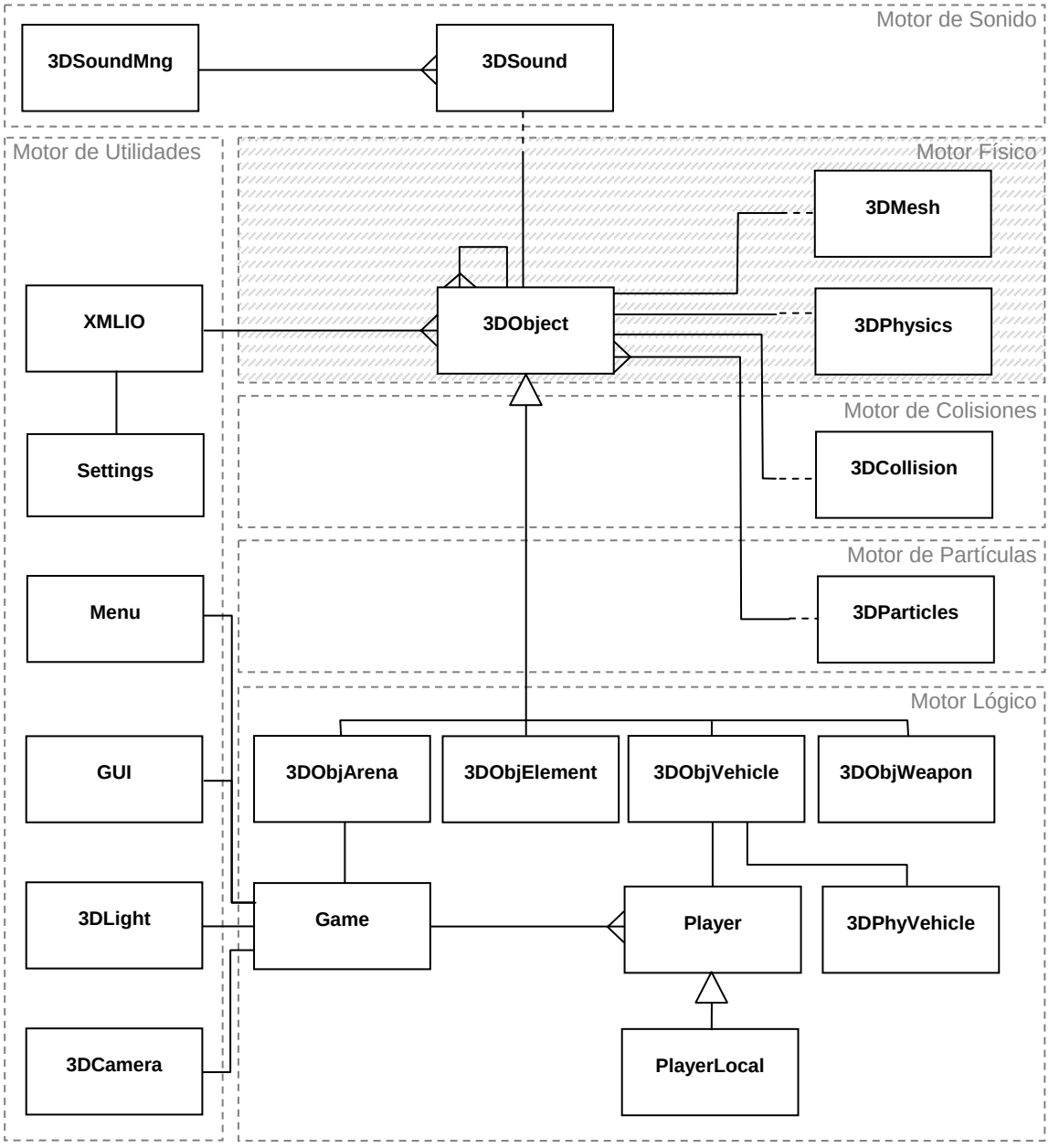


Figura 33 : Límites del motor físico

Cómo refleja en el diagrama, el módulo físico consta de las siguientes clases:

- **3DObject**
  Clase para abstraer el concepto de objeto dentro del motor gráfico. Es la clase que centraliza todas las propiedades técnicas y lógicas de cada elemento del videojuego

- **3DMesh**
  Permite gestionar la geometría y texturas del objeto.

- **3DPhysics**
  Permite mantener los datos físicos del objeto, como son posición, escalado, rotación velocidades, aceleraciones y fricciones lineales y angulares.

Por tanto este módulo implementa los objetos en sí, su construcción jerarquizada, su ubicación en la escena, geometría, texturas y parámetros físicos que tendrá en cada momento de la escena.

A continuación veremos en detalle cada uno de estos aspectos controlados por el motor físico.

15. Parámetros de la clase *3DObject*

La clase que abstrae el concepto de objeto es el centro de un modelo radial, dónde el resto de instancias están asociadas de algún modo a cada objeto.

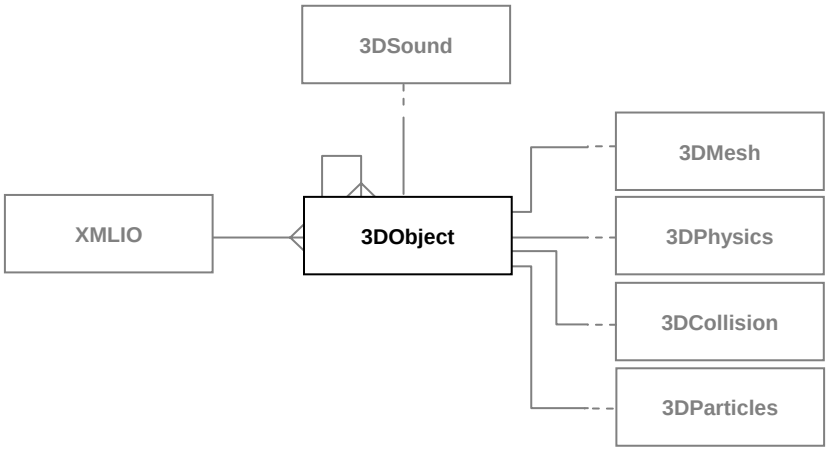


Figura 34 : Clase "*object*" como centro de un modelo radial

Los objetos constituyen las piezas unitarias más pequeñas manejadas por el interfaz gráfico, piezas que tendrán que ser transformadas y renderizadas por separado para componer la escena. El motor físico está diseñado para establecer una composición jerárquica, de forma que varios objetos pueden componer otros más complejos, y cada uno de ellos por separado o todos ellos en conjunto pueden mantener diferentes propiedades.

Es decir, cada objeto se puede construir jerárquicamente en base a otros más pequeños, dónde en cada nivel de la jerarquía se van aportando las propiedades comunes de los objetos que están por debajo. Esta estructuración ofrece una gran versatilidad para el resto del motor gráfico, ya que si desde un módulo se actúa sobre cierto nivel de la jerarquía, se estará actuando sobre todas sus dependencias.

Por ejemplo, un ventilador es un objeto del videojuego, compuesto por dos objetos gráficos más simples, la carcasa y las aspas, cada uno de ellos con propiedades específicas y en conjunto con algunas características en común.

El aspa tendrá una velocidad angular constante y un sonido asociado referente a su movimiento rotativo.

Figura 35 : Objetos simples (carcasa y aspas)

En conjunto, carcasa y aspa, serán colisionables, tendrán una ubicación determinada (posición rotación y escalado) y una física concreta (fricciones lineales y angulares).

Figura 36 : Objeto compuesto (ventilador)

Gráficamente, la jerarquía de instancias construida para el ventilador tendrá la siguiente forma:

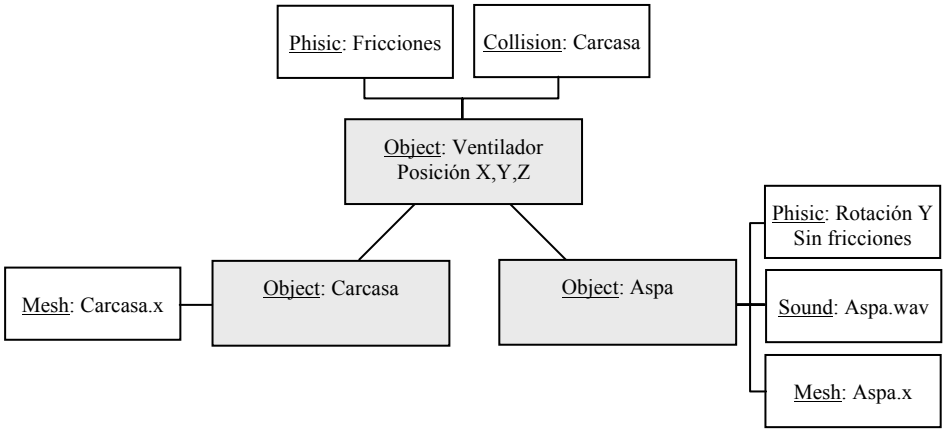


Figura 37 : Diagrama jerárquico de instancias para el ventilador

En adelante podremos hacer referencia a cada objeto específico por separado o a los dos en conjunto. Es decir, si colisionamos sobre la instancia del ventilador, estaremos moviendo aspa y carcasa conjuntamente, y al moverse actuarán las fricciones lineales y angulares asociadas. Sin embargo, en el movimiento rotativo del aspa no intervienen esas fricciones, ya que esta instancia tiene sus características físicas propias que prevalecen sobre las del objeto padre.

Resumiendo, los objetos definen sus propiedades de una forma jerárquica, dónde en cada nivel podremos definir de forma opcional:

- **Ubicación**
  Posición, rotación y escalado, mediante una matriz de conversión.
- **Geometría y Texturas**
  Geometría y texturas asociadas.
- **Física**
  Velocidades, aceleraciones y rozamientos lineales y angulares, así como la masa del objeto.
- **Colisiones**
  Área de colisión asociada.
- **Sonidos**
  Buffer de sonido que se debe ubicar en el origen del objeto.
- **Sistemas de partículas**
  Sistemas de partículas para modelar diferentes fenómenos.

16. Parámetros de la clase *3DMesh*

Esta clase gestiona tres buffer por cada instancia:

- **Geometría** Como un conjunto de vértices, uniones y normales.
- **Materiales** Como colores sólidos asociados a ciertos vértices.
- **Texturas** Como un buffer de la imagen y un mapeado a vértices.

Una vez en memoria, cada uno de los tres buffer tendrá acceso de sólo de lectura, ya que la geometría, materiales y texturas de los objetos permanecerán constantes durante la existencia de cada objeto.

Este hecho permitirá tener una sola instancia de cada buffer, a modo de singleton, y que la clase 3DMesh actúe como una factoría simple de esos recursos. Es decir, al cargar varios modelos, el motor detectará automáticamente los elementos comunes cargando una única instancia en memoria de cada tipo, ahorrando así tiempo de lectura de disco y ocupación en memoria.

Será la propia ruta de cada fichero la que identificará unívocamente cada recurso, ya que la geometría y materiales se cargan desde ficheros DirectX (.X) y las texturas se cargan desde ficheros de imagen (.JPG o .BMP).

Esta idea aporta gran eficiencia a la aplicación, ya que podremos cargar gran cantidad de objetos en escena, leyendo y manteniendo en memoria unos pocos ficheros, todo ello de forma transparente para el resto de módulos de la aplicación.

Por ejemplo, siguiendo con el diagrama de instancias anterior, si añadimos un nuevo ventilador, se compartirán las instancias de la geometría del aspa, la carcasa.

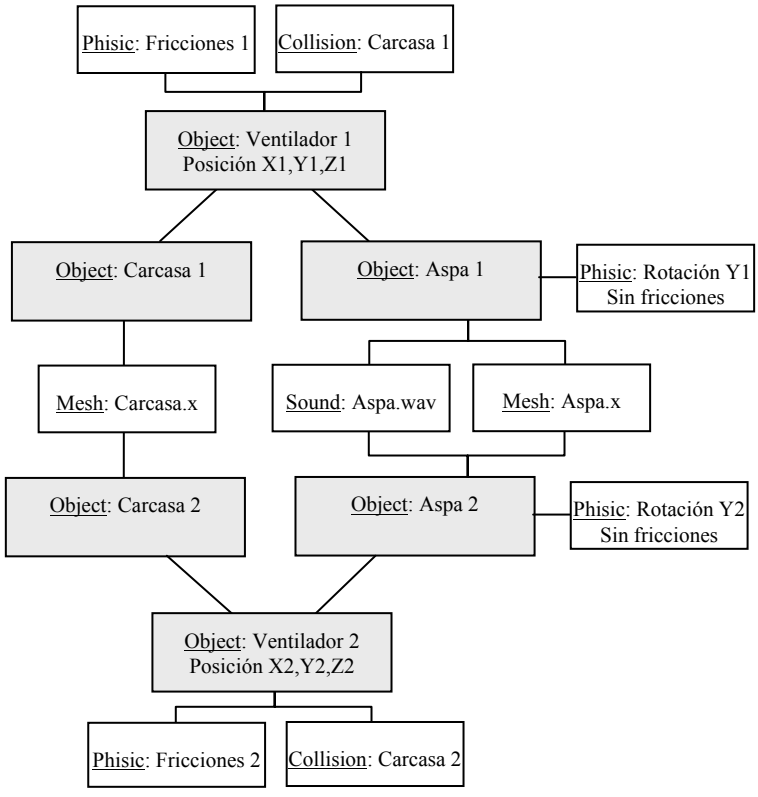


Figura 38 : Diagrama de instancias para dos ventiladores

17. Parámetros de la clase *3DPhysic*

Esta clase centraliza los datos físicos del objeto, como son:

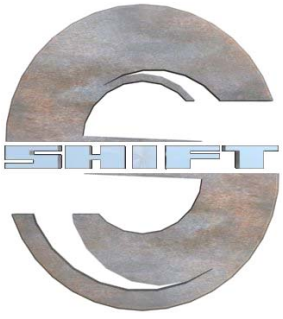
- Módulo de la velocidad lineal.
- Módulo de la aceleración lineal.
- Módulo del rozamiento lineal.
- Vector normalizado de la dirección de desplazamiento lineal.
- Velocidad angular en cada uno de los tres ejes.
- Aceleración angular en cada uno de los tres ejes.
- Rozamiento angular en cada uno de los tres ejes.

En adelante, el resto de módulos del motor gráfico actuarán sobre estos datos, modificándolos en cada iteración que el motor gráfico realiza sobre las instancias manejadas.

Capítulo 5:

Diseño del motor de colisiones

16 de marzo de 2008



Capítulo 5: Diseño del motor de colisiones

Tabla de apartados

Lista de figuras

18. Introducción (*motor de colisiones*)

Una de las tareas más importantes del motor gráfico es el sistema de detección de colisiones, ya que supondrá el mayor consumo computacional del programa y de él dependerá no sólo el rendimiento, sino la jugabilidad y el realismo final de cara al usuario

Dentro del esquema del motor gráfico, el módulo de colisiones queda limitado por el ámbito rayado:

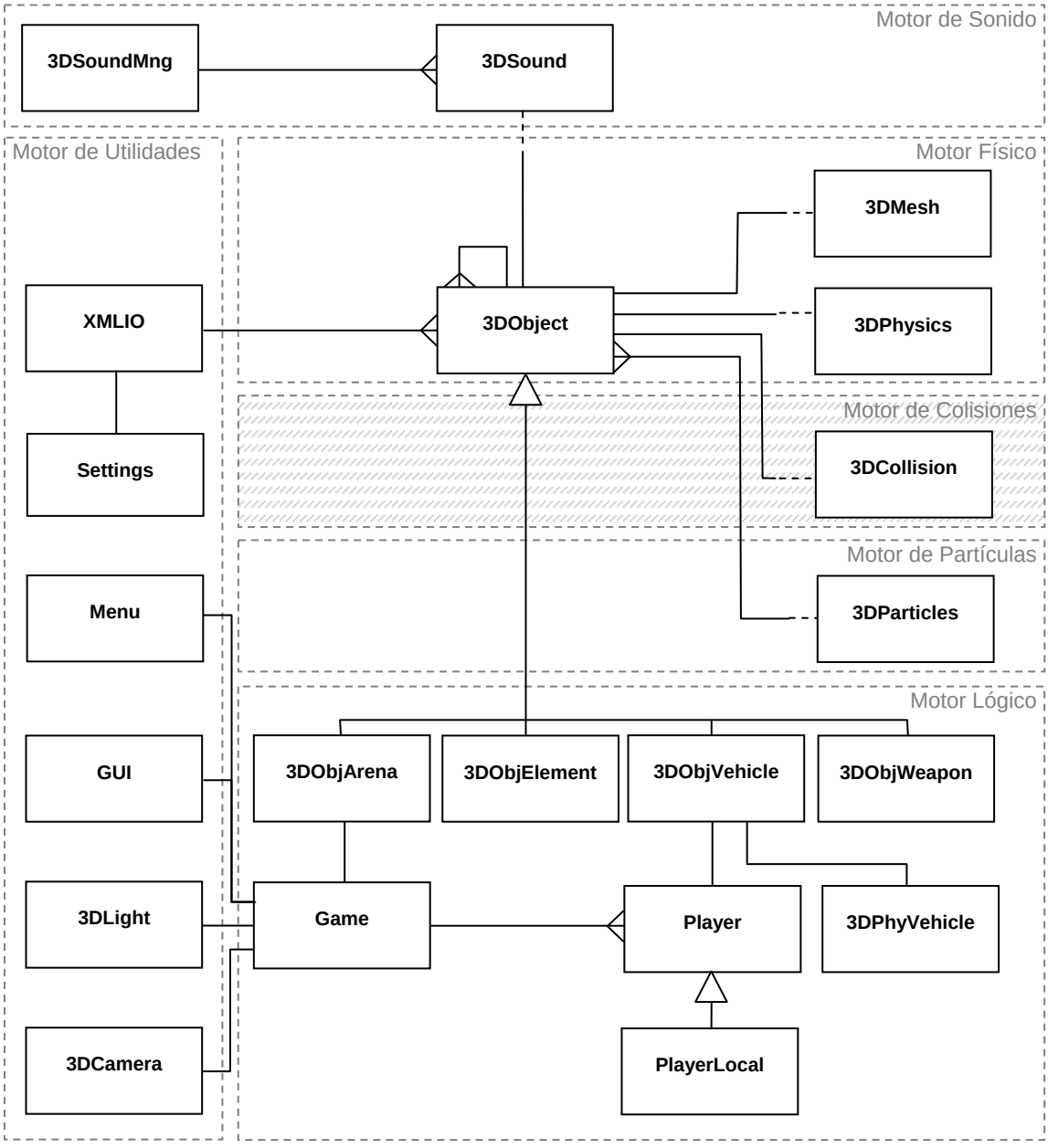


Figura 39 : Límites del motor de colisiones

Cómo refleja en el diagrama, el módulo de colisiones se abstrae con una única clase:

- **3DCollision**
  Clase que permite implementar toda la lógica de colisiones entre objetos, como son la detección, recolocación y el intercambio de parámetros físicos.

A continuación veremos en detalle cada uno de estos aspectos controlados por el motor de colisiones.

19. Parámetros de la clase *3DCollision*

19.1. Introducción

Desde un punto de vista técnico, el videojuego es en realidad un proceso iterativo continuo, en el que tras cada ejecución se actualizan los parámetros de la escena, considerando para ello el tiempo transcurrido desde la anterior iteración.

Actualizar la escena conlleva recalcular las nuevas ubicaciones de los objetos, según su física o comandos de usuario. Tras ello, algunos objetos pueden resultar superpuestos por lo que será necesario modificar sus propiedades para evitar incoherencias físicas.

El motor de colisiones supone el mayor consumo computacional, pero además constituye el módulo más complejo de cara al diseño e implementación, teniendo un carácter más analítico que técnico.

El sistema de colisiones desarrollado toma la simplicidad como principal y único criterio de diseño, ya que no se pretende realizar un algoritmo avanzado. Siguiendo esta idea, se consideran varias simplificaciones:

- Aunque el resto del motor gráfico está basado en tres dimensiones, las colisiones se reducen a un plano ficticio, llamado plano de colisión, por lo que sólo se consideran **dos dimensiones**.

- La proyección de los objetos colisionables modelados tienen una geometría rectangular, o en todo caso puede descomponerse en varias rectangulares. Por este motivo, sólo se considera el tipo de colisión **rectángulo-rectángulo**.

- Por último, y como consecuencia de las dos simplificaciones anteriores, **no todos los objetos de la escena serán colisionables**, ya que no todos los objetos cortarán el plano de colisión.

Tras estas premisas podemos hacer una clasificación de los objetos de la escena en función del tratamiento que van a tener dentro del proceso de colisiones:

- **Objetos colisionables**, cruzan el plano ficticio de colisión, formando en él un área rectangular llamado *área de colisión*. Estos objetos implementan lógica de colisiones y en función de su naturaleza física se clasifican del siguiente modo:

    - **Objetos fijos**, como por ejemplo los edificios, caracterizados por tener una física constante. Estos objetos intervienen en el proceso pero no pueden ser modificados.

    - **Objetos móviles**, como por ejemplo los vehículos, que intervienen en el proceso y además pueden ver modificados sus parámetros físicos.

- **Objetos no colisionables**, están a un nivel diferente del plano de colisión, por lo que serán excluidos del proceso aquí descrito.

La clasificación anterior se resume mediante el siguiente esquema:

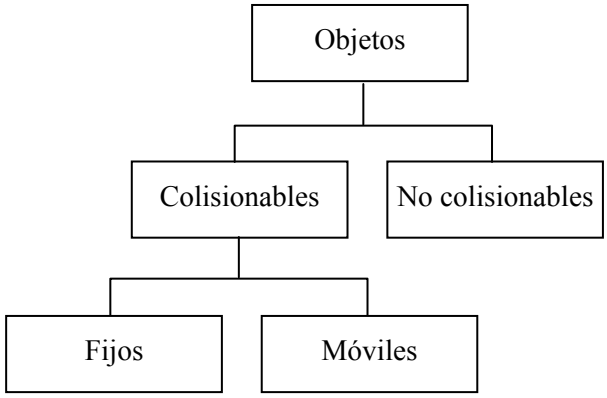


Figura 40 : Clasificación de los objetos de la escena

En todo momento se tienen identificados los objetos colisionables, sus áreas rectangulares de colisión y su tipo de movilidad física.

En los apartados posteriores se van a diseñar los algoritmos necesarios para incorporar los siguientes procedimientos de los que consta el motor de colisiones:

- **Cálculo del área de colisión** Al instanciar cada objeto
- **Sistema de detección de la colisión** Por cada iteración y par de objetos *colisionables*
- **Recolocación de los objetos** Por cada iteración y par de objetos *colisionados*
- **Intercambio de parámetros físicos** Por cada iteración y par de objetos *colisionados*

19.2. Cálculo del área de colisión

Como se verá más adelante, la detección de la colisión se hará en dos niveles, lo cual supondrá un ahorro considerable en el coste del proceso. Para permitir esa detección doble, es necesario calcular dos áreas de colisión por cada objeto colisionable:

- Un **rectángulo** de defina el área exacto de colisión (definido por cuatro puntos).
- Un **círculo** que englobe al anterior rectángulo (definido por un radio).

Para calcular el **rectángulo**, nos situamos en el sistema de referencia del objeto y recorremos los vértices de su geometría. Tomaremos el máximo y mínimo valor de las posiciones Z y X de cada punto de su geometría. Una vez determinados estos cuatro valores resulta sencillo determinar las posiciones (Z, X) de cada uno de los cuatro vértices del área colisionable.

Para calcular el **círculo**, tan sólo hay que tomar el mayor radio desde el centro del objeto a cada uno de los cuatro vértices del rectángulo anterior. De esta forma calcularemos el menor círculo centrado en el objeto y que contenga su rectángulo de colisión.

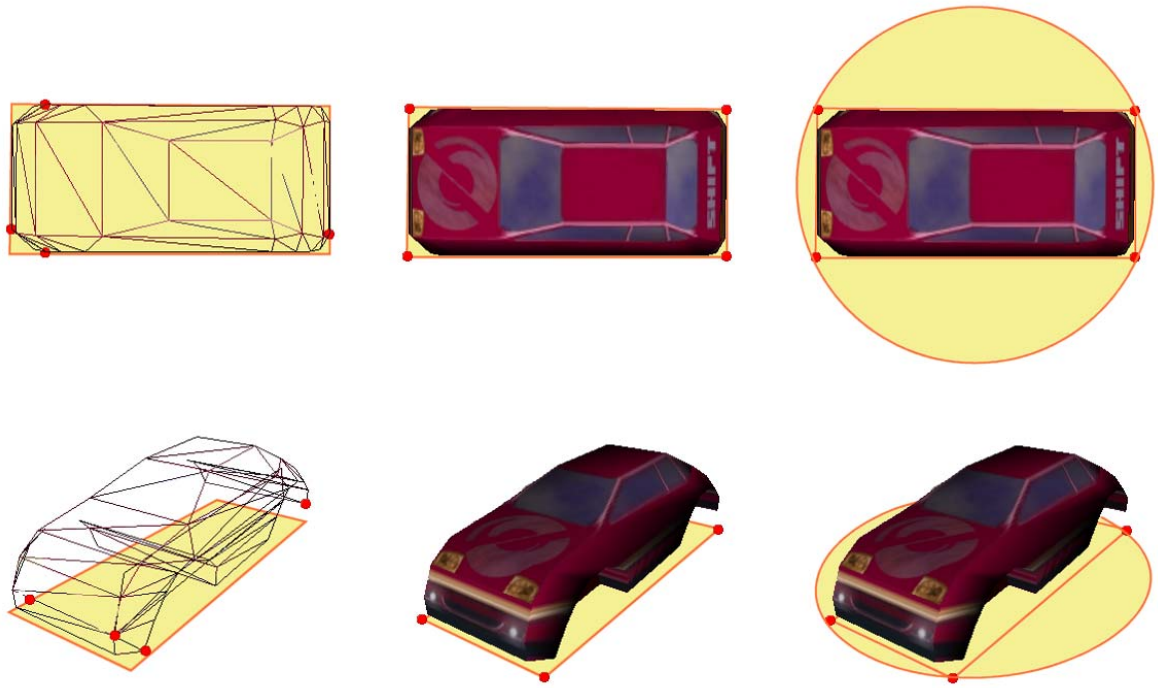


Figura 41 : Cálculo del área de colisión en un objeto

Sólo será necesario calcular estas áreas de colisión una vez por cada objeto, pudiendo realizarse el proceso durante la carga del mismo, al instanciar la clase *3Dcollision*.

19.3. Detección de la colisión

En cada iteración del proceso gráfico, la detección de colisión se evalúa por cada par de objetos colisionables en escena, por ejemplo:

- Con **3** objetos colisionables (A,B,C) hacemos **3** comprobaciones por iteración
  - Comprobamos **A - B**
  - Comprobamos **A - C**
  - Comprobamos **B - C**
- Con **4** objetos colisionables (A,B,C,D) hacemos **6** comprobaciones por iteración
  - Comprobamos **A - B**
  - Comprobamos **A - C**
  - Comprobamos **A - D**
  - Comprobamos **B - C**
  - Comprobamos **B - D**
  - Comprobamos **C - D**
- Con **n** objetos colisionables hacemos $\frac{n \cdot (n-1)}{2}$ comprobaciones por iteración.

Por lo tanto, la detección de colisiones es de orden $O(n^2)$ respecto el número de objetos colisionables, por lo que hay que prestar especial atención de cara a la optimización en este punto.

Vamos a describir lo que supondría una comprobación genérica, por ejemplo al evaluar la posible colisión del objeto **A** respecto el objeto **B**.

La detección se hará de forma progresiva mediante tres pasos, siguiendo un orden de complejidad ascendente en cada uno de ellos:

- Primero se descartan los casos triviales:
  - Objetos **A** y **B** nunca colisionan si son el mismo objeto.
  - Objetos **A** y **B** nunca colisionan si uno de los dos está deshabilitado
  - Objetos **A** y **B** nunca colisionan si ninguno de los dos modificó su matriz de transformación desde la última iteración.
- Si se ha superado el primer paso, se evalúa la colisión entre los **círculos de colisión** de los objetos. Esta comprobación dejará sólo aquellos pares de objetos con alta probabilidad de colisión
- Por último, si se han superado los dos pasos anteriores, se evalúa si hay colisión entre los **rectángulos de colisión** de los objetos, siendo este último el proceso más costoso.

Por lo tanto, obviando los casos triviales que son directos, tenemos dos algoritmos de detección:

- Detección círculo-circulo
- Detección rectángulo-rectángulo

19.3.1. Detección círculo-círculo

Este proceso es sencillo tanto en concepto como de forma computacional.

Tomamos los siguientes datos de los objetos en el SRU (sistema de referencia universal):

- Radio del objeto **A**, representado cómo r_1
- Radio del objeto **B**, representado cómo r_2
- Longitud del segmento que une los centros de los dos objetos $\|d\|$

Si la suma de los dos radios es mayor que la longitud del segmento que une los centros de los dos objetos, entonces hay colisión de los círculos, en caso contrario no.

En la siguiente figura no hay colisión de los círculos y por tanto tampoco de los objetos, ya que están contenidos.

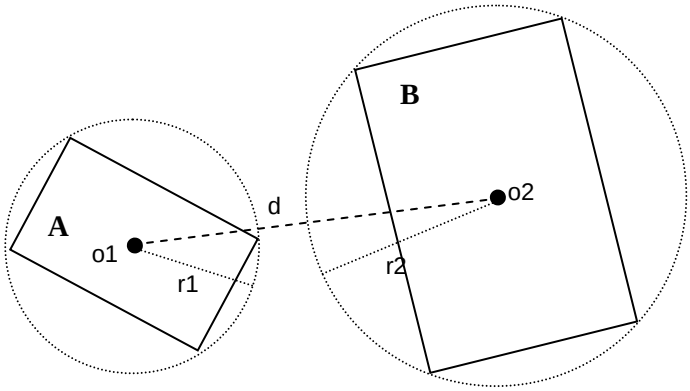


Figura 42 : Evaluando colisión (círculos sin colisión, objetos sin colisión)

En esta otra tenemos colisión de los círculos, sin embargo los objetos no están colisionados.

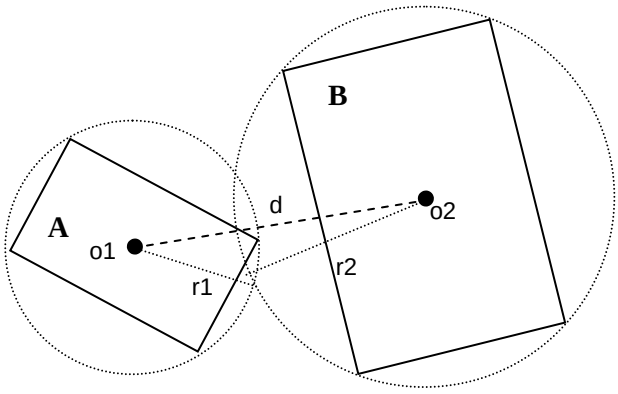


Figura 43 : Evaluando colisión (círculos con colisión, objetos sin colisión)

Por último, en esta tercera figura tenemos el caso en el que tanto círculos como objetos están colisionados:

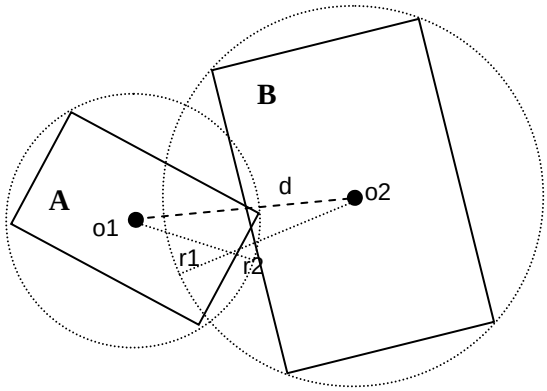


Figura 44 : Evaluando colisión (círculos con colisión, objetos con colisión)

Tras este proceso podemos afirmar:

- Si hay colisión de los círculos no podemos asegurar nada (tendremos que evaluar rectángulos)
- Si no hay colisión de los círculos podemos asegurar que no hay colisión entre los objetos.

19.3.2. Detección rectángulo-rectángulo

Este proceso es algo más costoso que el anterior, pero determina de forma unívoca si hubo colisión.

Para que la detección sea correcta, se debe realizar desde dos sistemas de referencia, primero se hará desde el SRO del objeto B y luego desde el SRO del objeto A.

En el primer caso, si queremos evaluar la colisión respecto el SRO de B, tenemos que empezar llevando todos los puntos a ese sistema de referencia, por lo que:

- Para el objeto **B**, mantenemos en su SRO los cuatro vértices del área de colisión calculado durante la carga, estos son los cuatro valores: X_{max}, X_{min}, Z_{max} y Z_{min}.
- Para el objeto **A**, hay que llevar sus cuatro vértices del área de colisión del al SRO del objeto B. Para ello hay que aplicar a cada vértice la matriz de transformación del objeto A y luego la matriz transformación inversa del objeto B, resultando los valores: $(X_1, Z_1)…(X_4, Z_4)$

En este momento ya tendríamos los ocho puntos en el mismo sistema de referencia, tal y cómo se indica en la figura:

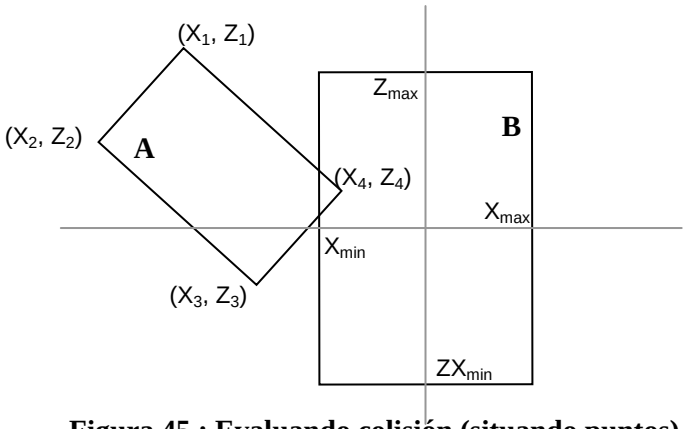


Figura 45 : Evaluando colisión (situando puntos)

A continuación sólo queda comprobar si alguno de los cuatro vértices transformados del objeto A tiene sus componentes X y Z en los rangos $[X_{max}, X_{min}]$ y $[Z_{max}, Z_{min}]$ respectivamente. En ese caso (y sólo en ese caso) habrá colisión del objeto A respecto el objeto B.

Como vemos en la imagen, en el ejemplo propuesto es el punto 4 el que cumple ($X_{max} \geq X4 \geq X_{min}$) y además ($Z_{max} \geq Z4 \geq Z_{min}$) por lo que hemos detectado la colisión.

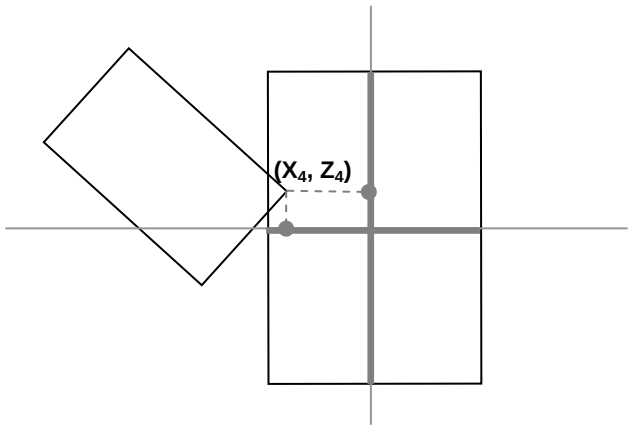


Figura 46 : Evaluando colisión (evaluando puntos)

Si no hubiésemos encontrado colisión respecto el SRO de B, deberíamos realizar la misma comprobación respecto el SRO de A. En caso de no encontrar colisión en ninguno de los dos sistemas de referencia, podemos afirmar que no existe colisión entre los objetos.

Este método es bastante eficiente, ya que por cada comprobación:

- Se hacen 2 inversas de matrices (orden 3)
- Se hacen 4 multiplicaciones de matrices (orden 3)
- Se hacen 8 multiplicaciones de vector por matriz (orden 3)
- Se hacen 32 comparaciones de números reales

Hay que considerar que la inversa y las multiplicaciones de matrices se harán a través del dispositivo gráfico DirectX, por lo que esta carga de proceso la soportará (en la medida de lo posible) la tarjeta gráfica, descargando así de responsabilidad a la CPU.

A continuación vamos a ver un ejemplo completo de detección de colisión entre dos objetos A y B bajo dos situaciones diferentes:

- **Sin colisión entre objetos A y B**
- **Con colisión entre objetos A y B**

Supongamos dos objetos colisionables, A y B, cuyas áreas de colisión no se superponen, por lo la comprobación debería ser en este caso negativa.

Primero hacemos la comprobación de A respecto B, vemos que ningún vértice de A está dentro de los rangos X y Z que define el área de B:

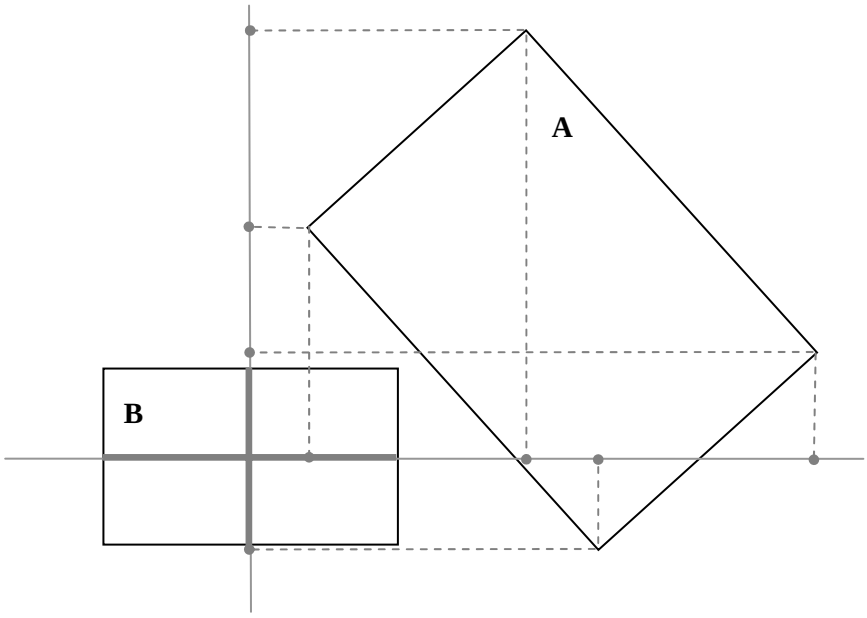


Figura 47 : Detección A-B sin colisión (respecto SRO de B)

Después hacemos la comprobación de B respecto A, vemos que tampoco hay vértices de B dentro de los rangos X y Z que define el área de A:

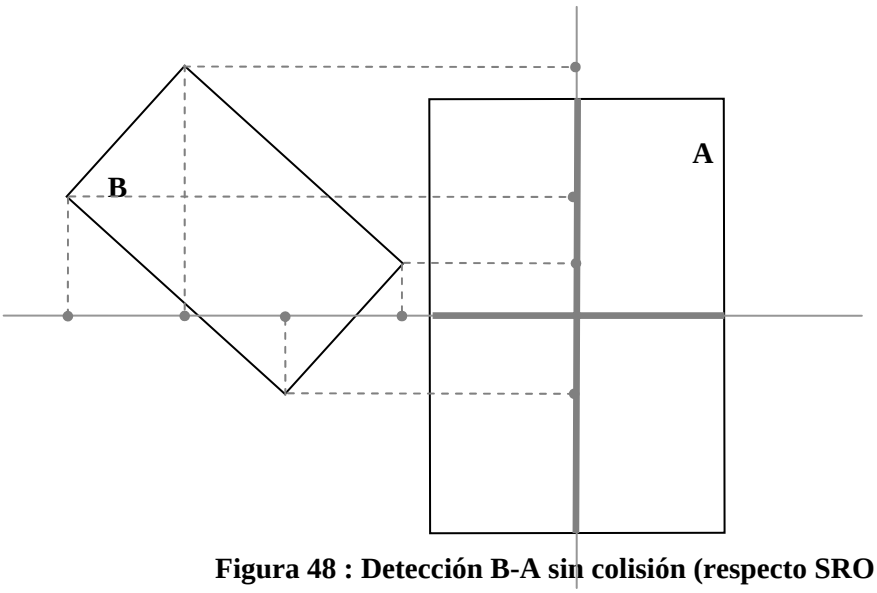


Figura 48 : Detección B-A sin colisión (respecto SRO de A)

Ahora supongamos los mismos objetos colisionables, A y B, pero en este caso hay un vértice de B dentro del área colisionable de A, por lo que la comprobación deberá ser positiva.

Primero hacemos la comprobación de A respecto B, vemos que ningún vértice de A está dentro de los rangos X y Z que define el área de B:

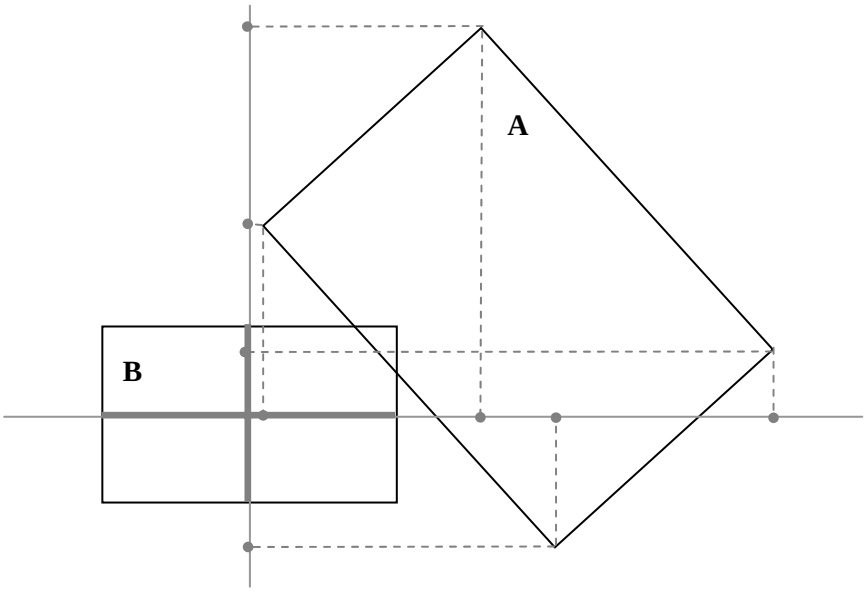


Figura 49 : Detección A-B con colisión (respecto SRO de B)

Después hacemos la comprobación de B respecto A, vemos que el vértice de B señalado queda dentro de los rangos que establece el área de A, por lo que en este caso tenemos detectada la colisión:

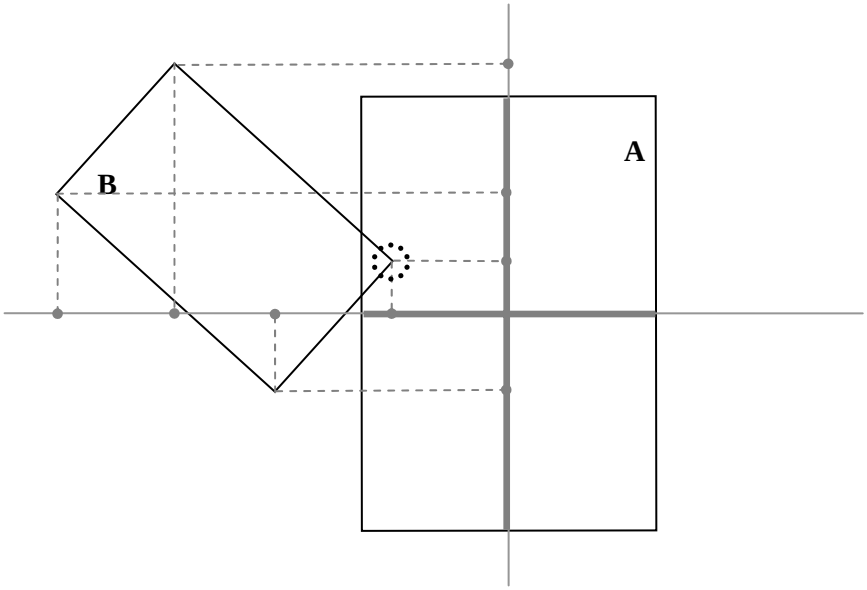


Figura 50 : Detección B-A con colisión (respecto SRO de A)

19.4. Recolocación de los objetos

Este paso sólo se realiza si hemos detectado una colisión entre dos objetos A y B.

Tras el paso anterior tenemos localizado el vértice de un objeto que se superpone sobre el área de colisión del otro objeto. Es posible que haya más vértices con superposición, pero tomaremos este primer vértice como el único a tratar de cara al proceso.

Tras encontrar la colisión es necesario recolocar los objetos, de forma que ambos no simulen un efecto *pegado* en las posteriores iteraciones. Esto es debido a que tras la colisión y el ajuste de los parámetros físicos, la siguiente iteración puede no resultar suficiente para *despegar* ambos objetos, por lo que volverá a haber colisión y volverán a invertir sus parámetros físicos.

Este hecho puede provocar que ambos objetos continúen colisionando hasta que de algún modo se separen por algún extremo de los rectángulos. La siguiente imagen ilustra varias iteraciones donde ocurre este efecto *pegado* al no recolocar los objetos:

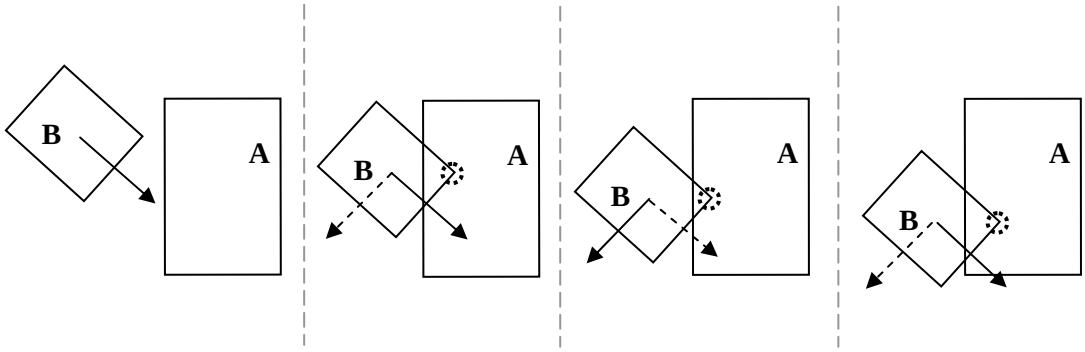


Figura 51 : Colisiones sin recolocación

Por el contrario, si se recoloca el objeto B correctamente tras la primera colisión, este problema no se producirá:

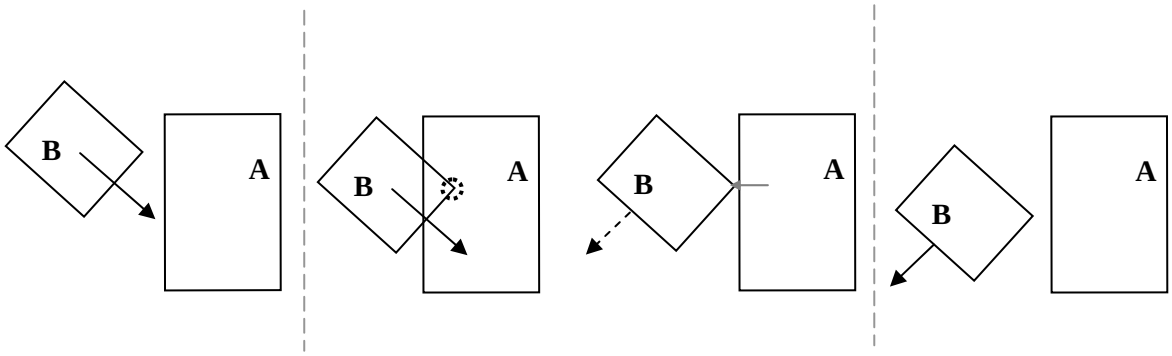


Figura 52 : Colisiones con recolocación

Por lo tanto es necesaria la recolocación de los objetos, separándolos cierta distancia ε para evitar futuras colisiones ficticias.

Para la recolocación partiremos del vértice dónde encontramos la colisión (X_p, Z_p) y hallaremos la menor distancia en X o Z para que ese vértice quede fuera de uno de los intervalos [X_{max}, X_{min}] ó [Z_{max}, Z_{min}].

Por lo tanto tomamos el mínimo entre los cuatro valores siguientes:

- $\mathbf{X_{max} - X_p}$
- $\mathbf{X_p - X_{min}}$
- $\mathbf{Z_{max} - Z_p}$
- $\mathbf{Z_p - Z_{min}}$

Una vez hallado, determinamos qué objeto vamos a recolocar, en función de su tipo (móvil o fijo):

- Si el objeto de referencia (el del SRO en el que estamos) es móvil, entonces recolocaremos este objeto, ya que es menos costoso calcular su traslación.
- En otro caso recolocaremos el objeto al que pertenece el vértice de colisión encontrado.

Ahora hay que hallar la traslación que le vamos a hacer al objeto seleccionado, para ello:

- Si hay que mover el objeto de referencia:
  - Si el mínimo es $X_{max} - X_p$ lo movemos en X $\mathbf{-(X_{max} - X_p + \varepsilon)}$
  - Si el mínimo es $X_p - X_{min}$ lo movemos en X $\mathbf{+(X_p - X_{min} + \varepsilon)}$
  - Si el mínimo es $Z_{max} - Z_p$ lo movemos en Z $\mathbf{-(Z_{max} - Z_p + \varepsilon)}$
  - Si el mínimo es $Z_p - Z_{min}$ lo movemos en Z $\mathbf{+(Z_p - Z_{min} + \varepsilon)}$
- Si hay que mover el objeto con el vértice de colisión:
  - Si el mínimo es $X_{max} - X_p$ lo movemos en X $\mathbf{+(X_{max} - X_p + \varepsilon)}$
  - Si el mínimo es $X_p - X_{min}$ lo movemos en X $\mathbf{-(X_p - X_{min} + \varepsilon)}$
  - Si el mínimo es $Z_{max} - Z_p$ lo movemos en Z $\mathbf{+(Z_{max} - Z_p + \varepsilon)}$
  - Si el mínimo es $Z_p - Z_{min}$ lo movemos en Z $\mathbf{-(Z_p - Z_{min} + \varepsilon)}$

Dónde ε es un pequeño valor para separar convenientemente ambos objetos.

Siguiendo con el ejemplo del apartado anterior, tenemos dos objetos colisionados: A objeto fijo y B objeto movible. Al comprobar el objeto B respecto el objeto A encontramos la colisión en el vértice (X_p, Z_p) del objeto B del modo que se muestra en la imagen:

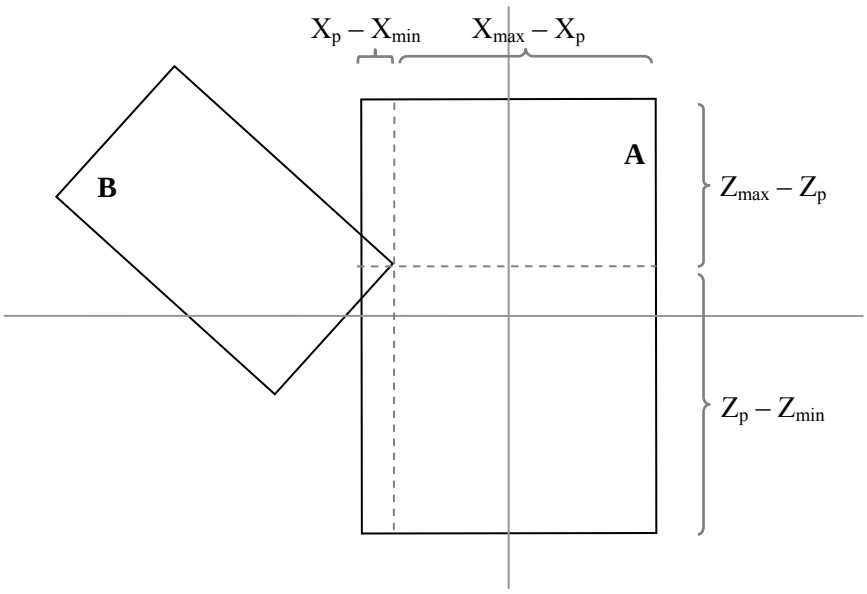


Figura 53 : Ejemplo de recolocación

Si resulta que A es un objeto fijo, tenemos que mover el objeto B [$\mathbf{- (X_p - X_{min} + \varepsilon)}$] en el eje X, de forma que quede desplazado ligeramente hacia la izquierda según la imagen. Este movimiento se puede aplicar fácilmente a la matriz de transformación del objeto B, multiplicando la traslación por la matriz inversa de transformación del objeto A.

En el caso contrario, si el objeto A hubiese sido móvil, habría que moverlo [$\mathbf{+ (X_p - X_{min} + \varepsilon)}$] en el eje X, algo que se puede aplicar directamente sobre la matriz de transformación de A.

En caso que tras la recolocación vuelva a haber colisión en otro punto, se tratará en la siguiente iteración del proceso global.

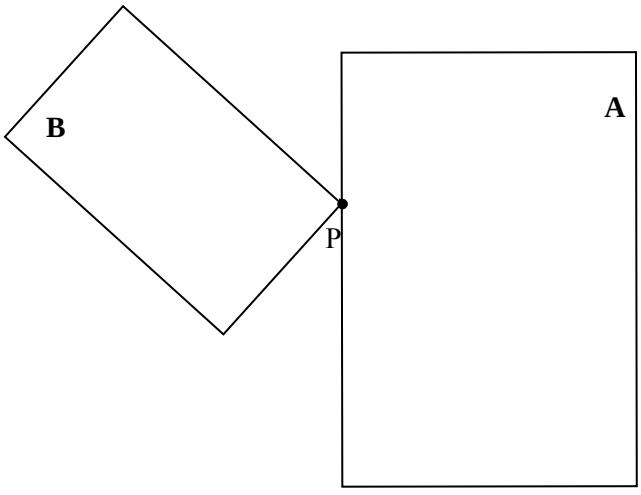


Figura 54 : Esquema tras la recolocación

19.5. Intercambio de parámetros físicos

Este es último paso del motor de colisiones, que se realiza tras haber localizado una colisión y tras haber recolocado los objetos. Supone además gran complejidad, tanto de cara al diseño como de cara a la implementación y ejecución del algoritmo. Sin embargo, hay que considerar que aquí no es tan necesaria la eficiencia, ya que en muy pocas iteraciones se van a producir colisiones de objetos.

19.5.1. Parámetros que intervienen

Tomamos un caso concreto de colisión entre los objetos A y B, dónde el objeto B mete la esquina dentro del objeto A. En este caso, sobre el SRU tendremos la siguiente situación previa a la colisión:

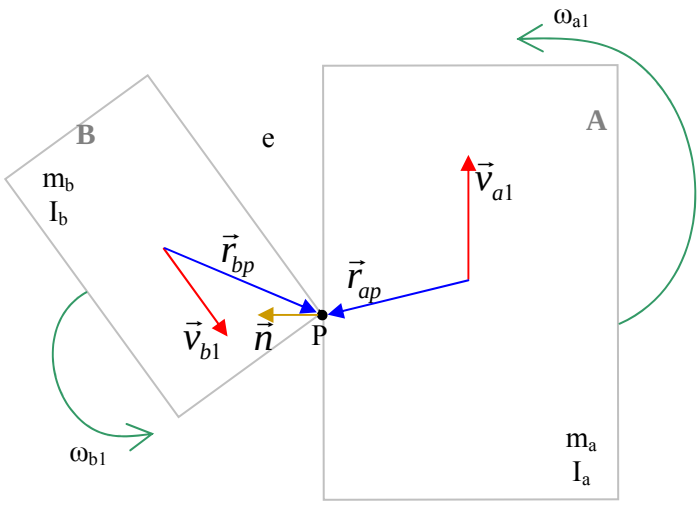


Figura 55 : Parámetros físicos considerados

Vamos a identificar los diferentes parámetros representados en el diagrama:

P	Punto exacto de colisión, que ya se ha calculado en el paso previo de recolocación
$\vec{v}_{a1}$ $\vec{v}_{b1}$	Velocidades lineales de A y B previas a la colisión. En el diagrama se han representado unas velocidades de ejemplo, pero el proceso es equivalente para cualquier otra dirección y sentido.
ω_{a1} ω_{b1}	Velocidades angulares de A y B previas a la colisión. Deberían ser vectores, pero al reducirse el estudio sobre un plano, sólo se toma el giro sobre el eje perpendicular.
m_a m_a	Masas de los objetos A y B, que cuantifican lo difícil que es moverlos linealmente.
I_a I_b	Momento de inercia de los objetos A y B, que cuantifican lo difícil que es moverlos angularmente. Para el caso de los rectángulos se calcula cómo: $I = \frac{m(ancho^2 + alto^2)}{12}$ (es una fórmula física común)
$\vec{r}_{ap}$ $\vec{r}_{bp}$	Radios desde el centro de cada objeto hasta el punto de colisión
$\vec{n}$	Vector ortogonal al lado de colisión, cuyo sentido es hacia fuera del objeto al que pertenece dicho lado (en este caso de A)
e	Factor de elasticidad de la colisión. Es un número del intervalo [0,1] dónde 0 equivale a una colisión inelástica y 1 equivale a una colisión elástica

Teniendo en cuenta los parámetros anteriores, el **momento lineal** de cada objeto se calcula cómo:

$$m\ \vec{v}$$

El **momento angular** de cada objeto se calcula cómo:

$$I\ w$$

Ambos momentos van a variar tras la colisión, provocando un cambio en las velocidades lineales y angulares de cada uno. Por lo que tras el proceso tenemos que obtener los siguientes parámetros:

$\vec{v}_{a2}$ $\vec{v}_{b2}$ Velocidades lineales de A y B posteriores a la colisión.

ω_{a2} ω_{b2} Velocidades angulares de A y B posteriores a la colisión.

19.5.2. Modelo del impulso

Una colisión real no es algo instantáneo, sino que durante un corto periodo de tiempo entran en juego multitud de fuerzas. Para modelar estos fenómenos correctamente necesitaríamos detalles sobre los materiales, su geometría exacta, deformaciones, propagación de la presión sobre el objeto, etc.

Sin embargo, este hecho puede modelarse de un modo más sencillo con el concepto de **impulso**, que se define como el cambio del momento lineal y angular de un objeto cuando se le aplica una fuerza durante un pequeño periodo de tiempo.

Siguiendo con este concepto, podemos hacer dos simplificaciones al modelo:

- Suponemos que la colisión ocurre de forma tan rápida que el estado de los objetos no varía durante la misma.
- Suponemos que no existe ninguna otra fuerza más que la propia de la colisión.

Por lo tanto sólo entra en juego una fuerza, cuya dirección es perpendicular al lado de colisión (esto es el vector $\vec{n}$), por lo que el impulso será algo cómo:

$$impulso = j\vec{n}$$

dónde ***j*** es el **factor del impulso**, que es necesario calcular.

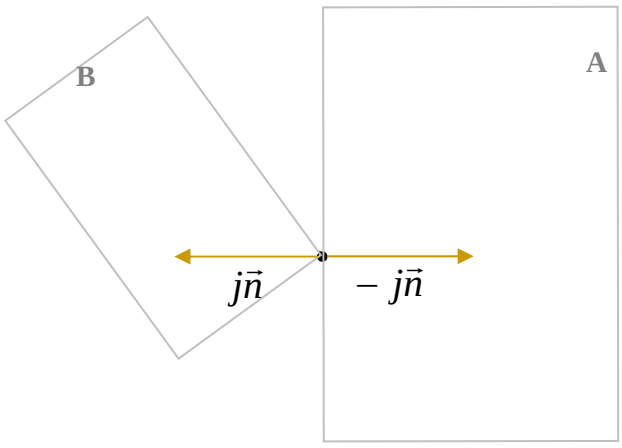


Figura 56 : Impulso para A y B

Como puede verse en la anterior figura, el cambio del momento para el objeto B (el objeto que mete la esquina) equivale a $j\vec{n}$ mientras que el cambio del momento para el objeto A (el objeto al que meten la esquina) es justamente el contrario $-j\vec{n}$

Por lo tanto, el momento lineal de cada objeto tras la colisión, será el momento lineal anterior a la colisión modificado por su impulso correspondiente:

$$m_a\vec{v}_{a2} = m_a\vec{v}_{a1} - j\vec{n} \quad (1)$$

$$m_b\vec{v}_{b2} = m_b\vec{v}_{b1} + j\vec{n} \quad (2)$$

El cambio del momento angular depende de la distancia del centro del objeto al punto *P* de colisión, de forma que cuanto más alejado esté *P* del centro del objeto más cambio angular habrá (y viceversa). Además, como estamos modelando colisiones 2D, el momento angular es numérico y por tanto el impulso angular debe ser igualmente numérico, no vectorial.

Para contemplar ambos hechos, basta con tomar como impulso angular $(\vec{r} \times j\vec{n}) \cdot \vec{k}$, dónde $\vec{k}$ es el vector unitario del eje perpendicular al plano de colisión. Podemos ver cómo el producto vectorial refleja la distancia del centro del objeto hasta el punto de colisión y el producto escalar extrae la componente adecuada del vector resultante.

Considerando lo anterior, tendremos:

$$I_a w_{a2} = I_a w_{a1} - (\vec{r}_{ap} \times j\vec{n}) \cdot \vec{k} \quad (3)$$

$$I_b w_{b2} = I_b w_{b1} + (\vec{r}_{bp} \times j\vec{n}) \cdot \vec{k} \quad (4)$$

Despejando el parámetro correspondiente en cada una de las cuatro fórmulas anteriores (1-4), podemos obtener las velocidades lineales y angulares finales de los objetos tras la colisión:

$$\vec{v}_{a2} = \vec{v}_{a1} - \frac{j\vec{n}}{m_a} \quad (5)$$

$$\vec{v}_{b2} = \vec{v}_{b1} + \frac{j\vec{n}}{m_b} \quad (6)$$

$$\omega_{a2} = \omega_{a1} - \frac{(\vec{r}_{ap} \times j\vec{n}) \cdot \vec{k}}{I_a} \quad (7)$$

$$\omega_{b2} = \omega_{b1} + \frac{(\vec{r}_{bp} \times j\vec{n}) \cdot \vec{k}}{I_b} \quad (8)$$

Por lo tanto, ya tenemos los parámetros finales identificados, y el estudio se reduce al cálculo del **factor de impulso** *j*, que se verá en el siguiente apartado.

19.5.3. Cálculo del factor del impulso

Como hemos visto en el anterior apartado, tan sólo nos queda calcular el factor del impulso, es decir cuantificar en qué medida van a intercambiar el momento los objetos A y B.

Vamos a comenzar estudiando la velocidad lineal que sigue el punto P de colisión en el objeto A y B antes de la colisión, a lo que llamaremos $\vec{v}_{ap1}$ .y $\vec{v}_{bp1}$.

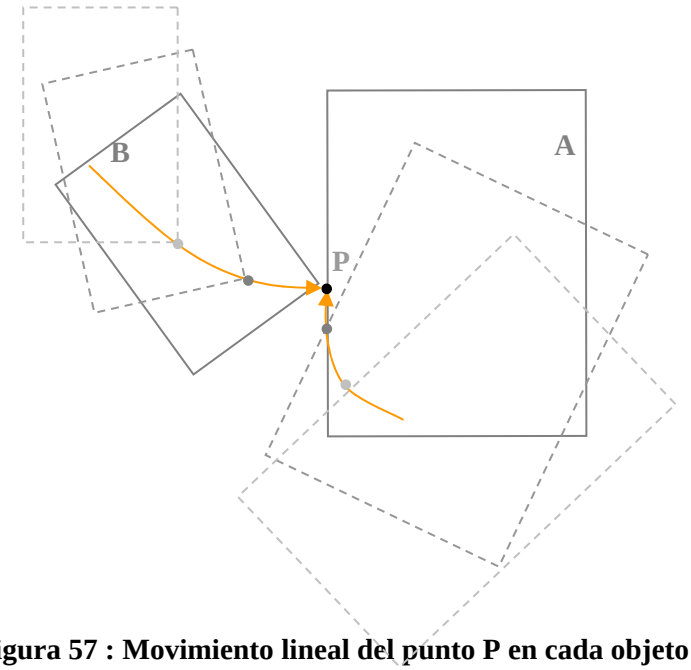


Figura 57 : Movimiento lineal del punto P en cada objeto A y B

Estas velocidades lineales del punto P en cada objeto justo antes de la colisión vienen dadas por las siguientes fórmulas:

$$\vec{v}_{ap1} = \vec{v}_{a1} + w_{a1}\vec{k} \times \vec{r}_{ap}$$

$$\vec{v}_{bp1} = \vec{v}_{b1} + w_{b1}\vec{k} \times \vec{r}_{bp}$$

dónde $\vec{k}$ es el vector unitario del eje perpendicular al plano de colisión

Por lo tanto podemos calcular la velocidad a la que ambos objetos se acercan respecto ese punto P justo antes de la colisión, que llamaremos $\vec{v}_{ab1}$ y se calculará cómo:

$$\vec{v}_{ab1} = \vec{v}_{bp1} - \vec{v}_{ap1}$$

$$\vec{v}_{ab1} = \left(\vec{v}_{b1} + w_{b1}\vec{k} \times \vec{r}_{bp}\right) - \left(\vec{v}_{a1} + w_{a1}\vec{k} \times \vec{r}_{ap}\right) \quad (9)$$

Siguiendo el mismo cálculo, podemos calcular la velocidad a la que ambos objetos se alejan respecto ese punto P justo después de la colisión, que llamaremos $\vec{v}_{ab2}$ y se calculará cómo:

$$\vec{v}_{ab2} = \left(\vec{v}_{b2} + w_{b2}\vec{k} \times \vec{r}_{bp}\right) - \left(\vec{v}_{a2} + w_{a2}\vec{k} \times \vec{r}_{ap}\right) \quad (10)$$

Como tenemos el vector $\vec{n}$ perpendicular y hacia afuera del lado de choque, podemos calcular la velocidad relativa en la dirección de $\vec{n}$ antes y después de la colisión del siguiente modo:

$$\vec{v}_{relativa\_anterior} = \vec{v}_{ab1} \cdot \vec{n}$$

$$\vec{v}_{relativa\_posterior} = -\vec{v}_{ab2} \cdot \vec{n}$$

Notar que después de la colisión la velocidad es inversa, y de ahí el signo negativo que se antepone.

Por el principio de conservación de la energía tendríamos:

$$\vec{v}_{ab2} \cdot \vec{n} = -\vec{v}_{ab1} \cdot \vec{n}$$

pero como estamos incluyendo un factor de elasticidad *e*, realmente tendremos:

$$\vec{v}_{ab2} \cdot \vec{n} = -e\vec{v}_{ab1} \cdot \vec{n} \qquad (11)$$

Partiendo de esta fórmula (11) y desglosando $\vec{v}_{ab2}$ con la anterior (10), tendremos lo siguiente:

$$\left(\left(\vec{v}_{b2} + w_{b2}\vec{k} \times \vec{r}_{bp}\right) - \left(\vec{v}_{a2} + w_{a2}\vec{k} \times \vec{r}_{ap}\right)\right) \cdot \vec{n} = -e\vec{v}_{ab1} \cdot \vec{n}$$

Sustituyendo ahora con las fórmulas (5-8) tendremos:

$$\left(\left(\vec{v}_{b1} + \frac{j\vec{n}}{m_b} + \omega_{b1} + \frac{\left(\vec{r}_{bp} \times j\vec{n}\right) \cdot \vec{k}}{I_b}\vec{k} \times \vec{r}_{bp}\right) - \left(\vec{v}_{a1} - \frac{j\vec{n}}{m_a} + \omega_{a1} - \frac{\left(\vec{r}_{ap} \times j\vec{n}\right) \cdot \vec{k}}{I_a}\vec{k} \times \vec{r}_{ap}\right)\right) \cdot \vec{n} = -e\vec{v}_{ab1} \cdot \vec{n}$$

La parte izquierda contiene $\vec{v}_{ab1} \cdot \vec{n}$ según aparece expresado en la fórmula (9), por lo que puedo sacara factor común en ambos lados, y obtengo así:

$$\left(\left(\frac{j\vec{n}}{m_b} + \frac{\left(\vec{r}_{bp} \times j\vec{n}\right) \times \vec{r}_{bp}}{I_b}\right) - \left(-\frac{j\vec{n}}{m_a} - \frac{\left(\vec{r}_{ap} \times j\vec{n}\right) \times \vec{r}_{ap}}{I_a}\right)\right) \cdot \vec{n} = -(1+e)\vec{v}_{ab1} \cdot \vec{n}$$

Y quitando los paréntesis, obtenemos:

$$\frac{j\vec{n} \cdot \vec{n}}{m_b} + \frac{\left(\vec{r}_{bp} \times j\vec{n}\right) \times \vec{r}_{bp} \cdot \vec{n}}{I_b} + \frac{j\vec{n} \cdot \vec{n}}{m_a} + \frac{\left(\vec{r}_{ap} \times j\vec{n}\right) \times \vec{r}_{ap} \cdot \vec{n}}{I_a} = -(1+e)\vec{v}_{ab1} \cdot \vec{n}$$

Como $\vec{n}$ es un vector normalizado ya podemos despejar el factor del impulso:

$$j = \frac{-(1+e)\vec{v}_{ab1} \cdot \vec{n}}{\frac{1}{m_b} + \frac{\left(\vec{r}_{bp} \times \vec{n}\right) \times \vec{r}_{bp} \cdot \vec{n}}{I_b} + \frac{1}{m_a} + \frac{\left(\vec{r}_{ap} \times \vec{n}\right) \times \vec{r}_{ap} \cdot \vec{n}}{I_a}}$$

Para optimizar el cálculo, podemos reducir el número de productos vectoriales usando la propiedad del triple producto escalar, por la que $(\vec{A}\times\vec{B})\cdot\vec{C}=(\vec{B}\times\vec{C})\cdot\vec{A}$. A partir de esta propiedad podemos determinar que $(\vec{A}\times\vec{B})\times\vec{A}\cdot\vec{B}=(\vec{A}\times\vec{B})\cdot(\vec{A}\times\vec{B})$ y por tanto tendremos la expresión final de *j*:

$$j=\frac{-(1+e)\vec{v}_{ab1}\cdot\vec{n}}{\frac{1}{m_b}+\frac{(\vec{r}_{bp}\times\vec{n})\cdot(\vec{r}_{bp}\times\vec{n})}{I_b}+\frac{1}{m_a}+\frac{(\vec{r}_{ap}\times\vec{n})\cdot(\vec{r}_{ap}\times\vec{n})}{I_a}} \tag{12}$$

Por lo que ahora sólo tendremos que calcular dos productos vectoriales: $(\vec{r}_{bp}\times\vec{n})$ y $(\vec{r}_{ap}\times\vec{n})$

Esta fórmula es válida para una colisión genérica entre objetos móviles. Para obtener la fórmula de la colisión entre un móvil (por ejemplo A) y un objeto fijo (por ejemplo B), basta con suponer que:

$$m_b\to\infty$$

$$I_b\to\infty$$

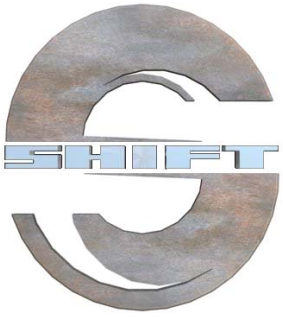
Y sustituyendo en la fórmula (12) tendremos:

$$j=\frac{-(1+e)\vec{v}_{ab1}\cdot\vec{n}}{\frac{1}{m_a}+\frac{(\vec{r}_{ap}\times\vec{n})\times(\vec{r}_{ap}\times\vec{n})}{I_a}} \tag{13}$$

Capítulo 6

Diseño del motor de sonido

16 de marzo de 2008



Capítulo 6: Diseño del motor de sonido

Tabla de apartados

Lista de figuras

20. Introducción (*motor de sonido*)

El motor de sonidos permite manipular los buffer de sonido, situándolos en el entorno 3D bajo la misma ubicación que los objetos a los que están asociados.

Dentro del esquema general, el sistema de sonidos queda limitado por el ámbito rayado:

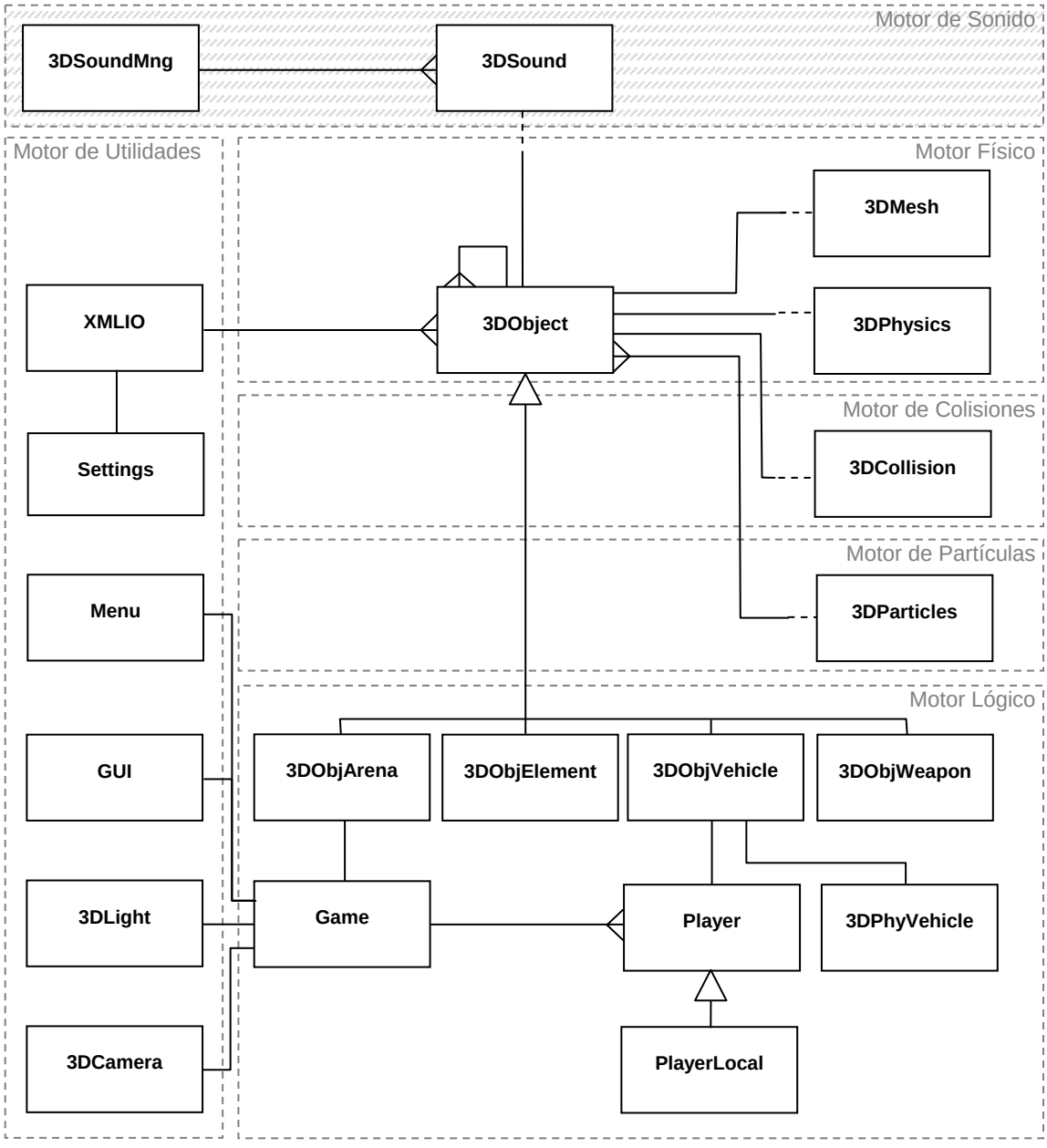


Figura 58 : Límites del motor de sonidos

Cómo refleja en el diagrama, el módulo de sonidos consta de las siguientes clases:

- **3DSoundMng**
  Gestor de todos los elementos de sonido de la aplicación.

- **3DSound**
  Clase para abstraer el buffer de cada uno de los sonidos de la aplicación.

A continuación veremos en detalle cada uno de estos aspectos controlados por el motor de sonidos.

21. Parámetros de la clase *3DSoundMng*

Sólo habrá una instancia de esta clase en ejecución, que nos servirá para comunicarnos con el dispositivo de audio a través de DirectSound y para gestionar todos los sonidos en escena.

El gestor nos ofrece las siguientes primitivas globales:

- Cargar sonidos externos .WAV al buffer correspondiente de memoria
- Mantener el formato global de audio, que en principio se establece en 22kHz y 16 bits
- Mantener el entorno global 3D de sonidos, junto con los parámetros que lo configuran, tales como el rango y factor de decaimiento.
- Gestionar el volumen global de todos los sonidos en escena

No se ha activado el efecto doppler, ya que por las pruebas realizadas resulta casi imperceptible durante el juego y no se desea complicar el proceso de sonido de la escena. En todo caso, se deja implementado el efecto doppler mediante una directiva de preprocesador por si se desea utilizar esta técnica en algún momento (mediante la directiva `SOUND3D_DOPPLER_FACTOR`, ver más detalles en el apartado *45. Documentación de clases*).

22. Parámetros de la clase *3DSound*

Existirá una instancia de esta clase por cada sonido cargado en escena, pudiendo asociarse con un objeto en concreto o de forma global para la escena.

- Si se asocia a un objeto, el sonido se situará en la misma posición 3D que el centro de dicho objeto, que viene dado por la matriz de transformación del mismo.
- Si se asocia a la escena de forma global, el sonido no tiene ubicación, por lo que su audio se reproduce sin implementar el entorno 3D.

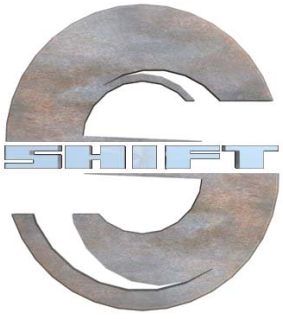
En todo caso la clase ofrece las siguientes operaciones sobre el sonido concreto:

- Gestionar el volumen
- Gestionar la frecuencia
- Reproducción única o en bucle
- Detención de la reproducción

Capítulo 7

Diseño del motor de utilidades

16 de marzo de 2008



Capítulo 7: Diseño del motor de utilidades

Tabla de apartados

Lista de figuras

23. Introducción (*motor de utilidades*)

El motor de utilidades permite mantener la cámara, la luz, la configuración de la aplicación, interfaz de usuario y un sistema de carga externa de modelos mediante XML.

Dentro del esquema general, el sistema de utilidades queda limitado por el ámbito rayado:

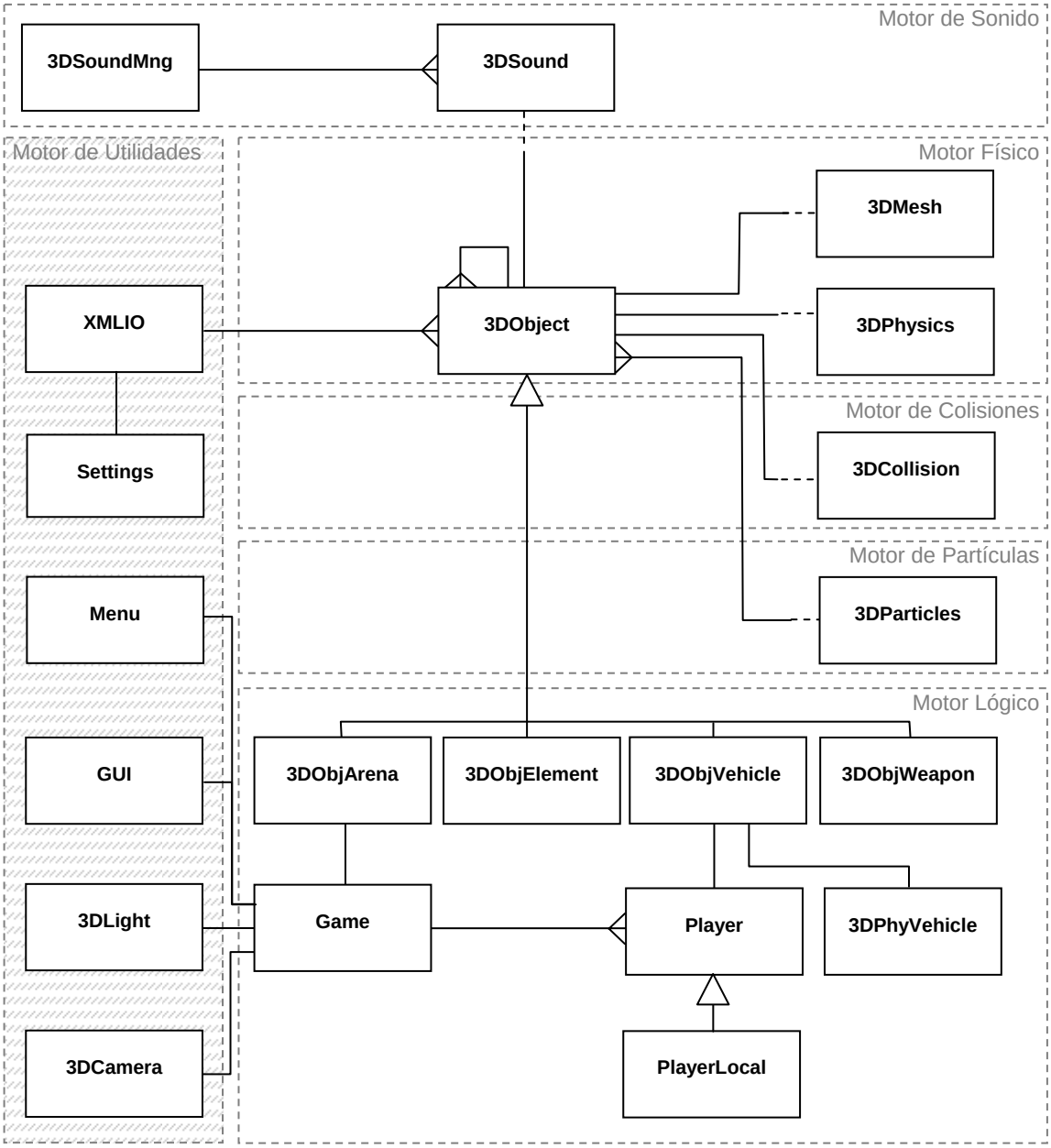


Figura 59 : Límites del motor de utilidades

Cómo refleja en el diagrama, el módulo de utilidades consta de las siguientes clases:

- **XMLIO**
  Permite cargar o almacenar de forma persistente datos y estructuras de la aplicación con formato XML. Se usará para los modelos, la configuración, el ranking, etc.

- **Settings**
  Esta clase implementa la gestión de parámetros personalizables por parte de los usuarios, tales como la configuración de video, sonidos, estilos de juego, etc.

- **Menu**
  Esta clase tan sólo hace llamada a las instancias apropiadas en cada momento, en función del entorno en el que se encuentre el interfaz de usuario.

- **GUI**
  Esta clase implementa el interfaz de usuario, incluyendo toda la lógica interna de navegación de menús y opciones de usuario.

- **3DLight**
  Clase para definir parámetros de luz ambiente, especular y difusa dentro de la aplicación. Para facilitar la programación no se implementan focos direccionales de luz, por lo que sólo se configura la luz global de la escena.

- **3DCamera**
  Clase que abstrae el concepto de cámara para el dispositivo gráfico, con todos sus parámetros de foco, posición y rotación. Dentro de esta clase se implementan los algoritmos de seguimiento de objetos, cambios de vista, etc.

En los siguientes apartados se darán a conocer más detalles sobre cada una de las clases de utilidades aquí listadas.

24. Parámetros de la clase *XMLIO*

Se implementa una clase que permite manipular los datos XML, tanto para la lectura como para escritura. Esta clase facilitará la navegación por la estructura jerárquica de cada XML, permitiendo tomar los datos de cada fichero externo, modificarlos, guardarlos, crearlos, etc.

Habrá una instancia global de esta clase que será utilizada por cada constructor dónde sea necesaria la lectura o almacenamiento de datos externos. En los siguientes apartados veremos en detalle la definición de elementos y atributos de cada XML utilizado en la aplicación.

24.1. XML para la configuración

Se utiliza un único fichero XML para almacenar los datos referentes a la configuración de usuario de la aplicación. Estos datos se utilizan en la clase *Settings*, que será detallada más adelante.

Siguiendo el DTD que define la estructura del XML, el elemento raíz es *settings*, que contiene los tres aspectos personalizables en la aplicación *display*, *sound* y *game*.

```
<!ELEMENT settings (display, sound, game)>
```

El elemento *display* permite mantener los parámetros elegidos para el sistema 3D de DirectX:

- *quality* para configurar la calidad general de video (0 personalizada, 1 alta, 2 media, 3 baja)
- *winfull* por si queremos (1) o no (0) pantalla completa
- *stadistics* por si queremos (1) o no (0) estadísticas de DirectX en pantalla
- *resolution* para la resolución de pantalla
- *filter* para el filtro usado en las texturas (0 puntual, 1 lineal, 2 anisotrópico)
- *midmapping* por si queremos (1) o no (0) activar el midmapping

```
<!ELEMENT display (quality winfull stadistics resolution filter mipmapping)>
<!ELEMENT quality EMPTY>
<!ATTLIST quality type (0 1 2 3) #REQUIRED>
<!ELEMENT winfull EMPTY>
<!ATTLIST winfull enabled (0 1) #REQUIRED>
<!ELEMENT stadistics EMPTY>
<!ATTLIST stadistics enabled (0 1) #REQUIRED>
<!ELEMENT resolution EMPTY>
<!ATTLIST resolution width CDATA #REQUIRED>
<!ELEMENT resolution EMPTY>
<!ATTLIST resolution height CDATA #REQUIRED>
<!ELEMENT filter EMPTY>
<!ATTLIST filter type (0 1 2) #REQUIRED
<!ELEMENT mipmapping EMPTY>
<!ATTLIST mipmapping enabled (0 1) #REQUIRED>
```

El elemento *sound* permite mantener los parámetros de volumen para los efectos de sonido.

```
<!ELEMENT sound (effects)>
<!ELEMENT effects EMPTY>
<!ATTLIST effects volume CDATA #REQUIRED>
```

El elemento *game* permite mantener el nombre por defecto para el jugador, que se usará a la hora de registrar los mejores tiempos en el ranking de cada arena.

```
<!ELEMENT game (player)>
<!ELEMENT player EMPTY>
<!ATTLIST player name CDATA #REQUIRED>
```

24.2. XML para ranking

Se usan ficheros XML para gestionar el ranking de los mejores tiempos en cada una de las arenas. Tendremos entonces un fichero XML de ranking por cada arena disponible. Cada fichero parte de un nodo *ranking* que contiene cada uno de los elementos *rank* de la lista.

```
<!ELEMENT ranking (object*)>
```

Por cada elemento *rank* almacenamos el nombre del jugador, el nombre del vehículo que seleccionó y el tiempo que hizo en la arena a la que hace referencia el fichero.

```
<!ELEMENT rank EMPTY>
<!ATTLIST rank player CDATA #REQUIRED>
<!ATTLIST rank vehicle CDATA #REQUIRED>
<!ATTLIST rank time CDATA #REQUIRED>
```

24.3. XML para modelos 3D

La lectura de ficheros XML también se realiza en el constructor de la clase *3DObject*, para cargar la definición de los objetos 3D y sus propiedades físicas, geométricas, de sonido, etc.

Por lo tanto los objetos internos se definen mediante ficheros XML externos que se cargan dinámicamente bajo demanda en la aplicación. Este interfaz ofrece gran extensibilidad al videojuego, ya que se pueden incorporar nuevos vehículos, escenarios u objetos incluyendo únicamente ficheros de definición XML.

Además, mediante este interfaz XML se ofrece la posibilidad de implementar un editor de pistas que permita diseñar nuevos escenarios mediante un entorno gráfico y los objetos básicos del videojuego. Este editor tan sólo tendría que representar el escenario en cada momento e ir manteniendo correctamente el fichero XML mediante la clase aquí implementada (ver detalles en el apartado *51. Mejoras propuestas*).

Para arenas y vehículos se define una estructura XML especializada, teniendo ambas en común la parte de los objetos, por lo que a continuación se va a describir la definición común de todos los objetos y posteriormente la especialización para arenas y vehículos

24.3.1. Objetos (incluidos vehículos y arenas)

Para todos los objetos podemos definir los siguientes elementos:

- Una geometría (opcional)
- Una posición (opcional)
- Una rotación (opcional)
- Una escala (opcional)
- Una física (opcional)
- Una colisión (opcional)
- Un sonido (opcional)
- Un sistema de partículas primario (opcional)
- Un sistema de partículas secundario (opcional)
- De cero a n objetos hijos

```
<!ELEMENT object (mesh?, position?, rotation?, scale?, physic?,
                  collision?, sound?, particles1?, particles2?, object*)>
```

Para la **geometría** es necesario indicar un fichero .X externo (formato DirectX para definir vértices, normales y texturas) y opcionalmente un atributo alpha, que indica si sobre el objeto se aplican trasparencias siempre (2) sólo en cámaras persecutorias bajas (1) o nunca (0, valor por defecto)

```
<!ELEMENT mesh EMPTY>
<!ATTLIST mesh scr CDATA #REQUIRED>
<!ATTLIST mesh alpha CDATA #IMPLIED>
```

Para la **posición** se pueden fijar los valores de X, Y, Z en el espacio:

```
<!ELEMENT position EMPTY>
<!ATTLIST position x CDATA #IMPLIED>
<!ATTLIST position y CDATA #IMPLIED>
<!ATTLIST position z CDATA #IMPLIED>
```

Para la **rotación** se pueden fijar los valores sobre los ejes X, Y, Z:

```
<!ELEMENT rotation EMPTY>
<!ATTLIST rotation x CDATA #IMPLIED>
<!ATTLIST rotation y CDATA #IMPLIED>
<!ATTLIST rotation z CDATA #IMPLIED>
```

Para el **escalado** se pueden fijar los valores sobre los ejes X, Y, Z:

```
<!ELEMENT scale EMPTY>
<!ATTLIST scale x CDATA #IMPLIED>
<!ATTLIST scale y CDATA #IMPLIED>
<!ATTLIST scale z CDATA #IMPLIED>
```

Para la **física** podemos fijar la velocidad, la aceleración y la fricción, tanto lineal (en unidades de la escena) como angular (en radianes):

```
<!ELEMENT physic EMPTY>
<!ATTLIST physic l_v CDATA #IMPLIED>
<!ATTLIST physic l_a CDATA #IMPLIED>
<!ATTLIST physic l_f CDATA #IMPLIED>
<!ATTLIST physic l_dx CDATA #IMPLIED>
<!ATTLIST physic l_dy CDATA #IMPLIED>
<!ATTLIST physic l_dz CDATA #IMPLIED>
<!ATTLIST physic a_vx CDATA #IMPLIED>
<!ATTLIST physic a_vy CDATA #IMPLIED>
<!ATTLIST physic a_vz CDATA #IMPLIED>
<!ATTLIST physic a_ax CDATA #IMPLIED>
<!ATTLIST physic a_ay CDATA #IMPLIED>
<!ATTLIST physic a_az CDATA #IMPLIED>
<!ATTLIST physic a_fx CDATA #IMPLIED>
<!ATTLIST physic a_fy CDATA #IMPLIED>
<!ATTLIST physic a_fz CDATA #IMPLIED>
```

Para las **colisiones** podemos indicar si el objeto es colisionable (se indica el nodo) o no (no se indica el nodo). Si incluimos el nodo, mediante el atributo *mass* podemos indicar si el objeto es móvil (se indica el atributo con una masa concreta) o fijo (no se indica el atributo):

```
<!ELEMENT collision EMPTY>
<!ATTLIST collision mass CDATA #IMPLIED>
```

Para el **sonido** del objeto tan sólo es necesario indicar el fichero externo .WAV asociado:

```
<!ELEMENT sound EMPTY>
<!ATTLIST sound src CDATA #REQUIRED>
```

Para los **sistemas de partículas** (tanto el primario como el secundario), podemos indicar varios atributos que lo definan, como son:

- Fichero .BMP o .JPG que define la textura de cada partícula
- Máximo número de partículas
- Número de partículas liberadas en cada iteración
- Tiempo para cada iteración
- Vida en segundos de cada partícula liberada
- Tamaño en unidades de la escena de cada partícula
- Aleatoriedad de la dirección con la que son liberadas las partículas
- Color y transparencia de las partículas
- Posición relativa al objeto dónde se liberan las partículas (valores para X, Y, Z)
- Velocidad absoluta de las partículas (valores para X, Y, Z)
- Fuerza gravitacional de las partículas (valores para X, Y, Z)
- Fuerza del viento sobre las partículas (valores para X, Y, Z)

```
<!ELEMENT particles1 EMPTY>
<!ATTLIST particles1 tex CDATA #REQUIRED>
<!ATTLIST particles1 max CDATA #IMPLIED>
<!ATTLIST particles1 release_n CDATA #IMPLIED>
<!ATTLIST particles1 release_t CDATA #IMPLIED>
<!ATTLIST particles1 life CDATA #IMPLIED>
<!ATTLIST particles1 size CDATA #IMPLIED>
<!ATTLIST particles1 vel_var CDATA #IMPLIED>
<!ATTLIST particles1 color_x CDATA #IMPLIED>
<!ATTLIST particles1 color_y CDATA #IMPLIED>
<!ATTLIST particles1 color_z CDATA #IMPLIED>
<!ATTLIST particles1 color_u CDATA #IMPLIED>
<!ATTLIST particles1 pos_x CDATA #IMPLIED>
<!ATTLIST particles1 pos_y CDATA #IMPLIED>
<!ATTLIST particles1 pos_z CDATA #IMPLIED>
<!ATTLIST particles1 vel_x CDATA #IMPLIED>
<!ATTLIST particles1 vel_y CDATA #IMPLIED>
<!ATTLIST particles1 vel_z CDATA #IMPLIED>
<!ATTLIST particles1 gra_x CDATA #IMPLIED>
<!ATTLIST particles1 gra_y CDATA #IMPLIED>
<!ATTLIST particles1 gra_z CDATA #IMPLIED>
<!ATTLIST particles1 wind_x CDATA #IMPLIED>
<!ATTLIST particles1 wind_y CDATA #IMPLIED>
<!ATTLIST particles1 wind_z CDATA #IMPLIED>

<!ELEMENT particles2 EMPTY>
<!ATTLIST particles2 tex CDATA #REQUIRED>
<!ATTLIST particles2 max CDATA #IMPLIED>
<!ATTLIST particles2 release_n CDATA #IMPLIED>
<!ATTLIST particles2 release_t CDATA #IMPLIED>
<!ATTLIST particles2 life CDATA #IMPLIED>
<!ATTLIST particles2 size CDATA #IMPLIED>
<!ATTLIST particles2 vel_var CDATA #IMPLIED>
<!ATTLIST particles2 color_x CDATA #IMPLIED>
<!ATTLIST particles2 color_y CDATA #IMPLIED>
<!ATTLIST particles2 color_z CDATA #IMPLIED>
<!ATTLIST particles2 color_u CDATA #IMPLIED>
<!ATTLIST particles2 pos_x CDATA #IMPLIED>
<!ATTLIST particles2 pos_y CDATA #IMPLIED>
<!ATTLIST particles2 pos_z CDATA #IMPLIED>
<!ATTLIST particles2 vel_x CDATA #IMPLIED>
<!ATTLIST particles2 vel_y CDATA #IMPLIED>
<!ATTLIST particles2 vel_z CDATA #IMPLIED>
<!ATTLIST particles2 gra_x CDATA #IMPLIED>
<!ATTLIST particles2 gra_y CDATA #IMPLIED>
<!ATTLIST particles2 gra_z CDATA #IMPLIED>
<!ATTLIST particles2 wind_x CDATA #IMPLIED>
<!ATTLIST particles2 wind_y CDATA #IMPLIED>
<!ATTLIST particles2 wind_z CDATA #IMPLIED>
```

Por último, de manera jerarquizada, podemos indicar dentro de cada objeto otros objetos hijos del actual, que heredarán las propiedades fijadas para éste.

24.3.2. Vehículos

Además de los anteriores, para los vehículos definimos varios nodos específicos que especializan la descripción del vehículo dentro del videojuego.

```
<!ELEMENT object (mesh?, position?, rotation?, scale?, physic?,
                  collision?, sound?, particles1?, particles2?, object*,
                  features, body, frwheel, flwheel, rrwheel, rlwheel)>
<!ATTLIST object type #FIXED "vehicle">
<!ATTLIST object name CDATA #REQUIRED>
<!ATTLIST object description CDATA #REQUIRED>
```

En el nodo principal incluimos atributos para identificar que se trata de un vehículo, el nombre y la descripción textual del mismo.

Tenemos un nodo de **características**, dónde podemos indicar los siguientes atributos:

- La capacidad máxima del vehículo para armamento
- La vida del vehículo
- La velocidad máxima
- La aceleración
- Parámetros referentes a la suspensión del vehículo
- El giro del vehículo ante el uso de las teclas de dirección
- El giro de las ruedas ante el avance y ante la dirección
- La distancia entre las ruedas
- La altura de la cámara interior del vehículo

```
<!ELEMENT features EMPTY>
<!ATTLIST features capacity CDATA #REQUIRED>
<!ATTLIST features health CDATA #REQUIRED>
<!ATTLIST features speed CDATA #REQUIRED>
<!ATTLIST features acceleration CDATA #REQUIRED>
<!ATTLIST features maxbodyZ CDATA #REQUIRED>
<!ATTLIST features maxbodyX CDATA #REQUIRED>
<!ATTLIST features bodyZ CDATA #REQUIRED>
<!ATTLIST features bodyX CDATA #REQUIRED>
<!ATTLIST features recovZ CDATA #REQUIRED>
<!ATTLIST features recovX CDATA #REQUIRED>
<!ATTLIST features turn CDATA #REQUIRED>
<!ATTLIST features wheelY CDATA #REQUIRED>
<!ATTLIST features wheelZ CDATA #REQUIRED>
<!ATTLIST features wheelDist CDATA #REQUIRED>
<!ATTLIST features cameraY CDATA #REQUIRED>
```

Además especificamos el **fuselaje** del vehículo y las cuatro **ruedas** con nodos especializados, ya que tendrán diferente tratamiento dentro de la lógica de escena:

```
<!ELEMENT body (mesh?, position?, rotation?, scale?, physic?,
                collision?, sound?, particles1?, particles2?, object*)>
<!ELEMENT frwheel (mesh?, position?, rotation?, scale?, physic?,
                collision?, sound?, particles1?, particles2?, object*)>
<!ELEMENT flwheel (mesh?, position?, rotation?, scale?, physic?,
                collision?, sound?, particles1?, particles2?, object*)>
<!ELEMENT rrwheel (mesh?, position?, rotation?, scale?, physic?,
                collision?, sound?, particles1?, particles2?, object*)>
<!ELEMENT rlwheel (mesh?, position?, rotation?, scale?, physic?,
                collision?, sound?, particles1?, particles2?, object*)>
```

24.3.3. Arenas

Al igual que en los vehículos, con las arenas tenemos un tratamiento XML adicional.

```
<!ELEMENT object (mesh?, position?, rotation?, scale?, physic?,
                  collision?, sound?, particles1?, particles2?, object*,
                  sky?, elements*)>
<!ATTLIST object type #FIXED "arena">
<!ATTLIST object name CDATA #REQUIRED>
<!ATTLIST object description CDATA #REQUIRED>
<!ATTLIST object f_factor CDATA #REQUIRED>
<!ATTLIST object num_obj CDATA #REQUIRED>
<!ATTLIST object time_ini CDATA #REQUIRED>
<!ATTLIST object time_obj CDATA #REQUIRED>
<!ATTLIST object size CDATA #REQUIRED>
<!ATTLIST object ranking CDATA #REQUIRED>
```

En el nodo principal incluimos atributos para identificar que se trata de una arena, el nombre, la descripción textual de la misma, y además:

- El factor de deslizamiento del suelo de la arena
- El número de objetivos que hay que capturar para acabar la arena
- El tiempo inicial con el que parte el jugador para capturar todos los objetivos
- El tiempo añadido al contador por cada objetivo capturado
- El tamaño de la arena (el mayor entre el ancho y el alto)
- La ruta relativa del fichero XML de ranking de la arena

Un nodo adicional que determina el **firmamento** de la arena, que tendrá un tratamiento diferente al resto de objetos de la arena:

```
<!ELEMENT sky (mesh?, position?, rotation?, scale?, physic?, collision?, sound?, object*)>
```

Varios nodos **elemento**s, que determinan las diferentes ubicaciones dónde pueden aparecer los elementos aleatorios de la escena, tales como los objetivos u otros elementos:

```
<!ELEMENT elements EMPTY>
<!ATTLIST elements x CDATA #IMPLIED>
<!ATTLIST elements y CDATA #IMPLIED>
<!ATTLIST elements z CDATA #IMPLIED>
```

25. Parámetros de la clase *Settings*

Esta clase implementa la gestión de parámetros personalizables por parte de los usuarios, tales como la configuración de video, sonidos y juego.

La funcionalidad de la clase y los parámetros personalizables de la aplicación ya han sido descritos en el apartado anterior, al describir el documento XML que almacena la configuración.

En resumen, se permite configurar el *display* para el sistema 3D de DirectX:

- *quality* para configurar la calidad general de video (0 personalizada, 1 alta, 2 media, 3 baja)
- *winfull* por si queremos (1) o no (0) pantalla completa
- *stadistics* por si queremos (1) o no (0) estadísticas de DirectX en pantalla
- *resolution* para la resolución de pantalla
- *filter* para el filtro usado en las texturas (0 puntual, 1 lineal, 2 anisotrópico)
- *midmapping* por si queremos (1) o no (0) activar el midmapping

El *sonido* para manipular el volumen de:

- *sound* para los efectos de sonido.

El *juego* permite mantener:

- *player_name* nombre del jugador

26. Parámetros de la clase *Menu*

Esta clase tan sólo hace llamada a las instancias apropiadas en cada momento, en función del entorno en el que se encuentre el interfaz de usuario.

Tendremos una única instancia de *Menu* en ejecución, y dependiendo del contexto de la aplicación, esta instancia hará llamada a las primitivas básicas (*update* y *render*) de los elementos apropiados para ese contexto.

Por ejemplo, si estamos en el menú inicial del juego, la instancia actualizará y dibujará la escena del logo con los anillos giratorios que aparece al iniciar la aplicación.

Si iniciamos una nueva partida y estamos seleccionando la arena o el vehículo, actualizará y dibujará la arena o vehículo que tengamos seleccionados en cada momento.

Por último, durante el juego actualizará y dibujará la instancia de la clase *Game*, dejándolo de actualizar si pausamos el juego.

27. Parámetros de la clase *GUI*

Esta clase implementa el interfaz de usuario, incluyendo toda la lógica interna de navegación de menús y opciones de usuario.

El siguiente diagrama resume la navegación de pantallas en la aplicación, dónde aparecen estados, acciones y transiciones. No es necesaria una descripción detallada de las pantallas, ya que resulta suficiente con el texto descriptivo que se acompaña en cada una:

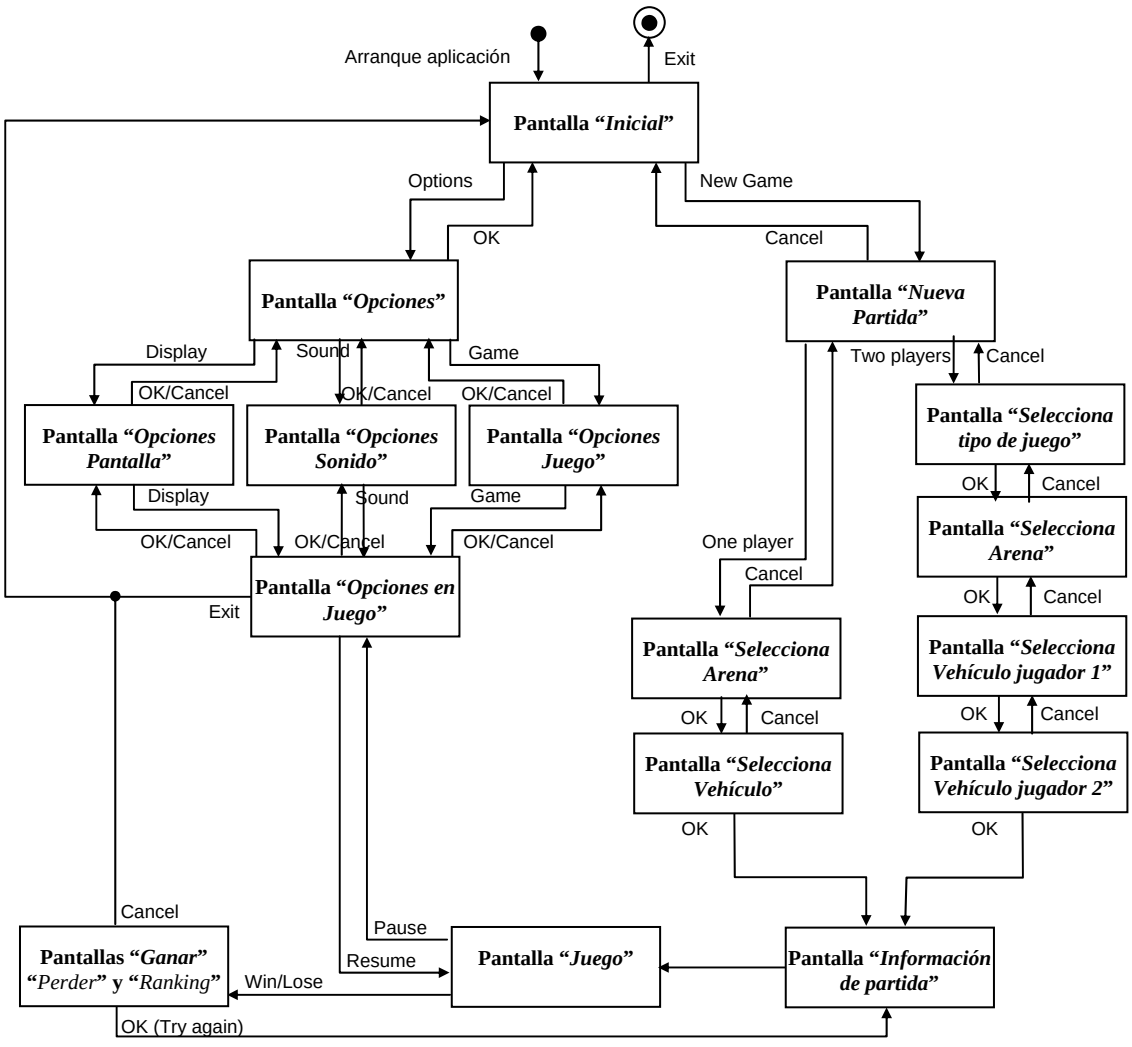


Figura 60 : Diagrama de pantallas del GUI

Para más información sobre los controles del juego, consultar los apartados *9.1. Manejo del GUI* y *9.2. Manejo del juego* dónde se detallan los controles específicos del GUI.

28. Parámetros de la clase *3DLight*

Clase para definir parámetros de luz ambiente, especular y difusa dentro de la aplicación. Para facilitar la programación no se implementan focos direccionales de luz, sino que sólo se configura la luz global de la escena.

Se implementa una clase que añade funcionalidades al objeto de luz de DirectX, como por ejemplo el apagado y encendido de la luz de forma atenuada, muy propio para los cambios de escena en la aplicación. Sin embargo no hay muchas más características añadidas al objeto, que simplemente ofrece un wrapper estándar que integra la luz como un objeto más en el motor gráfico.

29. Parámetros de la clase *3DCamera*

Clase que abstrae el concepto de cámara para el dispositivo gráfico, con todos sus parámetros de foco, posición y rotación. Dentro de esta clase se implementan los algoritmos de seguimiento de objetos, cambios de vista, etc.

Los algoritmos de seguimiento se hacen focalizando un vehículo y moviendo la cámara de manera retardada al movimiento del objeto (para crear un efecto elástico en el seguimiento).

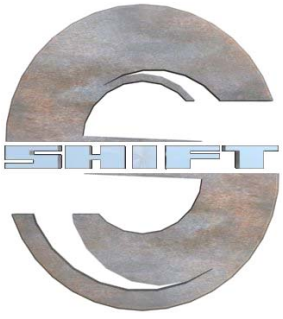
Además, la distancia de seguimiento se calcula en función de la velocidad del objeto, de forma que a mayor velocidad se amplía la distancia y viceversa.

Para más información sobre las cámaras disponibles, puede consultar el apartado *9.3. Vistas de la cámara*, dónde se comenta cada uno de los tipos de cámara implementados en esta clase.

Capítulo 8

Diseño del motor de partículas

16 de marzo de 2008



Capítulo 8: Diseño del motor de partículas

Tabla de apartados

Lista de figuras

30. Introducción (*motor de partículas*)

Permite gestionar sistemas de partículas en la escena. Utilizaremos estos sistemas para modelar diferentes fenómenos con mayor realismo, tales como el humo de los vehículos, nieve, chimeneas, explosiones, etc.

Dentro del esquema general, el módulo de partículas queda limitado por el ámbito rayado:

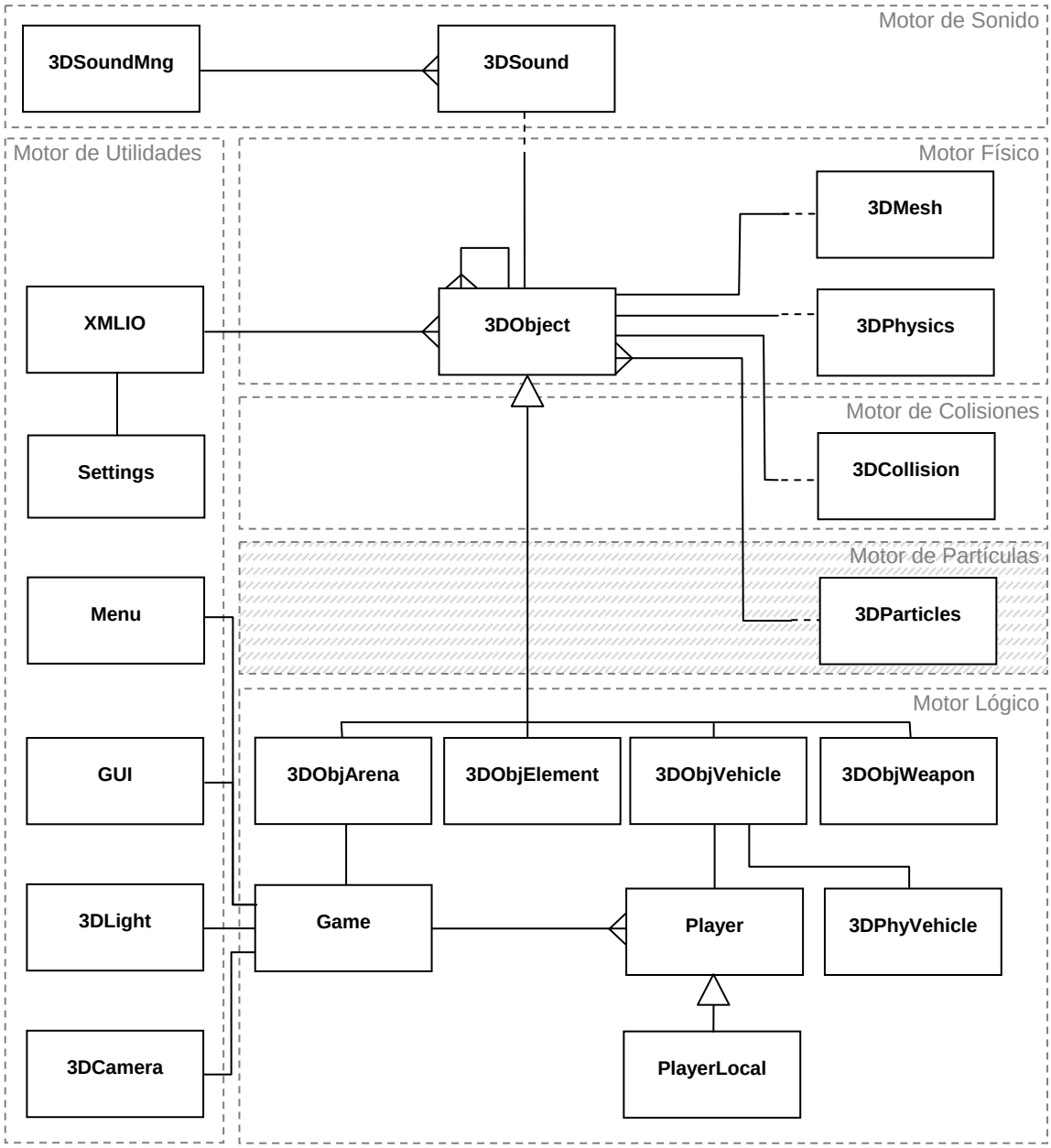


Figura 61 : Límites del motor de partículas

Cómo refleja en el diagrama, el módulo de partículas queda implementado por la clase:

- **3DParticles**
  Gestiona sistemas de partículas con varios parámetros físicos que permitirán adaptarlos a los resultados necesarios en cada momento.

A continuación veremos en detalle los aspectos controlados por el motor de partículas.

31. Parámetros de la clase *3DParticles*

Gestiona sistemas de partículas con varios parámetros físicos que permitirán adaptarlos a los resultados necesarios en cada momento.

31.1. Usos de las partículas

Los sistemas de partículas tienen diversos usos dentro de la aplicación, e incluso se pueden encontrar muchos más debido a la versatilidad con la que se cargan a través de los ficheros XML.

Dentro del videojuego se utilizan para diferentes finalidades, tales como el humo del tubo de escape de los vehículos, el humo de las chimeneas:

Figura 62 : Partículas para modelar humo

Diversos fenómenos atmosféricos, como la nieve o la lluvia:

Figura 63 : Partículas para modelar efectos atmosféricos

Traza de los cohetes y explosiones:

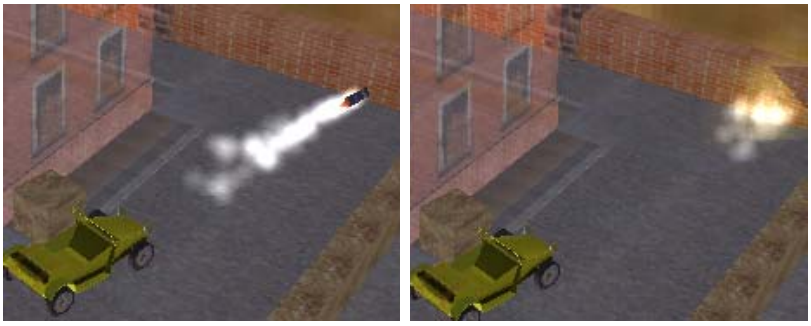


Figura 64 : Partículas para modelar cohetes y explosiones

O incluso para destacar algunos elementos o acciones del jugador (como capturar elementos):

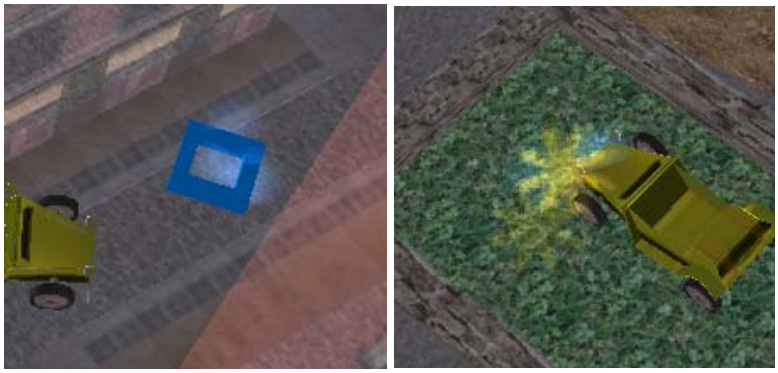


Figura 65 : Partículas para destacar elementos y acciones

31.2. Parámetros de configuración

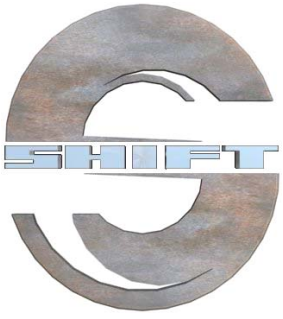
Como ya se ha visto en el detalle de la clase XMLIO (apartado *24. Parámetros de la clase XMLIO*) las partículas se configuran con diferentes parámetros que establecen lo siguiente:

- Fichero .BMP o .JPG que define la textura de cada partícula
- Máximo número de partículas
- Número de partículas liberadas en cada iteración
- Tiempo para cada iteración
- Vida en segundos de cada partícula liberada
- Tamaño en unidades de la escena de cada partícula
- Aleatoriedad de la dirección con la que son liberadas las partículas
- Color y transparencia de las partículas
- Posición relativa al objeto dónde se liberan las partículas (valores para X, Y, Z)
- Velocidad absoluta de las partículas (valores para X, Y, Z)
- Fuerza gravitacional de las partículas (valores para X, Y, Z)
- Fuerza del viento sobre las partículas (valores para X, Y, Z)

Capítulo 9

Diseño del motor lógico

16 de marzo de 2008



Capítulo 9: Diseño del motor lógico

Tabla de apartados

Lista de figuras

32. Introducción (*motor lógico*)

El motor lógico implementa las características propias del videojuego, necesarias para algunos objetos de la escena y para el funcionamiento interno del videojuego. Este módulo personaliza el comportamiento del videojuego y complementa el funcionamiento genérico del motor gráfico.

Dentro del esquema general, el módulo lógico queda limitado por el ámbito rayado:

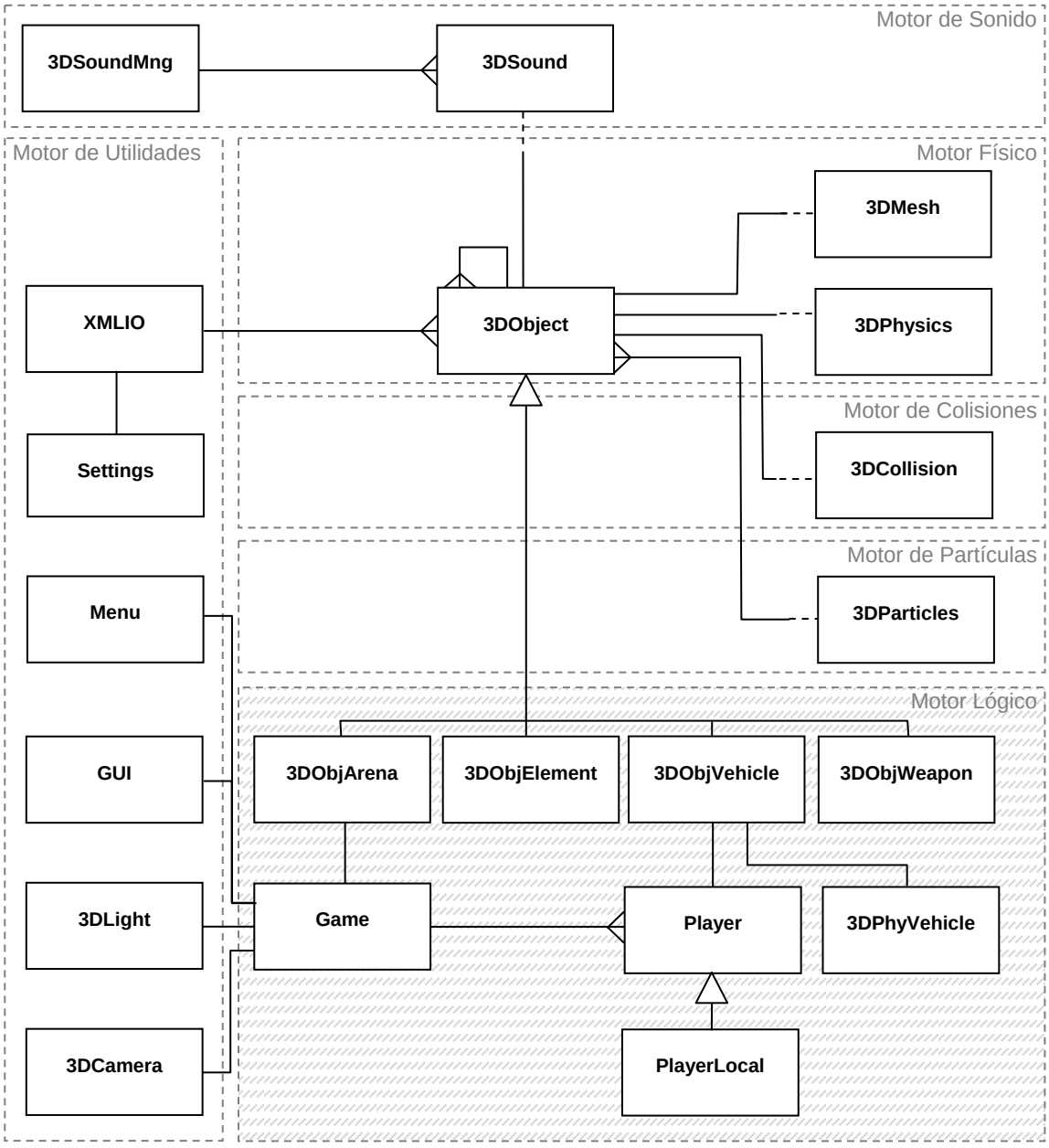


Figura 66 : Límites del motor lógico

Cómo refleja en el diagrama, el módulo lógico consta de las siguientes clases:

- **3DObjArena**
  Tipo de objeto con características especiales para contener una arena concreta, incluyendo gran cantidad de parámetros de juego que implementan la mayor parte de la lógica.

- **3DObjVehicle**
  Tipo de objeto con parámetros específicos para implementar los vehículos, incluyendo física y sonidos adicionales para ellos.

- **3DObjElement**
  Tipo de objeto con características especiales para contener los elementos aleatorios que aparecen en la arena, tales como cohetes, vida, interrogación y el elemento objetivo.

- **3DObjWeapon**
  Tipo de objeto con parámetros específicos para implementar las armas del juego, tales como minas o cohetes.

- **3DPhyVehicle**
  Clase que modela el comportamiento físico del vehículo, ya que dicho comportamiento es diferente al del resto de los objetos en escena.

- **Game**
  Clase que gestiona toda la información referente a una partida en curso, desde el tipo de juego, hasta la arena seleccionada y los jugadores que participan.

- **Player**
  Clase que gestiona todos los datos de cada uno de los jugadores de la partida en curso.

- **PlayerLocal**
  Tipo de jugador local en el equipo. Esta clase hereda de la anterior, permitiendo así implementar otra clase paralela PlayerRemote que gestione jugadores remotos en el caso de ampliar el juego con un modo multijugador y motor de red.

A continuación veremos en detalle cada uno de estos aspectos controlados por el motor lógico.

33. Parámetros de la clase *3DObjArena*

Este tipo de objeto incluye una lógica adicional para contener la arena del juego. Respecto los objetos tradicionales, se gestiona además:

- El nombre de la arena
- La descripción textual de la arena
- El factor de deslizamiento del suelo de la arena
- El número de objetivos que hay que capturar para acabar la arena
- El tiempo inicial con el que parte el jugador para capturar todos los objetivos
- El tiempo añadido al contador por cada objetivo capturado
- El tamaño de la arena (el mayor entre el ancho y el alto)
- La ruta relativa del fichero XML de ranking de la arena
- Lista de ubicaciones dónde pueden aparecer los elementos aleatorios de la escena.
- El firmamento, que se mantiene por separado para desactivarlo en algunas ocasiones (por ejemplo cuando estamos seleccionando una arena)

Parámetros de la clase *3DObjVehicle*

Este tipo de objeto incluye una lógica adicional para contener cada vehículo del juego. Respecto los objetos tradicionales, se gestionan además:

- El nombre del vehículo
- La descripción textual del vehículo
- La capacidad máxima del vehículo para armamento
- La vida del vehículo
- La velocidad máxima
- La aceleración
- Parámetros referentes a la suspensión del vehículo
- El giro del vehículo ante el uso de las teclas de dirección
- El giro de las ruedas ante el avance y ante la dirección
- La distancia entre las ruedas
- La altura de la cámara interior del vehículo
- El fuselaje y las cuatro ruedas, que se mantienen por separado ya que llevan una física especial
- Una física especial de vehículos, que veremos en el apartado *36. Parámetros de la clase 3DPhyVehicle*

34. Parámetros de la clase *3DObjElement*

Tipo de objeto con características especiales para contener los elementos aleatorios que aparecen en la arena. Respecto los objetos tradicionales se gestiona además el tipo de elemento, que puede ser:

- Elemento objetivo
- Elemento vida
- Elemento cohete
- Elemento mina
- Elemento interrogación

Y se ofrecen primitivas para:

- Mostrar el siguiente elemento
- Comprobar estado del elemento
- Efectuar acción sobre el jugador que captura el elemento

Más detalles sobre esta clase en el apartado *6.3. Elementos*.

35. Parámetros de la clase *3DObjWeapon*

Tipo de objeto con parámetros específicos para implementar las armas del juego. Respecto los objetos tradicionales se gestiona además el tipo de arma, que puede ser:

- Arma mina
- Arma cohete

Y se ofrecen primitivas para:

- Activar el arma
- Comprobar estado del arma
- Efectuar acción sobre el jugador que colisiona con el arma

Más detalles sobre esta clase en el apartado *6.4. Armas*.

36. Parámetros de la clase *3DPhyVehicle*

Los vehículos son los objetos protagonistas del videojuego, y por lo tanto deben implementar una física especial que incremente su realismo en escena respecto el resto de objetos.

Esta física especial depende de cada vehículo, por lo que en cada fichero de configuración deberán especificarse los parámetros especiales que lo modelen (más detalles en el apartado *24.3.2. Vehículos*).

36.1. Doble fricción lineal

Este concepto resume el hecho de que los vehículos no presentan la misma fricción respecto cantidades de movimiento lineal aplicadas en una u otra dirección.

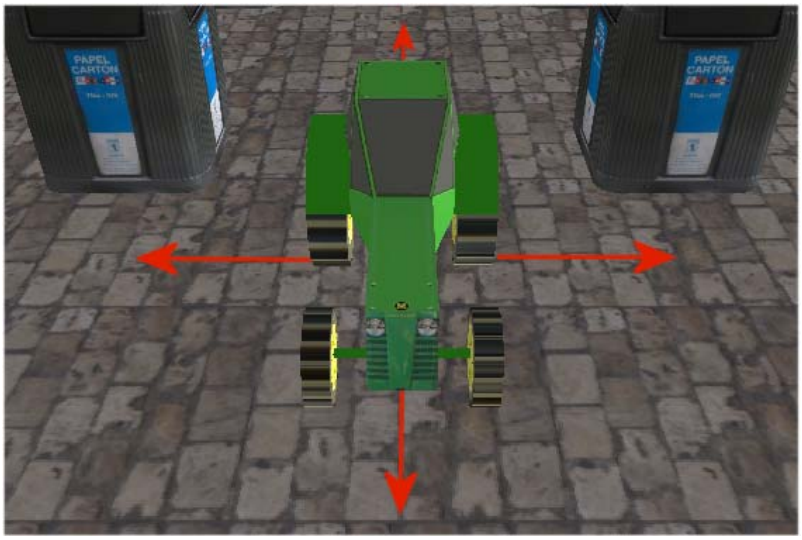


Figura 67 : Doble fricción lineal que presentan los vehículos

Como se observa en la figura, un vehículo no responde igual ante un movimiento frontal (o trasero) que ante un movimiento lateral. Como es lógico los desplazamientos laterales tienen una mayor fricción que los frontales o traseros, debido al propio comportamiento de los neumáticos.

Para modelar esto, cualquier movimiento lineal aplicado sobre un vehículo se va a desglosar en dos componentes, la que se corresponde con la dirección del vehículo y otra ortogonal. Sobre cada una de las componentes se aplicarán fricciones diferentes, siendo superior la fricción aplicada sobre la componente ortogonal.

36.2. Punto de giro móvil

Este fenómeno corresponde con los conceptos de sobreviraje y subviraje de los vehículos. La idea es que los vehículos giran respecto puntos de pivote diferentes en cada momento.

Figura 68 : Punto de giro móvil en los vehículos

En un comportamiento normal, como el indicado en el vehículo de la izquierda, el coche realiza giros respecto un pivote cercano al eje trasero.

Si su velocidad es elevada y realizamos un giro brusco, el coche rotará respecto un punto cada vez más cercano al eje delantero, tal y como se muestra en la figura derecha. Esto es más notable ante el uso del freno de mano, que bloquea las ruedas traseras y permite adelantar aún más el pivote de giro al eje frontal.

En definitiva, considerando ciertos factores, podemos mover el punto pivote de giro ligeramente sobre un pequeño segmento entre el eje trasero y delantero del vehículo, cómo se ha representado en las dos figuras adjuntas.

36.3. Comportamiento de las ruedas

Otro de los aspectos delicados es el comportamiento de las ruedas. Por una parte la rotación de las ruedas debe coincidir con la velocidad lineal del vehículo, de forma que corresponda su avance con la distancia perimetral recorrida. En este sentido hay que tener especial cuidado con el tamaño de los neumáticos, sobre todo en casos como en el tractor, dónde las ruedas traseras son mayores que las delanteras.

También es importante definir el giro de las ruedas delanteras ante giros del vehículo. Cuando giramos a un lado las ruedas tardan cierto tiempo en llegar a su máximo y cuando dejamos de girar tardarán cierto tiempo hasta que recuperen su estado normal.

Otro aspecto importante es el bloqueo de las ruedas traseras ante el uso del freno de mano, que deben detener su giro en caso de dicha acción.

36.4. Comportamiento del fuselaje

Por último, el comportamiento del fuselaje depende de los cambios físicos aplicados sobre el vehículo, de forma que ante aceleraciones, frenazos, curvas o colisiones, el fuselaje debe experimentar cierto balanceo que modele la amortiguación del vehículo.

Figura 69 : Comportamiento del fuselaje

37. Parámetros de la clase *Game*

Dentro de esta clase se gestionan las instancias propias del entorno de juego, tales como:

- La arena
- Los jugadores

Además, crea un nexo entre diferentes instancias del motor gráfico, utilizando sus primitivas para crear el contexto del juego a medida. Las instancias que maneja son:

- El menú
- El GUI
- La luz
- La cámara

Con todo ello implementa la lógica propia del juego, como el contexto con el que se gana, el contexto con el que se pierde, la lógica de aparición de los objetivos, la lógica del contador de tiempo, etc.

38. Parámetros de las clases *Player* y *PlayerLocal*

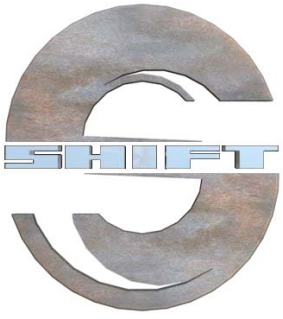
Estas dos clases gestionan los datos de cada uno de los jugadores de la partida y las acciones que realizan mediante los controles del teclado.

Se ha hecho una clase base genérica (*Player*) y otra hija para implementar el caso concreto de un jugador local en el equipo (*PlayerLocal*). Esta estructura permitirá ampliar el videojuego mediante otra clase paralela PlayerRemote que gestione jugadores remotos (más detalles en el apartado *51.1. Sistema multijugador*).

Capítulo 10

Diseño 2D y 3D

16 de marzo de 2008



Capítulo 10: Diseño 2D y 3D

Tabla de apartados

Lista de figuras

39. Introducción (*diseño 2D y 3D*)

El diseño y construcción de los modelos gráficos es una etapa estética vital, ya que el aspecto del videojuego dependerá de la calidad de los modelos aquí desarrollados, pero también es una etapa dónde hay que tener en cuenta la optimización, ya que la mayor parte del tiempo de procesador de la tarjeta gráfica se va a utilizar para mover los vértices de cada modelo construido.

Nos proponemos cumplir con cada modelo las siguientes cualidades:

- **Eficiente**
  No hay que olvidar que el modelado es un punto clave en cuanto a la eficiencia final de la aplicación, por lo que los modelos construidos tratarán de ser lo más simples posibles para minimizar el número de polígonos finales y lograr mayor fluidez en la simulación.

- **Realista**
  Y en contra de lo anterior, la eficiencia no debería suponer menor realismo, por lo que también sopesaremos esta cualidad, ya que la estética del videojuego va a depender casi por completo de esta fase.

- **Independiente**
  Se modelará cada objeto simple de forma independiente, lo que permitirá componer con unos pocos objetos simples gran cantidad de objetos complejos y por tanto gran cantidad de escenas diferentes.

- **Acorde con la licencia GPLv3**
  Este es un apartado muy importante, en el que hay que tener especial cuidado con las herramientas y recursos utilizados para no infringir la licencia copyleft con la que se publica la aplicación.

En principio el modelado constará de dos pasos que se pueden realizar de forma paralela para cada objeto a desarrollar:

- **Texturas 2D**
  Son imágenes 2D que se mapean por la superficie del objeto 3D. Una buena elección de estas texturas dotará al modelo de mayor realismo, llegando incluso a ocultar la geometría simplista que se utilice para el objeto.

- **Modelos 3D**
  Es la propia geometría de los modelos 3D, que viene dada por una estructura de datos que almacena vértices, uniones, normales y mapeado de las anteriores texturas.

En cada uno de estos dos pasos tenemos que determinar los formatos elegidos:

- **Para las texturas 2D** (.JPG y .BMP)
  El formato elegido es .JPG al garantizar una buena calidad sin exceder el tamaño en disco, que podría aumentar en exceso el tamaño final de la aplicación. En los casos en los que el tamaño de la imagen no sea muy elevado, se podrá utilizar el formato .BMP.

- **Para los modelos 3D** (.X)
  Existen diversos formatos que permiten mantener un objeto 3D, algunos son específicos para un motor de programación y otros son de propósito general. En nuestro caso la aplicación se programará directamente con DirectX por lo que usaremos su formato nativo (.X)

Y una vez determinados los dos formatos, elegimos los programas de desarrollo, los cuales deben ser acordes con la licencia GPLv3:

- **Para las texturas 2D** (Gimp)
  Programa de referencia dentro de la comunidad de software libre para edición fotográfica.

- **Para los modelos 3D** (Blender)
  Un editor 3D libre cuyo entorno gráfico lo hace manejable y sencillo.

Para el diseño de modelos fotográficos 2D vamos a considerar tres fuentes de recursos:

- **Diseños ad-hoc**
  Se han realizado diseños propios para los vehículos y ruedas, evitando en todo caso tomar patrones de vehículos comerciales.

- **Fotografía propia**
  Se han tomado gran cantidad de imágenes fotográficas propias, especialmente para los edificios, firmamentos y otros objetos.

- **Galerías gratuitas de Internet**
  Se han considerado además algunas texturas especiales descargadas de galerías gratuitas de Internet, aunque estas fuentes se han utilizado para la menor parte de las texturas.

Los modelos 3D diseñados se dividen en:

- **Fuselajes de vehículos**
  Son los modelos "body***X***.x", con X un entero.

- **Ruedas de vehículos**
  Son los modelos "wheel***X***.x" , con X un entero.

- **Edificios**
  Son los modelos "building***X***.x" , con X un entero.

- **Suelos**
  Son los modelos "floor***X***.x" , con X un entero.

- **Aceras**
  Son los modelos "sidewalk***X***.x" , con X un entero.

- **Firmamentos**
  Son los modelos "sky***X***.x" , con X un entero.

- **Otros objetos**
  El resto de los modelos

En los siguientes apartados se detallará el texturizado y posterior modelado de cada uno de los tipos de objetos anteriores.

40. Edición fotográfica

40.1. Diseños ad-hoc

Se tratan de diseños gráficos propios, que suponen bastante esfuerzo para su elaboración, por lo que sólo serán utilizados para algunos objetos esenciales. Estos diseños reúnen en una misma imagen todos los aspectos gráficos del objeto en cuestión

En la siguiente imagen podemos ver la textura utilizada para el vehículo *Classic*:

Figura 70 : Diseño ad-hoc de un vehículo

Por si sola la imagen puede no aportar detalles del objeto, ya que será en la etapa de modelado cuando el mapeo sobre los diferentes polígonos del objeto muestren el resultado final:

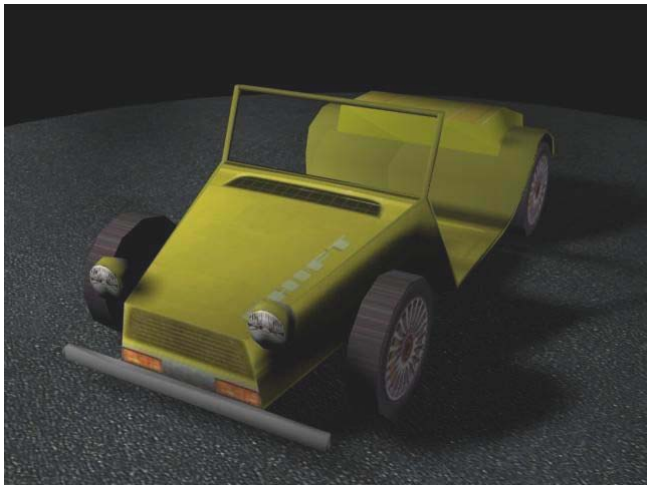


Figura 71 : Resultado final de un diseño ad-hoc

40.2. Fotografía propia

La fotografía propia permite obtener texturas de gran calidad en poco tiempo y esfuerzo, sin embargo es necesario considerar algunas ideas para obtener la máxima calidad:

- Se deben realizar de forma ortogonal a la textura a capturar
- Si es posible se deberían hacer con zoom y a la máxima distancia posible, ya que de esta forma se reduce el efecto de la perspectiva y podemos obtener imágenes *planas*
- Debemos tener luz ambiental uniforme y poco focalizada, ya que de esta forma evitaremos reflejos, malas iluminaciones o sombras, que pueden falsear el realismo de la textura
- Tras tomar cada fotografía, será necesario realizar un retoque posterior para reducir su tamaño, rotarla, recortar el patrón deseado y unificar tonos para que no se note la repetición del patrón.

Teniendo en cuenta estos aspectos, la textura final puede tener el siguiente aspecto (este ejemplo se ha obtenido a partir de una fotografía propia al palacio real de Madrid):

Figura 72 : Fotografía adaptada

Y el modelo dónde la apliquemos quedará del siguiente modo:

Figura 73 : Resultado final de una fotografía adaptada

40.3. Galerías gratuitas de Internet

En este caso se han obtenido algunas texturas adicionales de galerías gratuitas de Internet. Sólo se han usado para algunas texturas de metales, agua y madera, siendo la menor parte de esta procedencia.

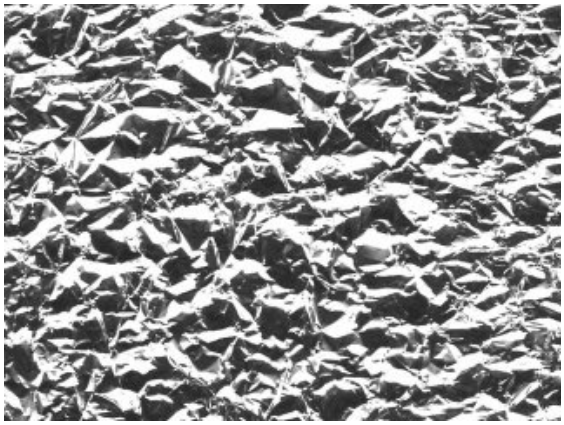


Figura 74 : Textura externa

El resultado final de estas texturas sobre algunos objetos de la escena:

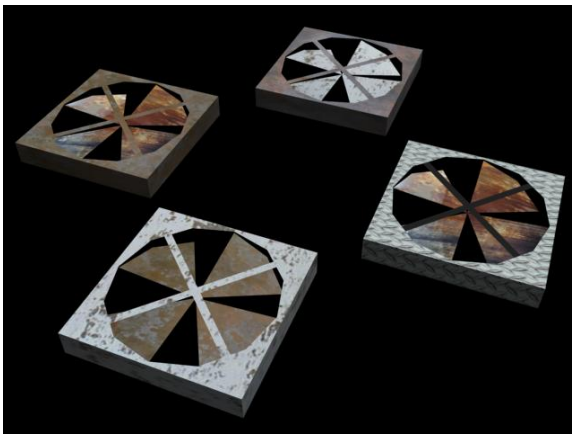


Figura 75 : Resultado final de las texturas externas

41. Modelado 3D

41.1. Fuselajes de vehículos

El modelado del fuselaje de los vehículos es quizás la parte más complicada, ya que la geometría de estos objetos es compleja y su apariencia es preferente.

El primer paso es encontrar un *blueprint* (esquema de la planta, alzado y perfil) de un vehículo comercial con el fin de usarlo como guía sobre el vehículo ad-hoc que vamos a diseñar. En todo caso, como el diseño de los vehículos del juego es propio (no se basan en ningún vehículo comercial), el *blueprint* sólo nos será útil para no perder las proporciones en el modelo.

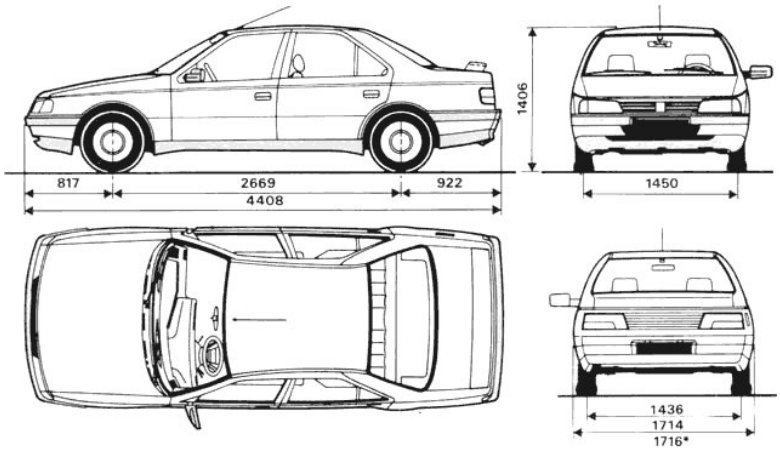


Figura 76 : Blueprint de un vehículo comercial

Tras tomar el blueprint se colocan cuatro planos en el espacio con las imágenes superior, lateral, frontal y posterior del blueprint tal y como se muestra en la figura. Tras esto ya tenemos el marco inicial de trabajo para el modelado.

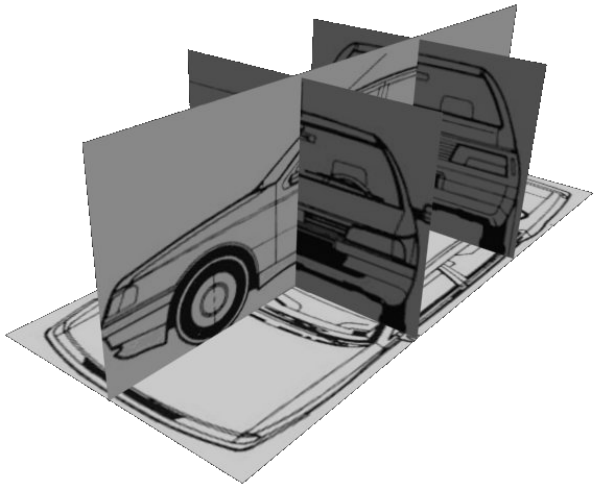


Figura 77 : Marco de trabajo para el modelado

Para modelar el vehículo partimos de un único polígono situado en alguna parte del fuselaje. A partir de dicho polígono vamos a ir construyendo el modelo, duplicando las aristas y uniendo vértices. Además debemos ir situando correctamente cada vértice en todas sus vistas, de manera que vayamos dando forma al vehículo inicialmente diseñado.

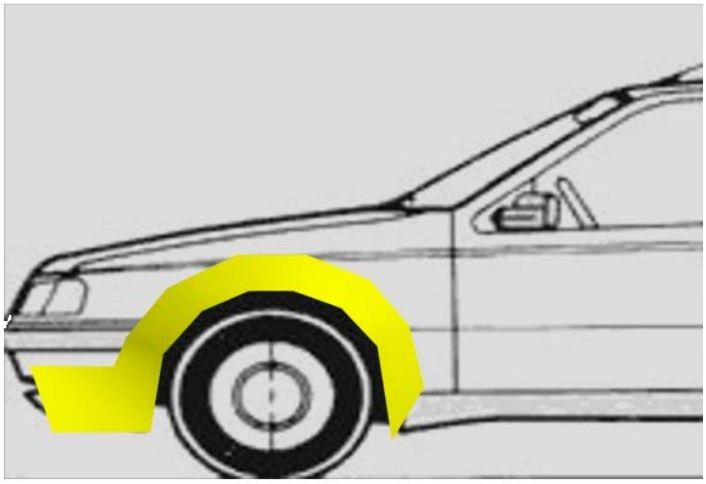


Figura 78 : Modelado del faldón en vista lateral

Como se puede observar en perspectiva, cada uno de los polígonos y/o vértices se va situando correctamente en la escena para adaptarse al fuselaje del vehículo diseñado (para ello haremos uso de diferentes puntos de vista y las guías colocadas del *blueprint*).

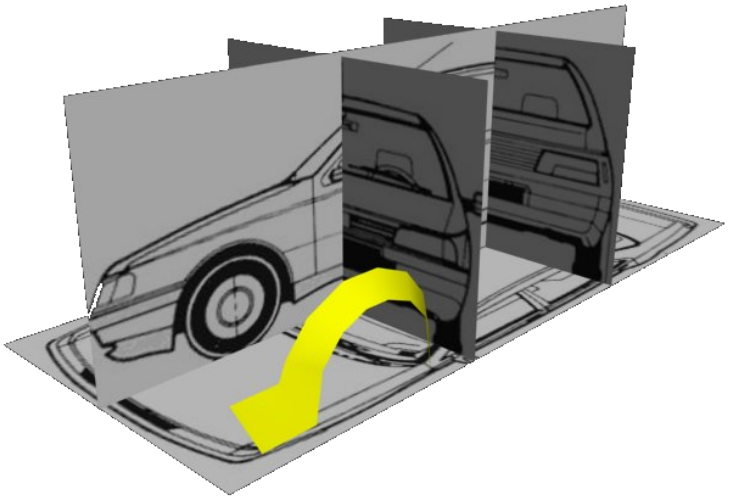


Figura 79 : Modelado del faldón en perspectiva

Duplicando aristas y uniendo vértices podemos terminar fácilmente un lateral del vehículo.

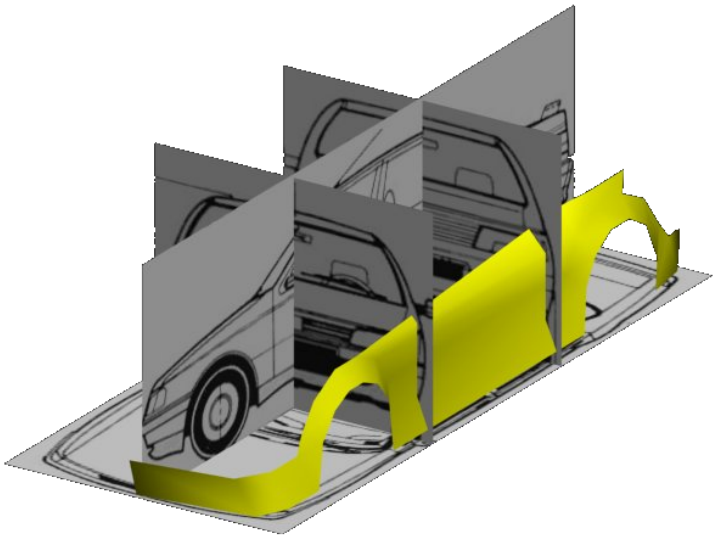


Figura 80 : Lateral del vehículo en perspectiva

Del mismo modo podemos continuar con la parte superior del vehículo hasta terminar una mitad completa del modelo.

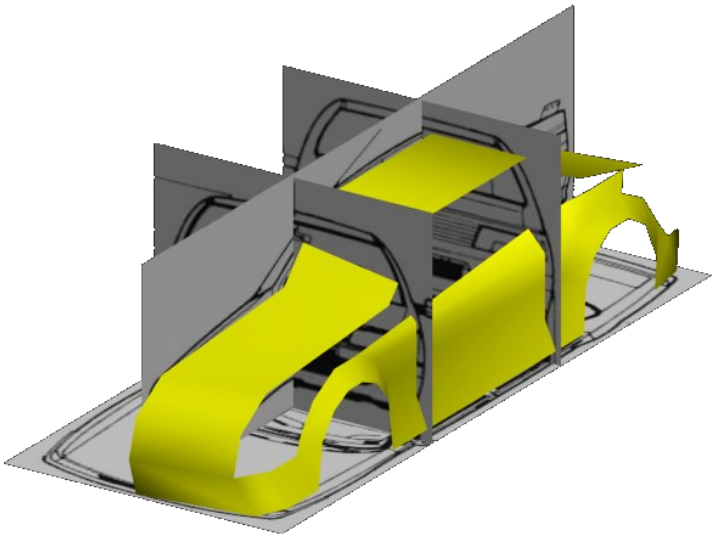


Figura 81 : Lateral y superior del vehículo en perspectiva

Finalmente sólo será necesario copiar la mitad del coche, reflejarla y fusionar los vértices centrales para finalizar el modelo. Además, lo más común es que esos vértices centrales puedan eliminarse, ya que no aportan ninguna geometría al modelo

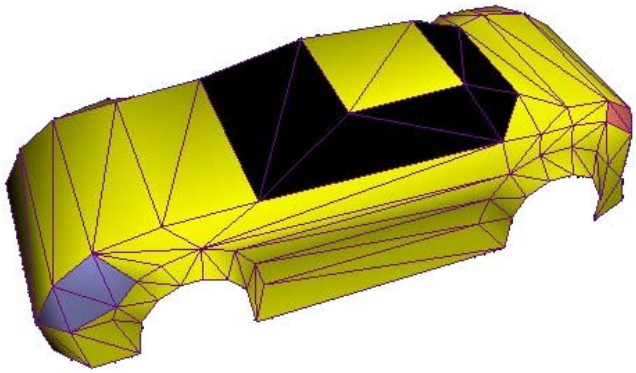


Figura 82 : Fuselaje del vehículo finalizado

Ahora pasamos a diseñar las texturas del vehículo que posteriormente aplicaremos sobre el modelo. Para ello basta con tomar imágenes de la estructura de alambres del modelo (superior, lateral, frontal y trasera) y componerlas en una única imagen de manera que se minimice la superficie final.

Tras componer esta imagen sólo tenemos que decorar el vehículo considerando los límites definidos por la estructura de alambres.

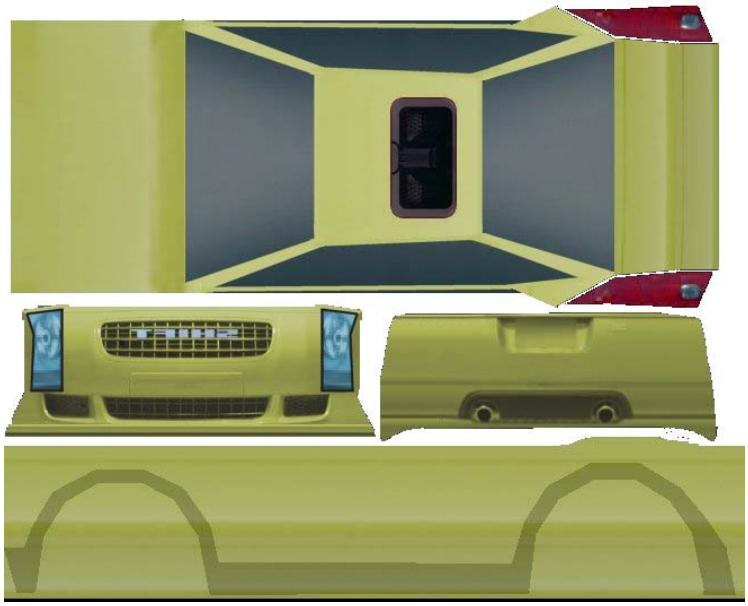


Figura 83 : Textura final para el fuselaje del vehículo

Por último solo queda fijar la textura anterior al modelo 3D. Primero se asocia la textura con el modelo y luego se ajusta para cada conjunto de polígonos según las vistas que contenga (superior, lateral, frontal y trasera).

Figura 84 : Resultado final del vehículo (las ruedas van aparte)

41.2. Ruedas de vehículos

Su proceso de modelado sigue un proceso similar al del fuselaje de los vehículos, aunque en este caso el modelo se hace directamente con un cilindro. Además, la textura debe aplicarse lo más centrada posible, para que al rodar en la aplicación final no se note ningún tipo de oscilación.

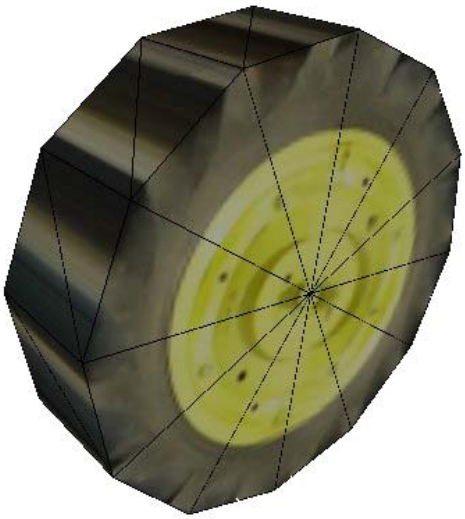


Figura 85 : Rueda de vehículo

41.3. Edificios

El modelado de estos objetos es bastante sencillo, basándose únicamente en el uso de cubos u otras formas simples. Dónde realmente reside el esfuerzo de estos objetos es en las texturas, a partir del tratamiento de fotografías propias de edificios.

Figura 86 : Resultado final de un edificio

41.4. Suelos

En este caso, las carreteras se componen únicamente de un plano simple. Es necesario que la textura pueda combinarse a modo de puzzle, para que la concatenación de diferentes trozos no sea apreciable. De nuevo la mayor dificultad reside en la elaboración de una buena textura.

Figura 87 : Textura adecuada para un suelo

Figura 88 : Resultado final del suelo

41.5. Aceras

Al igual que en el caso anterior, las aceras se modelan con un plano simple, algo más elevado que el plano del suelo. La textura vuelve a ser importante para que su repetición a modo mosaico no sea apreciable.

Figura 89 : Textura adecuada para las aceras

Figura 90 : Resultado final de las aceras

41.6. Firmamentos

Los firmamentos se refieren al cielo y el background de cada entorno 3D. En la aplicación se tomarán como una esfera o semiesfera en la que se consideran únicamente las caras internas. Desde cierta distancia el modelo puede parecer poco realista con este tipo de firmamento.

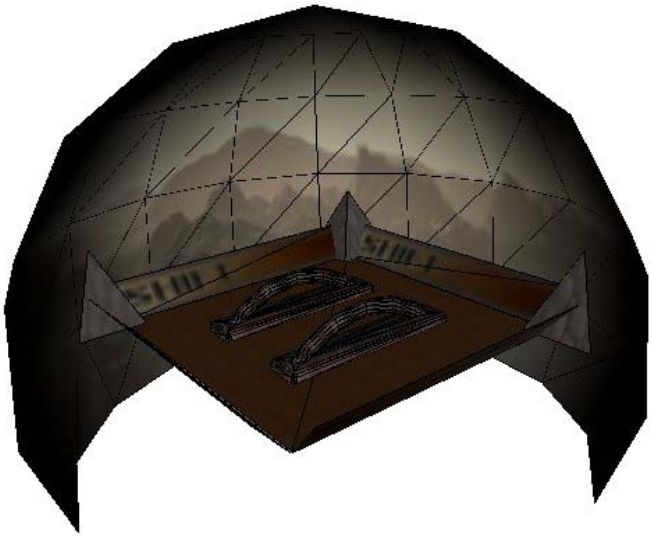


Figura 91 : Modelo de esfera para el firmamento

Pero sin embargo, desde dentro de la esfera el resultado es bastante aceptable.

Figura 92 : Resultado final del firmamento

Por lo tanto la esfera deberá escalarse de manera que contenga a todos los objetos de la escena para que el entorno tenga una apariencia realista.

41.7. Otros objetos

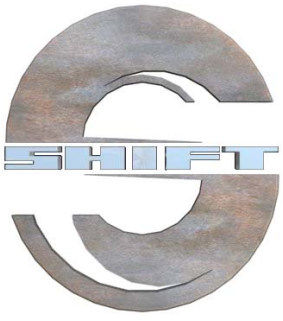
Su modelado es muy sencillo, haciendo uso de las primitivas básicas (cubos, conos, etc.) y en la mayoría de los casos no requieren texturas o son también sencillas.

Figura 93 : Cubo de reciclaje

Capítulo 11

Implementación

16 de marzo de 2008



Capítulo 11: Implementación

Tabla de apartados

42. Introducción (*implementación*)

A continuación se incluye la documentación del código auto generada mediante la herramienta *Doxigen*. Todo el código se encuentra comentado para que sea procesado por esta herramienta, lo que ofrece gran versatilidad de cara a cambios posteriores.

En todo caso, la documentación completa y detallada del código se encuentra en Internet, en la URL: http://shift.delblanco.es/develop/prog/index.html

43. Tecnologías Utilizadas

Para la programación técnica del videojuego se han utilizado las siguientes herramientas:

- Lenguaje C++, dónde todo el código sigue una estricta orientación a objetos impuesta por la propia naturaleza del proyecto. Esto es debido a que toda la jerarquía de clases diseñada modela de forma directa la escena 3D a gestionar en el código.

- DirectX 9.0 SDK (Summer 2004), que incluye las librerías y recursos necesarios para el manejo de la plataforma DirectX. Su licencia de uso (EULA) se puede ver en el apartado *52. ANEXO: Licencia DirectX 9.0c SDK*, dónde se permite el uso para el desarrollo y distribución de software.

- Compilador en línea de comandos CL de Microsoft, incluido en el paquete de desarrollo Visual C++ Toolkit 2003. Su uso es en línea de comandos mediante un fichero de dependencias al estilo Makefile (nmake de Microsoft). Su licencia (EULA) permite el desarrollo y distribución de software puede verse en el apartado *53. ANEXO: Licencia Visual C++ Toolkit 2003*.

44. Estructura de Directorios

Directorio *Raíz*

- archivo **Main.cpp**
- archivo **Main.h**
  Funciones principales.

Directorio *Units*

- archivo **3DCamera.cpp**
- archivo **3DCamera.h**
  Clase para manejar la cámara 3D de la escena.
- archivo **3DCollision.cpp**
- archivo **3DCollision.h**
  Para manejar las colisiones de los objetos de la escena.
- archivo **3DLight.cpp**
- archivo **3DLight.h**
  Clase para manejar las luces 3D de la escena.
- archivo **3DMesh.cpp**
- archivo **3DMesh.h**
  Para manejar los modelos gráficos de los objetos de la escena.
- archivo **3DObjArena.cpp**
- archivo **3DObjArena.h**
  Para manejar las arenas de la escena.
- archivo **3DObject.cpp**
- archivo **3DObject.h**
  Para manejar los objetos de la escena.
- archivo **3DObjElement.cpp**
- archivo **3DObjElement.h**
  Para manejar los elementos de la escena.
- archivo **3DObjVehicle.cpp**
- archivo **3DObjVehicle.h**
  Para manejar los vehículos de la escena.
- archivo **3DObjWeapon.cpp**
- archivo **3DObjWeapon.h**
  Para manejar las armas de la escena.
- archivo **3DPhysics.cpp**
- archivo **3DPhysics.h**
  Para manejar la física de los objetos de la escena.
- archivo **3DParticles.cpp**
- archivo **3DParticles.h**
  Para manejar los sistemas de partículas de la escena.
- archivo **3DPhysics.cpp**
- archivo **3DPhysics.h**
  Para manejar la física genérica de los objetos de la escena.

- archivo **3DPhyVehicle.cpp**
- archivo **3DPhyVehicle.h**

  Para manejar la física concreta de los vehículos.
- archivo **3DSound.cpp**
- archivo **3DSound.h**

  Clase para manejar los sonidos 3D de los objetos de la escena.
- archivo **3DSoundMng.cpp**
- archivo **3DSoundMng.h**

  Clase para gestionar los sonidos 3D de la escena.
- archivo **Game.cpp**
- archivo **Game.h**

  Para gestionar un juego completo Cada juego tiene asociado una arena y una lista de jugadores Este módulo implementa todas las operaciones para gestionar el juego.
- archivo **GUI.cpp**
- archivo **GUI.h**

  Clase para manejar el interfaz gráfico de usuario.
- archivo **Menu.cpp**
- archivo **Menu.h**

  Clase para manejar la escena 3D del menú.
- archivo **Player.cpp**
- archivo **Player.h**

  Para manejar los jugadores de la escena.
- archivo **PlayerLocal.cpp**
- archivo **PlayerLocal.h**

  Para manejar los jugadores de la escena de tipo local.
- archivo **Settings.cpp**
- archivo **Settings.h**

  Clase para manejar la configuración de la aplicación.
- archivo **XMLIO.cpp**
- archivo **XMLIO.h**

  Clase para manejar la entrada y salida de ficheros XML.

45. Documentación de clases

45.1. Referencia de la Clase C3DCamera

`#include <3DCamera.h>`

Métodos públicos

- **C3DCamera** ()
  Constructor.
- **~C3DCamera** ()
  Destructor.
- bool **initialize** ()
  Carga la cámara ante un CreateDevice.
- void **render** ()
  Renderiza la cámara ante un FrameRender.
- void **update** ()
  Actualiza la cámara ante un FrameMove.
- void **restore** ()
  Restaura la cámara ante un ResetDevice.
- void **free** ()
  Libera la cámara ante un LostDevice.
- void **Release** ()
  Libera la cámara ante un DestroyDevice.
- void **Follow** (**C3DObjVehicle** *Vehicle)
  Cambia los parámetros de la cámara para perseguir a un punto con la posición y velocidad indicadas.
- int **getType** ()
  Funcion observadora del tipo de cámara.
- int **getLastType** ()
  Funcion observadora del último tipo de cámara.
- float **getFOV** ()
  Funcion observadora de la apertura de campo de vision.
- float **getAspect** ()
  Funcion observadora de la relación de aspecto entre ancho y alto (ancho/alto).
- float **getNear** ()
  Funcion observadora del plano de corte cercano.
- float **getFar** ()
  Funcion observadora del plano de corte lejano.
- float **getZoom** ()
  Funcion observadora del grado de zoom.
- void **setType** (int Type)
  Establece el tipo de cámara.
- void **setLastType** (int Type)
  Establece el último tipo de cámara.

- void **nextType** (void)
  Toma el siguiente tipo de cámara.
- void **setFOV** (float fFOV)
  Establece la apertura de campo de vision.
- void **setAspect** (float fAspect)
  Establece la relación de aspecto entre ancho y alto (ancho/alto).
- void **setNear** (float fNear)
  Establece el plano de corte cercano.
- void **setFar** (float fFar)
  Establece el plano de corte lejano.
- void **setZoom** (float fZoom)
  Establece el grado de zoom.
- void **getPosition** (D3DXVECTOR3 &vPos)
  Funcion observadora de la posición de la cámara.
- void **setPosition** (D3DXVECTOR3 vPos)
  Establece la posición de la cámara.
- void **addPosition** (D3DXVECTOR3 vPos)
  Añade un desplazamiento lineal a la posición de la cámara.
- void **getRotation** (D3DXVECTOR3 &vRot)
  Funcion observadora de la rotación de la cámara (sólo para la cámara libre).
- void **setRotation** (D3DXVECTOR3 vRot)
  Establece la rotación de la cámara (sólo para la cámara libre).
- void **addRotation** (D3DXVECTOR3 vRot)
  Añade un desplazamiento angular a la rotación de la cámara (sólo para la cámara libre).
- void **getTo** (D3DXVECTOR3 &vTo)
  Funcion observadora de la dirección frontal de la cámara.
- void **setTo** (D3DXVECTOR3 vTo)
  Establece la dirección frontal de la cámara.
- void **getUp** (D3DXVECTOR3 &vUp)
  Funcion observadora de la dirección superior de la cámara.
- void **setUp** (D3DXVECTOR3 vUp)
  Establece la dirección superior de la cámara.

Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- IDirect3DDevice9 * **m_pDevice**
  Puntero al dispositivo de DirectX.
- D3DXMATRIX **m_ViewMatrix**
  Matriz de vista.
- D3DXMATRIX **m_ProjectionMatrix**
  Matriz de proyeccion.
- D3DXVECTOR3 **m_Position**
  Posicion de la camara.
- D3DXVECTOR3 **m_Rotation**
  Rotacion de la camara (sólo para la cámara libre).
- D3DXVECTOR3 **m_To**
  Orientación frontal de la cámara.
- D3DXVECTOR3 **m_Up**
  Orientación superior de la cámara.
- float **m_fFOV**
  Apertura de campo de vision.
- float **m_fZoom**
  Grado de zoom.
- float **m_fAspect**
  Relacion entre ancho y alto (ancho/alto).
- float **m_fNear**
  Plano de corte cercano.
- float **m_fFar**
  Plano de corte lejano.
- bool **m_bModified**
  Indica si hay que refrescar las matrices.
- bool **m_bUpdate**
  Indica si hay que recalcular su posición de manera explícita.
- int **m_iType**
  Tipo de cámara.
- int **m_iLastType**
  Tipo de cámara anterior.

45.2. Referencia de la Clase C3DCollision

`#include <3DCollision.h>`

Métodos públicos

- bool **initialize** (**C3DObject** *Object, float dmass)
  Inicializa el área de colisión.
- void **update** ()
  Comprueba las colisiones del área de colisión con el resto de la escena.
- void **Release** ()
  Libera el área de colisión.
- **C3DObject** * **getObject** (void)
  Función observadora del objeto asociado.
- bool **getIsVehicle** (void)
  Función observadora del flag que indica si es vehículo.
- void **setV** (D3DXVECTOR4 *dv, bool IsVehicle)
  Establece los cuatro vértices que determinan el área de colisión 2D (ejes X,Z) a partir de un vector dado.
- void **setV** (LPD3DXMESH pMesh, bool IsVehicle)
  Establece los cuatro vértices que determinan el área de colisión 2D (ejes X,Z) a partir de una geometría.
- D3DXVECTOR4 * **getV** (void)
  Función observadora de los cuatro vértices que determinan el área de colisión 2D (ejes X,Z).
- void **setMass** (float dmass)
  Establece la masa del objeto, si es 0.0f el objeto no es móvil.
- float **getMass** (void)
  Función observadora de la masa del objeto, si es 0.0f el objeto no es móvil.
- D3DXMATRIX * **getTransInv** (void)
  Obtiene la inversa de la transformacion actual del objeto.
- bool **CheckBasicCollision** (**C3DCollision** *Collision)
  Función para comprobar condiciones de colisión triviales (es un método directo).
- bool **CheckQuickCollision** (**C3DCollision** *Collision)
  Función para comprobar colisiones mediante esferas (es un método rápido pero aproximado).
- bool **CheckFullCollision** (**C3DCollision** *Collision, bool ReplaceElements, bool ExchangePhysics)
  Función para comprobar colisiones mediante rectángulos (es un método lento pero más detallado).

Métodos protegidos

- **C3DCollision** ()
  Constructor privado, sólo se podrá crear desde una instancia 3DObject.
- **~C3DCollision** ()
  Destructor.

Atributos protegidos

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- bool **m_bIsVehicle**
  Flag que indica si el objeto es un vehículo.
- **C3DObject** * **m_pObject**
  Objeto asociado a este área de colisión.
- float **m_mass**
  Masa del objeto, si es 0.0f el objeto no es móvil.
- D3DXVECTOR4 **m_v** [4]
  Los cuatro vértices que determinan el área de colisión 2D (ejes X,Z).
- float **m_radius**
  Radio de la circunferencia más pequeña que engloba el objeto.
- float **m_momentum**
  Momento de inercia del objeto.

45.3. Referencia de la Clase C3DLight

`#include <3DLight.h>`

Métodos públicos

- **C3DLight** ()
  Constructor.
- **~C3DLight** ()
  Destructor.
- bool **initialize** ()
  Carga la luz ante un CreateDevice.
- void **render** ()
  Renderiza la luz ante un FrameRender (no requiere implementación).
- void **update** ()
  Actualiza la luz ante un FrameMove.
- void **restore** ()
  Restaura la luz ante un ResetDevice.
- void **free** ()
  Libera la luz ante un LostDevice (no requiere implementación).
- void **Release** ()
  Libera la luz ante un DestroyDevice.
- void **FallOff** (float Spd=0.0f)
  Apagar las luces lentamente.
- void **FallIn** (float Spd=0.0f)
  Enciende las luces lentamente.
- D3DLIGHTTYPE **getType** ()
  Funcion observadora del tipo de luz.
- void **getDiffuse** (D3DCOLORVALUE &vCol)
  Funcion observadora de la componente difusa de la luz.
- void **getSpecular** (D3DCOLORVALUE &vCol)
  Funcion observadora de la componente especular de la luz.
- void **getAmbient** (D3DCOLORVALUE &vCol)
  Funcion observadora de la componente ambiental de la luz.
- void **getPosition** (D3DVECTOR &vPos)
  Funcion observadora de la posición de la luz.
- void **getDirection** (D3DVECTOR &vDir)
  Funcion observadora de la dirección de la luz.
- float **getRange** ()
  Funcion observadora del rango.
- float **getFalloff** ()
  Funcion observadora del decaimiento.
- float **getAttenuation0** ()
  Funcion observadora de la atenuación0.
- float **getAttenuation1** ()
  Funcion observadora de la atenuación1.

- float **getAttenuation2** ()
  Funcion observadora de la atenuación2.
- float **getTheta** ()
  Funcion observadora de Theta.
- float **getPhi** ()
  Funcion observadora de Phi.
- void **setType** (D3DLIGHTTYPE v)
  Establece el tipo de luz.
- void **setDiffuse** (float r, float g, float b, float a)
  Establece la componente difusa de la luz.
- void **setSpecular** (float r, float g, float b, float a)
  Establece la componente especular de la luz.
- void **setAmbient** (float r, float g, float b, float a)
  Establece la componente ambiental de la luz.
- void **setPosition** (D3DVECTOR vPos)
  Establece la posición.
- void **setDirection** (D3DVECTOR vDir)
  Establece la dirección.
- void **setRange** (float v)
  Establece el rango.
- void **setFalloff** (float v)
  Establece el decaimiento de la luz.
- void **setAttenuation0** (float v)
  Establece la ateniación0.
- void **setAttenuation1** (float v)
  Establece la ateniación1.
- void **setAttenuation2** (float v)
  Establece la ateniación2.
- void **setTheta** (float v)
  Establece Theta.
- void **setPhi** (float v)
  Establece Phi.

Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- IDirect3DDevice9 * **m_pDevice**
  Puntero al dispositivo de DirectX.
- D3DLIGHT9 **m_Light**
  Luz.
- bool **m_bModified**
  Indica si hay que refrescar la luz.
- bool **m_bFallOff**
  Indica si se están apagando las luces lentamente.
- bool **m_bFallIn**
  Indica si se están encendiendo las luces lentamente.
- float **m_FallSpd**
  Velocidad del FallOff o FallIn actual.
- D3DCOLORVALUE **m_Diffuse_ori**
  Luz difusa original.
- D3DCOLORVALUE **m_Specular_ori**
  Luz especular original.
- D3DCOLORVALUE **m_Ambient_ori**
  Luz ambiental original.

45.4. Referencia de la Clase C3DMesh

`#include <3DMesh.h>`

Métodos públicos

- bool **initialize** (**C3DObject** *dObject, LPCWSTR MeshPath, int dAlpha)
  Inicializa el modelo gráfico.
- void **render** (bool force=false)
  Renderiza el modelo gráfico.
- void **Release** ()
  Libera el modelo gráfico.
- **C3DObject** * **getObject** (void)
  Función observadora del objeto asociado.
- LPD3DXMESH **getMesh** (void)
  Función observadora de la geometría del objeto.

Métodos protegidos

- **C3DMesh** ()
  Constructor privado, sólo se podrá crear desde una instancia 3DObject.
- **~C3DMesh** ()
  Destructor.

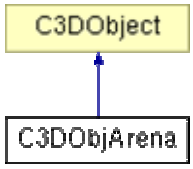
Atributos protegidos

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- IDirect3DDevice9 * **m_pDevice**
  Puntero al dispositivo de DirectX.
- **C3DObject** * **m_pObject**
  Objeto asociado a este modelo gráfico.
- int **m_iAlpha**
  Tipo de transparencia de la geometría.
- LPWSTR **m_awMeshPath**
  Path dónde se encuentra el modelo gráfico.
- LPD3DXMESH **m_pMesh**
  Geometria.
- std::vector< D3DMATERIAL9 > * **m_aMtrls**
  Lista de Materiales.
- std::vector< IDirect3DTexture9 * > * **m_aTextures**
  Lista de Texturas.

45.5. Referencia de la Clase C3DObjArena

`#include <3DObjArena.h>`

Diagrama de herencias de C3DObjArena



Métodos públicos

- **C3DObjArena** ()
  Constructor.
- **~C3DObjArena** ()
  Destructor.
- bool **initialize** (**C3DObject** *dParent, LPCWSTR ObjectPath)
  Inicializa la arena.
- void **update** ()
  Actualiza la arena.
- void **render** ()
  Renderiza la arena.
- bool **loadRanking** (void)
  Carga el fichero de ranking.
- bool **saveRanking** (void)
  Guarda el fichero de ranking.
- void **addRanking** (**Rank** &R)
  Añade un elemento al ranking.
- LPWSTR **getName** (void)
  Función observadora del nombre de la arena.
- LPWSTR **getDescription** (void)
  Función observadora de la descripción de la arena.
- LPWSTR **getRankTxt** (void)
  Función observadora del texto de ranking.
- **C3DObject** * **getSky** (void)
  Función observadora del cielo de la arena.
- void **getPosObjective** (D3DXVECTOR3 &Vec)
  Toma la siguiente posición para el objetivo.
- void **getPosElement** (D3DXVECTOR3 &Vec)
  Toma la siguiente posición para elementos.
- void **nextObjective** (void)
  Situa el siguiente objetivo.
- void **nextElement** (void)
  Situa el siguiente elemento.
- **C3DObjElement** * **getObjetive** (void)
  Función observadora del elemento objetivo de la arena.

- **C3DObjElement * getRand** (void)
  Función observadora del elemento interrogación de la arena.
- **C3DObjElement * getHealth** (void)
  Función observadora del elemento de vida de la arena.
- **C3DObjElement * getRocket** (void)
  Función observadora del elemento cohete de la arena.
- **C3DObjElement * getMine** (void)
  Función observadora del elemento mina de la arena.

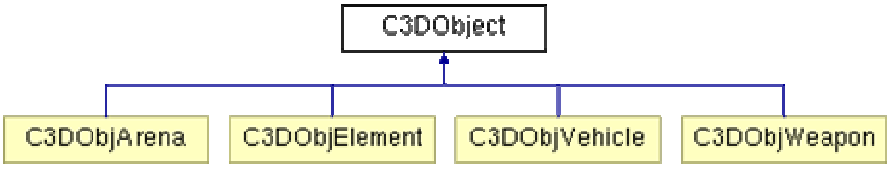
Atributos privados

- LPWSTR **m_awName**
  Nombre de la arena.
- LPWSTR **m_awDescription**
  Descripción textual de la arena.
- LPWSTR **m_awRankFile**
  Ruta del fichero de rankings.
- LPWSTR **m_awRankTxt**
  Texto del ranking.
- **C3DObject * m_pSky**
  Puntero al cielo de la arena.
- **C3DObjElement * m_pObjetive**
  Puntero al elemento objetivo.
- **C3DObjElement * m_pRand**
  Puntero al elemento interrogación.
- **C3DObjElement * m_pHealth**
  Puntero al elemento de vida.
- **C3DObjElement * m_pRocket**
  Puntero al elemento cohete.
- **C3DObjElement * m_pMine**
  Puntero al elemento mina.
- std::vector< D3DXVECTOR3 > **m_pPositions**
  Lista de posibles ubicaciones para los elementos.
- int **m_pPosObjective**
  Última posición del objetivo.
- int **m_pPosElement**
  Última posición del elemento.
- float **m_pNumObj**
  Número de objetivos a capturar.
- float **m_pTimeIni**
  Tiempo inicial para la arena.
- float **m_pTimeObj**
  Tiempo por objetivo capturado.
- float **m_pSize**
  Tamaño de la arena.
- std::list< **Rank** > **m_pRanking**
  Ranking de mejores tiempos de la arena.

45.6. Referencia de la Clase C3DObject

`#include <3DObject.h>`

Diagrama de herencias de C3DObject



Métodos públicos

- **C3DObject** ()
  Constructor.
- **~C3DObject** ()
  Destructor.
- bool **initialize** (**C3DObject** *dParent, LPCWSTR ObjectPath)
  Inicializa el objeto a partir de la ruta del fichero XML (colisiones + física + sonido + modelo gráfico + resto de objetos).
- bool **initialize** (**C3DObject** *dParent, **CXMLIO** *myXMLIO)
  Inicializa el objeto a partir de un lector de XML (colisiones + física + sonido + modelo gráfico + resto de objetos).
- void **render** ()
  Renderiza el objeto (modelo gráfico + resto de objetos).
- void **update** ()
  Actualiza el objeto (colisiones + física + sonido + resto de objetos).
- void **Release** ()
  Libera el objeto (colisiones + física + sonido + modelo gráfico + resto de objetos).
- **C3DMesh** * **getMesh** (void)
  Función observadora del modelo gráfico asociado.
- **C3DCollision** * **getCollision** (void)
  Función observadora de la colisión asociada.
- **C3DPhysics** * **getPhysics** (void)
  Función observadora de la física asociada.
- bool **hasPhysics** (void)
  Función que indica si el objeto tiene o no física asociada.
- **C3DSound** * **getSound** (void)
  Función observadora del sonido asociado.
- std::vector< **C3DObject** * > * **getObjects** (void)
  Función observadora de la lista de objetos que componen el objeto.
- int **getTypeObject** (void)
  Función observadora del tipo de objeto (móvil, fijo con física, fijo sin física, no colisionable).
- D3DXMATRIX * **getTransformation** (void)
  Obtiene la transformacion actual del objeto.

- D3DXMATRIX * **calculateTransInv** (D3DXMATRIX *result)
  Toma o calcula la inversa de la transformacion actual del objeto.
- **C3DParticles * getParticles1** (void)
  Función para obtener el sistema de partículas primario.
- **C3DParticles * getParticles2** (void)
  Función para obtener el sistema de partículas secundario.
- void **setEnabled** (bool enabled)
  Función para cambiar el estado de procesamiento de un objeto, activado/desactivado.
- bool **getEnabled** (void)
  Función tomar el estado de procesamiento de un objeto, activado/desactivado.
- bool **getModified** (void)
  Función tomar el flag de modificación del objeto tras la última iteración.
- bool **getLastModified** (void)
  Función tomar el flag de modificación del objeto en esta última iteración.
- void **setScale** (D3DXVECTOR3 &Vec)
  Establece el escalado del objeto.
- void **getScale** (D3DXVECTOR3 &Vec)
  Toma el escalado actual del objeto.
- void **resetScale** (void)
  Elimina el escalado actual del objeto.
- void **setRotation** (D3DXVECTOR3 &Vec)
  Establece la rotación del objeto.
- void **getRotation** (D3DXVECTOR3 &Vec)
  Toma la rotación actual del objeto.
- void **resetRotation** (void)
  Elimina la rotación actual del objeto.
- void **setPosition** (D3DXVECTOR3 &Vec)
  Establece la posición del objeto.
- void **getPosition** (D3DXVECTOR3 &Vec)
  Toma la posición actual del objeto.
- void **resetPosition** (void)
  Elimina la posición actual del objeto.
- void **addScale** (D3DXVECTOR3 &Vec)
  Añade un escalado al objeto.
- void **addPrePosition** (D3DXVECTOR3 &Vec)
  Añade una posición al objeto para aplicar antes de la rotación.
- void **addRotation** (D3DXVECTOR3 &Vec)
  Añade una rotación al objeto.
- void **addPosition** (D3DXVECTOR3 &Vec)
  Añade una posición al objeto.
- virtual bool **onQuickCollision** (**C3DObject** *object)
  Manejador de colisión básica.
- virtual bool **onFullCollision** (**C3DObject** *object)
  Manejador de colisión completa.

Métodos protegidos

- bool **load** (LPCWSTR ObjectPath)
  Carga la configuración a partir de la ruta del fichero XML (colisiones + física + modelo gráfico + resto de objetos).
- bool **load** (**CXMLIO** *myXMLIO)
  Carga la configuración a partir de un lector de XML (colisiones + física + modelo gráfico + resto de objetos).
- void **setModified** (void)
  Para indicar al objeto que debe de reactualizar las matrices.
- void **refreshTransformation** (void)
  Refresca la transformación actual del objeto, sólo si es necesario.

Atributos protegidos

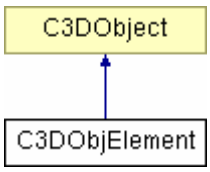
- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- bool **m_bEnabled**
  Estado de procesamiento de un objeto.
- **C3DObject * m_pParent**
  Puntero al objeto padre (NULL si no hay padre).
- **C3DMesh * m_pMesh**
  Modelo gráfico asociado al objeto.
- **C3DCollision * m_pCollision**
  Sistema de colisiones asociado al objeto.
- **C3DPhysics * m_pPhysics**
  Sistema físico asociado al objeto.
- **C3DSound * m_pSound**
  Sonido asociado al objeto.
- std::vector< **C3DObject** * > * **m_paObjets**
  Lista de objetos que componen el resto del objeto.
- D3DXMATRIX **m_Transformation**
  Matriz de transformacion final neta.
- D3DXVECTOR3 * **m_pScale**
  Vector del escalado.
- D3DXVECTOR3 * **m_pRotation**
  Vector de la rotacion.
- D3DXVECTOR3 * **m_pPosition**
  Vector de la posicion.
- bool **m_bModified**
  Indica si el objeto fue modificado tras la última iteración.
- bool **m_bLastModified**
  Indica si el objeto fue modificado en esta última iteración.

- int **m_iTypeObject**
  Tipo de objeto (móvil, fijo con física, fijo sin física, no colisionable).
- **C3DParticles * m_pParticles1**
  Sistema de partículas primario.
- D3DXVECTOR3 * **m_pParticles1Pos**
  Vector de posicion relativa del sistema de partículas primario.
- **C3DParticles * m_pParticles2**
  Sistema de partículas secundario.
- D3DXVECTOR3 * **m_pParticles2Pos**
  Vector de posicion relativa del sistema de partículas secundario.

45.7. Referencia de la Clase C3DObjElement

`#include <3DObjElement.h>`

Diagrama de herencias de C3DObjElement



Métodos públicos

- **C3DObjElement** ()
  Constructor.
- **~C3DObjElement** ()
  Destructor.
- bool **initialize** (**C3DObjArena** *Arena, LPCWSTR ObjectPath, int Type)
  Inicializa el elemento de la arena.
- void **update** ()
  Actualiza el elemento de la arena.
- void **setEnabled** (bool enabled)
  Función para cambiar el estado de procesamiento del elemento, activado/desactivado.
- int **getType** (void)
  Función observadora del tipo de elemento.
- int **getState** (void)
  Función observadora del estado del elemento.
- void **setState** (int dState)
  Función para cambiar el estado del elemento.
- void **playAppear** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de aparición del elemento.
- bool **isPlayAppear** (bool)
  Para saber si está activo el sonido de aparición del elemento.
- void **stopAppear** (void)
  Para desactivar el sonido de aparición del elemento.
- void **playGet** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de captura del elemento.
- bool **isPlayGet** (bool)
  Para saber si está activo el sonido de captura del elemento.
- void **stopGet** (void)
  Para desactivar el sonido de captura del elemento.
- bool **onQuickCollision** (**C3DObject** *object)
  Manejador de colisión básica.

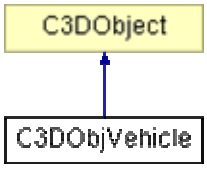
Atributos privados

- **C3DObjArena * m_pArena**
  Arena asociada a este elemento.
- int **m_pType**
  Tipo de elemento de la arena.
- int **m_pState**
  Indica si el elemento está libre, lanzado o colisionado.
- float **m_pDelay**
  Tiempo desde que colisionó el elemento.
- **C3DSound * m_pSndAppear**
  Sonido del aparición.
- **C3DSound * m_pSndGet**
  Sonido del captura.

45.8. Referencia de la Clase C3DObjVehicle

`#include <3DObjVehicle.h>`

Diagrama de herencias de C3DObjVehicle



Métodos públicos

- **C3DObjVehicle** ()
  Constructor.
- **~C3DObjVehicle** ()
  Destructor.
- bool **initialize** (**C3DObject** *dParent, LPCWSTR ObjectPath, **CPlayer** *dPlayer)
  Inicializa el vehículo.
- void **update** ()
  Actualiza el vehículo.
- void **render** ()
  Renderiza el vehículo.
- D3DXMATRIX * **calculateTransInv** (D3DXMATRIX *result)
  Toma o calcula la inversa de la transformacion actual del vehículo.
- **CPlayer** * **getPlayer** (void)
  Función observadora del jugador asociado con el vehículo.
- LPWSTR **getName** (void)
  Función observadora del nombre del vehículo.
- LPWSTR **getDescription** (void)
  Función observadora de la descripción del vehículo.
- float **getMaxCapacity** (void)
  Función observadora de la capacidad máxima del vehículo.
- float **getMaxHealth** (void)
  Función observadora de la vida máxima del vehículo.
- float **getMaxSpeed** (void)
  Función observadora de la velocidad máxima del vehículo.
- void **setMaxSpeed** (float val)
  Función para indicar la velocidad máxima del vehículo.
- float **getMaxAcceleration** (void)
  Función observadora de la aceleración máxima del vehículo.
- float **getMaxTurnBodyZ** (void)
  Función observadora de la máxima rotación para la suspensión frontal.
- float **getMaxTurnBodyX** (void)
  Función observadora de la máxima rotación para la suspensión lateral.
- float **getTurnBodyZ** (void)
  Función observadora de la dureza para la suspensión frontal.

- float **getTurnBodyX** (void)
  Función observadora de la dureza para la suspensión frontal.
- float **getRecvBodyZ** (void)
  Función observadora de la recuperación para la suspensión frontal.
- float **getRecvBodyX** (void)
  Función observadora de la recuperación para la suspensión frontal.
- float **getTurn** (void)
  Función observadora del giro del vehículo.
- float **getMaxTurnWheelY** (void)
  Función observadora del máximo de las ruedas en Y.
- float **getTurnWheelZ** (void)
  Función observadora del máximo de las ruedas en Z.
- float **getWheelDist** (void)
  Función observadora de la longitud del centro del vehículo al tren trasero o delantero.
- void **setWheelDist** (float val)
  Para cambiar la longitud del centro del vehículo al tren trasero o delantero.
- float **getCameraY** (void)
  Función observadora de la altura de la cámara en el modo de vista interior.
- float **getLastSpeed** (void)
  Función observadora de la última velocidad registrada.
- **C3DObject * getBody** (void)
  Función observadora del fuselaje del vehículo.
- **C3DObject * getFRWheel** (void)
  Función observadora de la 'Front Right Wheel' del vehículo.
- **C3DObject * getFLWheel** (void)
  Función observadora de la 'Front Left Wheel' del vehículo.
- **C3DObject * getRRWheel** (void)
  Función observadora de la 'Rear Right Wheel' del vehículo.
- **C3DObject * getRLWheel** (void)
  Función observadora de la 'Rear Left Wheel' del vehículo.
- **C3DObjWeapon * getRocket** (void)
  Función observadora del cohete asociado del vehículo.
- **C3DObjWeapon * getMine** (void)
  Función observadora de la mina asociada del vehículo.
- bool **getHandBrake** (void)
  Función observadora del estado del freno de mano.
- void **setHandBrake** (bool Value)
  Función para activar o desactivar el freno de mano.
- bool **isSkidding** (void)
  Función que determina si el vehículo está o no derrapando.
- float **getAcumTurn** (void)
  Función observadora del ángulo de giro acumulado.
- void **setAcumTurn** (float Value)
  Función para fijar un ángulo de giro acumulado determinado.

- bool **getAccelerate** (void)
  Función observadora del estado del acelerador.
- void **setAccelerate** (bool Value)
  Función para activar o desactivar el acelerador.
- void **playSkid** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de derrape del coche.
- bool **isPlaySkid** (bool)
  Para saber si está activo el sonido de derrape del coche.
- void **stopSkid** (void)
  Para desactivar el sonido de derrape del coche.
- void **playCrash** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de colisiones del coche.
- bool **isPlayCrash** (bool)
  Para saber si está activo el sonido de colisiones del coche.
- void **stopCrash** (void)
  Para desactivar el sonido de colisiones del coche.

Métodos privados

- float **updateLastSpeed** (void)
  Función para actualizar la última velocidad del vehículo, limitándola a la máxima, retorna la diferencia respecto la anterior.
- float **updateLastYRotation** (void)
  Función para actualizar la última rotación del vehículo, retorna la diferencia respecto la anterior.
- void **updateBody** (float DiffSpeed, float DiffRotation)
  Función para actualizar la suspensión del fuselaje del vehículo (frontal y transversal).
- void **updateWheels** (void)
  Función para actualizar el rodaje de las ruedas del vehículo.
- void **updateDirection** (float DiffRotation)
  Función para actualizar la dirección de la velocidad.
- void **updateSound** (float DiffSpeed)
  Función para actualizar el sonido del vehículo.
- void **refreshTransformation** (float DiffRotation)
  Refresca la transformación actual del objeto, sólo si es necesario.

Atributos privados

- **CPlayer * m_pPlayer**
  Puntero al jugador asociado con el vehículo.
- LPWSTR **m_awName**
  Nombre del vehículo.
- LPWSTR **m_awDescription**
  Descripción textual del vehículo.
- float **m_iMaxCapacity**
  Capacidad máxima del vehículo.
- float **m_iMaxHealth**
  Vida máxima del vehículo.
- float **m_fMaxSpeed**
  Velocidad máxima del vehículo.
- float **m_fMaxAcceleration**
  Aceleración máxima del vehículo.
- bool **m_bUpdate**
  Flag que indica si el objeto está actualizado (para forzar el primer update).
- **C3DPhyVehicle * m_pPhyVehicle**
  Sistema físico asociado al vehículo.
- float **m_fMaxTurnBodyZ**
  Máxima rotación para la suspensión frontal.
- float **m_fMaxTurnBodyX**
  Máxima rotación para la suspensión lateral.
- float **m_fTurnBodyZ**
  Dureza para la suspensión frontal.
- float **m_fTurnBodyX**
  Dureza para la suspensión lateral.
- float **m_fRecvBodyZ**
  Recuperación para la suspensión frontal.
- float **m_fRecvBodyX**
  Recuperación para la suspensión frontal.
- float **m_fTurn**
  Giro del vehículo.
- float **m_fMaxTurnWheelY**
  Giro máximo de las ruedas en Y.
- float **m_fTurnWheelZ**
  Giro máximo de las ruedas en Z.
- float **m_fWheelDist**
  Longitud del centro del vehículo al tren trasero o delantero.
- float **m_fCameraY**
  Altura de la cámara en el modo de vista interior.
- **C3DObject * m_pBody**
  Puntero al fuselaje del vehículo.
- **C3DObject * m_pFRWheel**
  Puntero a la 'Front Right Wheel' del vehículo.

- **C3DObject * m_pFLWheel**
  Puntero a la 'Front Left Wheel' del vehículo.
- **C3DObject * m_pRRWheel**
  Puntero a la 'Rear Right Wheel' del vehículo.
- **C3DObject * m_pRLWheel**
  Puntero a la 'Rear Left Wheel' del vehículo.
- **C3DObjWeapon * m_pRocket**
  Puntero al cohete del vehículo.
- **C3DObjWeapon * m_pMine**
  Puntero a la mina del vehículo.
- float **m_fLastSpeed**
  Última velocidad instantánea registrada en el vehículo.
- float **m_fLastYRotation**
  Última posición de rotación en Y del vehículo.
- bool **m_bHandBrake**
  Flag que indica si está activado el freno de mano.
- bool **m_bAccelerate**
  Flag que indica si está activado el acelerador.
- float **m_fAcumTurn**
  Giro acumulado del vehículo (debido a derrapes).
- **C3DSound * m_pSndSkid**
  Sonido del derrape.
- **C3DSound * m_pSndCrash**
  Sonido del colisión.

45.9. Referencia de la Clase C3DObjWeapon

`#include <3DObjWeapon.h>`

Diagrama de herencias de C3DObjWeapon

Métodos públicos

- **C3DObjWeapon** ()
  Constructor.
- **~C3DObjWeapon** ()
  Destructor.
- bool **initialize** (**C3DObjVehicle** *Vehicle, LPCWSTR ObjectPath, int Type)
  Inicializa el arma del vehículo.
- void **update** ()
  Actualiza el arma del vehículo.
- void **setEnabled** (bool enabled)
  Función para cambiar el estado de procesamiento del arma, activado/desactivado.
- int **getType** (void)
  Función observadora del tipo de arma.
- int **getState** (void)
  Función observadora del estado del arma.
- void **playLaunch** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de lanzamiento del arma.
- bool **isPlayLaunch** (bool)
  Para saber si está activo el sonido de lanzamiento del arma.
- void **stopLaunch** (void)
  Para desactivar el sonido de lanzamiento del arma.
- void **playCrash** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de colisiones del arma.
- bool **isPlayCrash** (bool)
  Para saber si está activo el sonido de colisiones del arma.
- void **stopCrash** (void)
  Para desactivar el sonido de colisiones del arma.
- void **playEmpty** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de vacío del arma.
- bool **isPlayEmpty** (bool)
  Para saber si está activo el sonido de vacío del arma.
- void **stopEmpty** (void)
  Para desactivar el sonido de vacío del arma.
- bool **onQuickCollision** (**C3DObject** *object)
  Manejador de colisión básica.
- bool **onFullCollision** (**C3DObject** *object)
  Manejador de colisión completa.

Atributos privados

- **C3DObjVehicle * m_pVehicle**
  Vehículo asociado a este arma.
- int **m_pType**
  Tipo de arma del vehículo.
- int **m_pState**
  Indica si el arma está libre, lanzado o colisionado.
- float **m_pDelay**
  Tiempo desde el lanzamiento o desde la colisión.
- **C3DSound * m_pSndLaunch**
  Sonido del lanzamiento.
- **C3DSound * m_pSndCrash**
  Sonido de la colisión.
- **C3DSound * m_pSndEmpty**
  Sonido de vacío.

45.10. Referencia de la Clase C3DParticles

`#include <3DParticle.h>`

Métodos públicos

- **C3DParticles** (void)
  Constructor.
- **~C3DParticles** (void)
  Destructor.
- HRESULT **initialize** ()
  Inicializa el sistema de partículas.
- HRESULT **update** ()
  Actualiza el sistema de partículas.
- HRESULT **render** ()
  Renderiza el sistema de partículas.
- HRESULT **restore** ()
  Restaura el sistema de partículas.
- HRESULT **free** ()
  Libera el sistema de partículas.
- void **Release** ()
  Destruye el sistema de partículas.

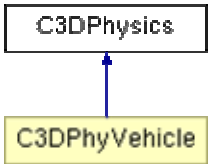
Atributos privados

- IDirect3DDevice9 * **m_pDevice**
  Puntero al dispositivo de DirectX.
- LPDIRECT3DVERTEXBUFFER9 **m_pVB**
  Buffer de vértices.
- LPDIRECT3DTEXTURE9 **m_ptexParticle**
  Textura de las partículas.

45.11. Referencia de la Clase C3DPhysics

`#include <3DPhysics.h>`

Diagrama de herencias de C3DPhysics



Métodos públicos

- bool **initialize** (**C3DObject** *Object)
  Inicializa el sistema físico.
- void **update** ()
  Aplica los parámetros físicos.
- void **Release** ()
  Libera los parámetros físicos.
- **C3DObject** * **getObject** (void)
  Función observadora del objeto asociado.
- void **setLSpeed** (float Val)
  Establece la velocidad lineal actual.
- float **getLSpeed** (void)
  Toma la velocidad lineal actual.
- void **addLSpeed** (float Val)
  Añade una velocidad lineal a la existente.
- D3DXVECTOR3 * **getLDirection** (void)
  Toma el vector normalizado de dirección lineal.
- void **setLDirection** (D3DXVECTOR3 &Vec)
  Establece el vector normalizado de dirección lineal y la magnitud de velocidad lineal.
- void **setLAcceleration** (float Val)
  Establece la aceleración lineal actual.
- float **getLAcceleration** (void)
  Toma la aceleración lineal actual.
- void **addLAcceleration** (float Val)
  Añade una aceleración lineal a la existente.
- void **setLFriction** (float Val)
  Establece la fricción lineal actual.
- float **getLFriction** (void)
  Toma la fricción lineal actual.
- void **addRotLDirection** (D3DXVECTOR3 &Vec)
  Añade una rotación a la dirección de la velocidad lineal actual.
- void **setRotLDirection** (D3DXVECTOR3 &Vec)
  Establece una rotación a la dirección de la velocidad lineal actual.
- void **resetRotLDirection** (void)
  Resetea la rotación de la dirección de la velocidad lineal actual.

- void **setASpeed** (D3DXVECTOR3 &Vec)
  Establece la velocidad angular actual.
- void **getASpeed** (D3DXVECTOR3 &Vec)
  Toma la velocidad angular actual.
- void **addASpeed** (D3DXVECTOR3 &Vec)
  Añade una velocidad angular a la existente.
- void **setAAcceleration** (D3DXVECTOR3 &Vec)
  Establece la aceleración angular actual.
- void **getAAcceleration** (D3DXVECTOR3 &Vec)
  Toma la aceleración angular actual.
- void **addAAcceleration** (D3DXVECTOR3 &Vec)
  Añade una aceleración angular a la existente.
- void **setAFriction** (D3DXVECTOR3 &Vec)
  Establece la fricción angular actual.
- void **getAFriction** (D3DXVECTOR3 &Vec)
  Toma la fricción angular actual.
- void **addAFriction** (D3DXVECTOR3 &Vec)
  Añade una fricción angular a la existente.

Métodos protegidos

- **C3DPhysics** ()
  Constructor privado, sólo se podrá crear desde una instancia 3DObject.
- **~C3DPhysics** ()
  Destructor.

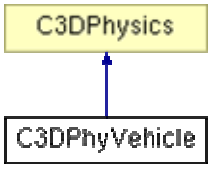
Atributos protegidos

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- float **m_fLSpeed**
  Módulo de la velocidad lineal.
- float **m_fLAcceleration**
  Módulo de la aceleración lineal.
- float **m_fLFriction**
  Módulo del rozamiento lineal.
- D3DXVECTOR3 **m_pLDirection**
  Vector normalizado de la dirección de desplazamiento lineal.
- D3DXVECTOR3 * **m_pASpeed**
  Velocidad angular en cada uno de los tres ejes.
- D3DXVECTOR3 * **m_pAAcceleration**
  Aceleración angular en cada uno de los tres ejes.
- D3DXVECTOR3 * **m_pAFriction**
  Rozamiento angular en cada uno de los tres ejes.
- **C3DObject * m_pObject**
  Objeto asociado a esta física.

45.12. Referencia de la Clase C3DPhyVehicle

`#include <3DPhyVehicle.h>`

Diagrama de herencias de C3DPhyVehicle



Métodos públicos

- bool **initialize** (**C3DObjVehicle** *Vehicle)
  Inicializa el sistema físico.
- void **update** ()
  Aplica los parámetros físicos.
- **C3DObjVehicle** * **getVehicle** (void)
  Función observadora del vehículo asociado.
- float **getAditionalLSpeed** (void)
  Toma la magnitud de desplazamientos lineales adicionales.
- void **setAditionalLSpeed** (float Val)
  Establece la magnitud de desplazamientos lineales adicionales.
- void **addAditionalLSpeed** (float Val)
  Añade a la magnitud de desplazamientos lineales adicionales.
- D3DXVECTOR3 * **getAditionalLDirection** (void)
  Toma el vector normalizado de desplazamientos lineales adicionales.
- void **setAditionalL** (D3DXVECTOR3 &Vec)
  Establece desplazamientos lineales adicionales para el vehículo.

Métodos protegidos

- **C3DPhyVehicle** ()
  Constructor privado, sólo se podrá crear desde una instancia 3DObject.
- **~C3DPhyVehicle** ()
  Destructor.

Atributos protegidos

- **C3DObjVehicle** * **m_pVehicle**
  Vehículo asociado a esta física.
- float **m_pAditionalLSpeed**
  Magnitud de desplazamientos lineales adicionales.
- D3DXVECTOR3 **m_pAditionalLDirection**
  Dirección de desplazamientos lineales adicionales.

45.13. Referencia de la Clase C3DSound

`#include <3DSound.h>`

Métodos públicos

- void **update** ()
  Actualiza el sonido.
- void **Play** (bool loop=false)
  Reproduce el sonido, se puede indicar una reproducción en bucle cerrado.
- void **Pause** (bool enabled)
  Pausa y Despausa la reproducción del sonido.
- void **Stop** (void)
  Detiene la reproducción del sonido.
- void **setVolume** (long lVolume)
  Cambia el volumen de reproducción del sonido.
- void **setFrecuency** (long lFrequency)
  Cambia la frecuencia de reproducción del sonido.
- long **getFrecuency** (void)
  Toma la frecuencia actual de reproducción del sonido.
- long **getInitFrecuency** (void)
  Toma la frecuencia inicial de reproducción del sonido.
- bool **isPlaying** (void)
  Retorna si está o no en ejecución el sonido.
- **C3DObject** * **getObject** (void)
  Función observadora del objeto asociado.

Métodos protegidos

- **C3DSound** (**C3DObject** *pOwner, CSound *pSound, LPDIRECTSOUND3DBUFFER pDS3DBuffer, long lFrequency, long lVolume)
  Constructor.
- **~C3DSound** ()
  Destructor.
- void **Release** ()
  Libera los sonidos.

Atributos protegidos

- CSound * **m_pSound**
  Instancia del sonido a reproducir.
- LPDIRECTSOUND3DBUFFER **m_pDS3DBuffer**
  Buffer 3D para el sonido.
- DWORD **m_lInitFrequency**
  Frecuencia base para el sonido emitido.
- DWORD **m_lLastFrequency**
  Frecuencia actual para el sonido emitido.
- bool **m_bLoop**
  Looping o no en la reproducción del sonido.
- **C3DObject** * **m_pObject**
  Objeto asociado a este sonido.

Amigas

- class **C3DSoundMng**

45.14. Referencia de la Clase C3DSoundMng

`#include <3DSoundMng.h>`

Métodos públicos

- **C3DSoundMng** ()
  Constructor.
- **~C3DSoundMng** ()
  Destructor.
- bool **initialize** (void)
  Carga el manejador de sonidos.
- **C3DSound** * **Create** (LPWSTR File, **C3DObject** *Owner)
  Crea un nuevo sonido 3D.
- bool **Destroy** (**C3DSound** *Sound)
  Destruye un sonido 3D existente.
- void **Pause** (bool enabled)
  Pausa y Despausa la reproducción de todos los sonidos que maneja.
- void **setVolume** (long lVolume)
  Cambia el volumen de reproducción de todos los sonidos que maneja.
- void **update** (void)
  Actualiza los cambios de los sonidos 3D.
- void **Release** ()
  Libera todos los sonidos que maneja.

Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- CSoundManager * **m_pSoundManager**
  Manejador principal de sonidos 3D.
- LPDIRECTSOUND3DLISTENER **m_pDSListener**
  Entorno 3D para el sonido.
- long **m_lVolume**
  Volumen actual de los sonidos.
- std::vector< **C3DSound** * > **m_aSounds**
  Lista de sonidos 3D manejados.

45.15. Referencia de la Clase CGame

`#include <Game.h>`

Métodos públicos

- **CGame** ()
  Constructor.
- **~CGame** ()
  Destructor.
- bool **initialize** (**SDataGame** *dData)
  Inicializa el juego.
- void **render** ()
  Renderiza los objetos del juego.
- void **update** ()
  Actualiza el juego.
- void **Release** ()
  Libera los datos del juego.
- **CPlayerLocal** * **getLocalPlayer1** (void)
  Función observadora del jugador 1.
- **CPlayerLocal** * **getLocalPlayer2** (void)
  Función observadora del jugador 2.
- **C3DObjArena** * **getArena** (void)
  Función observadora de la arena del juego.
- int **getType** (void)
  Función observadora del tipo de juego.
- void **playWin** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de victoria.
- bool **isPlayWin** (bool)
  Para saber si está activo el sonido de victoria.
- void **stopWin** (void)
  Para desactivar el sonido de victoria.
- void **playLose** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de derrota.
- bool **isPlayLose** (bool)
  Para saber si está activo el sonido de derrota.
- void **stopLose** (void)
  Para desactivar el sonido de derrota.
- void **playTime** (bool loop, long volume=0, long frecuency=0)
  Para activar el sonido de tiempo.
- bool **isPlayTime** (bool)
  Para saber si está activo el sonido de tiempo.
- void **stopTime** (void)
  Para desactivar el sonido de tiempo.

Métodos privados

- void **updateInfo** (int num_player)
  Para actualizar varios marcadores del GUI durante el juego.
- bool **updateTime** (void)
  Para actualizar el tiempo durante el juego.

Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- int **m_iType**
  Tipo de juego.
- **C3DObjArena * m_pArena**
  Puntero a la arena del juego.
- **CPlayerLocal * m_pLPlayer1**
  Puntero al jugador 1.
- **CPlayerLocal * m_pLPlayer2**
  Puntero al jugador 2.
- float **m_pTime**
  Tiempo disponible en el momento actual.
- float **m_pLastTime**
  Tiempo disponible en el momento anterior.
- bool **m_bChangeTime**
  Flag que indica si se cambió el marcador de tiempo.
- **C3DSound * m_pSndWin**
  Sonido de victoria.
- **C3DSound * m_pSndLose**
  Sonido de derrota.
- **C3DSound * m_pSndTime**
  Sonido de tiempo.

45.16. Referencia de la Clase CGUI

`#include <GUI.h>`

Métodos públicos

- **CGUI** ()
  Constructor.
- **~CGUI** ()
  Destructor.
- bool **initialize** ()
  Carga el GUI ante un CreateDevice.
- void **render** ()
  Renderiza el GUI ante un FrameRender.
- void **update** ()
  Actualiza el GUI ante un FrameMove.
- void **restore** ()
  Restaura objetos ante un ResetDevice.
- void **free** ()
  Libera objetos ante un LostDevice.
- void **Release** ()
  Libera el GUI ante un DestroyDevice.
- void **iniGameConfig** (void)
  Inicializa algunas variables y elementos de la configuración del juego.
- void **setMarcador** (int num, int marcador)
  Cambia el marcador indicado con el número indicado.
- void **setText** (LPCWSTR txt, int Type)
  Cambia el texto de diferentes etiquetas personalizables.
- void **throwEvent** (int ID)
  Para lanzar un evento.
- int * **getIndex** (int id)
  Vector que retorna un puntero a un entero con el valor indicado, se usa para gestionar los elementos de los comboboxes del GUI.
- bool **setArena** (int dIndex)
  Establece la arena con el índice dado, cargando todas antes si fuese necesario.
- bool **setVehicle** (int numPlayer, int dIndex)
  Establece el vehículo con el índice dado, cargando todos antes si fuese necesario.
- bool **nextArena** (void)
  Establece la siguiente arena de la lista.
- bool **priorArena** (void)
  Establece la anterior arena de la lista.
- bool **nextVehicle** (void)
  Establece el siguiente vehículo de la lista.
- bool **priorVehicle** (void)
  Establece el anterior vehículo de la lista.
- void **setTypeGUI** (int dType)
  Establece el tipo de GUI.

- int **getTypeGUI** (void)
  Funcion observadora del tipo de GUI actual.
- int **getOldTypeGUI** (void)
  Funcion observadora del tipo de GUI anterior.
- void **setStadistics** (bool dType)
  Muestra u oculta las estadísticas.
- bool **getStadistics** (void)
  Comprueba si están o no las estadísticas activadas.
- void **setTypeGame** (int dType)
  Función para establecer el tipo de juego seleccionado.
- int **getTypeGame** (void)
  Funcion observadora del tipo de juego.
- int **getTypeArena** (void)
  Funcion observadora del tipo de arena.
- int **getTypeVehicle** (void)
  Funcion observadora del tipo de vehículo.
- LPCWSTR **getLocalPlayerName** (void)
  Función observadora del nombre del jugador actual.
- bool **getPaused** (void)
  Comprueba si estamos o no en modo pausa.
- bool **getGamming** (void)
  Comprueba si estamos o no en modo juego.
- CDXUTDialog ***** **getGUI** ()
  Para acceder directamente al objeto GUI.

Métodos privados

- void **HideAll** (void)
  Para ocultar todos los elementos del GUI.

Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- IDirect3DDevice9 * **m_pDevice**
  Puntero al dispositivo de DirectX.
- CDXUTDialog ***** **m_pGUI**
  Cuadro para contener objetos.
- ID3DXFont * **m_pFont**
  Para dibujar textos.
- ID3DXSprite * **m_pTextSprite**
  Para efectos Sprite.
- bool **m_bStadistics**
  Flag para mostrar las estadísticas.

- int **m_iType**
  Tipo de GUI mostrado actualmente.
- int **m_ioldType**
  Tipo de GUI mostrado anteriormente.
- int **m_iTypeGame**
  Tipo de juego seleccionado.
- int **m_iTypeArena**
  Tipo de arena seleccionado.
- int **m_iTypeVehicle**
  Tipo de vehículo seleccionado.
- bool **m_bPaused**
  Indica si estamos o no en pausa.
- bool **m_bGamming**
  Indica si estamos o no jugando (pausados o no).
- UINT **m_uWidth**
  Dimensiones actuales del dispositivo (ancho).
- UINT **m_uHeight**
  Dimensiones actuales del dispositivo (alto).
- UINT **m_uMidW**
  Dimensiones actuales del dispositivo (ancho/2).
- UINT **m_uMidH**
  Dimensiones actuales del dispositivo (alto/2).
- int **m_iArena**
  Índice de la arena actualmente seleccionada.
- int **m_iVehicle1**
  Índice del vehículo del jugador 1 actualmente seleccionado.
- int **m_iVehicle2**
  Índice del vehículo del jugador 2 actualmente seleccionado.
- int **m_inumArenas**
  Número total de arenas encontradas.
- int **m_inumVehicles**
  Número total de vehículos encontrados.
- int **m_piIndex** [GUI_MAX_ITEMS_COMBOBOX]
  Vector en el que cada posición es el valor de un índice de los comboboxes del GUI.
- std::vector< CDXUTControl * > **m_aCtrls**
  Lista de controles del GUI.

45.17. Referencia de la Clase CMenu

`#include <Menu.h>`

Métodos públicos

- **CMenu** (void)
  Constructor.
- **~CMenu** (void)
  Destructor.
- bool **initialize** ()
  Carga el menú ante un CreateDevice.
- void **render** ()
  Renderiza el menú ante un FrameRender.
- void **update** ()
  Actualiza el menú ante un FrameMove.
- void **restore** ()
  Restaura objetos ante un ResetDevice.
- void **free** ()
  Libera objetos ante un LostDevice.
- void **Release** ()
  Libera el menú ante un DestroyDevice.
- **C3DLight** * **getMLight** (void)
  Retorna la luz utilizada.
- **C3DObjArena** * **getMArena** (void)
  Retorna la arena actualmente seleccionada.
- **C3DObjVehicle** * **getMVehicle1** (void)
  Retorna el vehículo del jugador 1 actualmente seleccionado.
- **C3DObjVehicle** * **getMVehicle2** (void)
  Retorna el vehículo del jugador 2 actualmente seleccionado.
- **CGame** * **getMGame** (void)
  Retorna el juego actual.
- float **getTotalTime** (void)
  Función observadora del tiempo total del menú.
- bool **setTypeMenu** (int dTipo, LPCWSTR dStr=NULL, int dTipoGame=0)
  Para comenzar el juego.

Métodos privados

- void **AnimDemo** (void)
  Para animar en el modo demo o pausa.

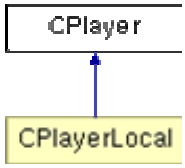
Atributos privados

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- **C3DLight * m_pLight**
  Luz por defecto.
- **C3DObjArena * m_pScene**
  Escena del menú.
- **C3DObject * m_pLogo**
  Objeto con el logo del videojuego.
- **C3DObjArena * m_pArena**
  Arena seleccionada.
- **C3DObjVehicle * m_pVehicle1**
  Vehículo seleccionado para el jugador 1.
- **C3DObjVehicle * m_pVehicle2**
  Vehículo seleccionado para el jugador 2.
- **CGame * m_pGame**
  Juego actual.
- **SDataGame m_sDataGame**
  Información sobre el juego (tipo, jugadores y arena).
- float **m_fTotalTime**
  Tiempo total transcurrido.
- float **m_fElapsedTime**
  Tiempo desde el último renderizado.
- int **m_iTipo**
  Tipo de menú mostrado.
- int **m_iEstadoAnim**
  Para las animaciones.
- bool **m_bLightIn**
  Flag que indica si la luz está apagada o encendida.

45.18. Referencia de la Clase CPlayer

`#include <Player.h>`

Diagrama de herencias de CPlayer



Métodos públicos

- **CPlayer** ()
  Constructor.
- **~CPlayer** ()
  Destructor.
- bool **initialize** (LPCWSTR dName, LPCWSTR dPathVehicle, **C3DObject** *dParent)
  Inicializa el jugador.
- void **render** ()
  Renderiza el jugador.
- void **update** ()
  Actualiza el jugador.
- void **Release** ()
  Libera los datos del jugador.
- float **getHealth** (void)
  Función observadora de la vida actual del jugador.
- void **setHealth** (float val)
  Función para establecer la vida actual del jugador.
- float **getRockets** (void)
  Función observadora del número de cohetes actuales del jugador.
- void **setRockets** (float val)
  Función para establecer un número de cohetes actuales del jugador.
- float **getMines** (void)
  Función observadora del número de minas actuales del jugador.
- void **setMines** (float val)
  Función para establecer un número de minas actuales del jugador.
- float **getObjectives** (void)
  Función observadora de los objetivos del jugador.
- void **setObjectives** (float val)
  Función para establecer los objetivos del jugador.
- bool **changedHealth** (void)
  Función para saber si en la última iteración cambió la vida del jugador.
- bool **changedRockets** (void)
  Función para saber si en la última iteración cambió el número de cohetes del jugador.
- bool **changedMines** (void)
  Función para saber si en la última iteración cambió el número de minas del jugador.

- bool **changedObjectives** (void)
  Función para saber si en la última iteración cambió el número de objetivos del jugador.
- float **getHealthSecond** (void)
  Función observadora de la vida perdida durante el último segundo.
- void **resetHealthSecond** (void)
  Función para reiniciar la vida perdida durante el último segundo.
- **C3DObjVehicle** * **getVehicle** (void)
  Función observadora del vehículo del jugador.

Métodos protegidos

- void **pressUp** ()
  Actualiza el jugador tras la pulsación de la orden UP.
- void **releaseUp** ()
  Actualiza el jugador tras la liberación de la orden UP.
- void **pressDown** ()
  Actualiza el jugador tras la pulsación de la orden DOWN.
- void **releaseDown** ()
  Actualiza el jugador tras la liberación de la orden DOWN.
- void **pressRight** ()
  Actualiza el jugador tras la pulsación de la orden RIGHT.
- void **releaseRight** ()
  Actualiza el jugador tras la liberación de la orden RIGHT.
- void **pressLeft** ()
  Actualiza el jugador tras la pulsación de la orden LEFT.
- void **releaseLeft** ()
  Actualiza el jugador tras la liberación de la orden LEFT.
- void **pressHandBrake** ()
  Actualiza el jugador tras la pulsación de la orden HANDBRAKE.
- void **releaseHandBrake** ()
  Actualiza el jugador tras la liberación de la orden HANDBRAKE.
- void **pressMine** ()
  Actualiza el jugador tras la pulsación de la orden MINE.
- void **releaseMine** ()
  Actualiza el jugador tras la liberación de la orden MINE.
- void **pressRocket** ()
  Actualiza el jugador tras la pulsación de la orden ROCKET.
- void **releaseRocket** ()
  Actualiza el jugador tras la liberación de la orden ROCKET.

Atributos protegidos

- bool **m_bInit**
  Flag que indica si el objeto está inicializado.
- LPWSTR **m_awName**
  Nombre del jugador.
- **C3DObjVehicle * m_pVehicle**
  Puntero al vehículo del jugador.
- **C3DObjArena * m_pArena**
  Puntero a la arena de juego.
- float **m_pOriFriction**
  Fricción original del vehículo.
- int **m_iStateUp**
  Flag que indica el estado de la orden de UP.
- int **m_iStateDown**
  Flag que indica el estado de la orden de DOWN.
- int **m_iStateRight**
  Flag que indica el estado de la orden de RIGHT.
- int **m_iStateLeft**
  Flag que indica el estado de la orden de LEFT.
- int **m_iStateHandBrake**
  Flag que indica el estado de la orden de HANDBRAKE.
- int **m_iStateMine**
  Flag que indica el estado de la orden de MINE.
- int **m_iStateRocket**
  Flag que indica el estado de la orden de ROCKET.
- int **m_bUp**
  Flag que refleja la orden de UP.
- int **m_bDown**
  Flag que refleja la orden de DOWN.
- int **m_bRight**
  Flag que refleja la orden de RIGHT.
- int **m_bLeft**
  Flag que refleja la orden de LEFT.
- int **m_bHandBrake**
  Flag que refleja la orden de HANDBRAKE.
- int **m_bMine**
  Flag que refleja la orden de MINE.
- int **m_bRocket**
  Flag que refleja la orden de ROCKET.
- float **m_pHealth**
  Vida actual del jugador.
- float **m_pRockets**
  Número de cohetes actuales del jugador.
- float **m_pMines**
  Número de minas actuales del jugador.
- float **m_pObjectives**

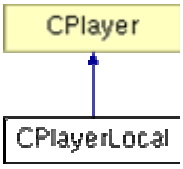
Número de objetos actuales del jugador.

- float **m_pLastHealth**
  Última vida del jugador.
- float **m_pLastRockets**
  Último número de cohetes del jugador.
- float **m_pLastMines**
  Último número de minas del jugador.
- float **m_pLastObjectives**
  Último número de objetivos del jugador.
- float **m_pHealthSecond**
  Vida perdida durante el último segundo.

45.19. Referencia de la Clase CPlayerLocal

`#include <PlayerLocal.h>`

Diagrama de herencias de CPlayerLocal



Métodos públicos

- **CPlayerLocal** ()
  Constructor.
- **~CPlayerLocal** ()
  Destructor.
- bool **initialize** (LPCWSTR dName, LPCWSTR dPathVehicle, **C3DObject** *dParent, int dType)
  Inicializa el jugador local.
- void **update** ()
  Actualiza el jugador local.
- int **getType** ()
  Funcion observadora del tipo de jugador.

Atributos privados

- int **m_iType**
  Tipo de jugador local, 1 o 2.

45.20. Referencia de la Clase CSettings

`#include <Settings.h>`

Métodos públicos

- **CSettings** ()
  Constructor.
- **~CSettings** ()
  Destructor.
- void **Apply** (int Section)
  Aplica la configuración establecida.
- void **RefreshControls** (int Section)
  Recarga el GUI con los datos.
- void **RefreshData** (int Section)
  Recarga los datos con el GUI.
- **SDataSettings** * **getData** (void)
  Toma la estructura con los datos de configuración.
- DWORD **getWidth** (void)
  Toma el ancho del dispositivo definido en la configuración.
- DWORD **getHeigth** (void)
  Toma el alto del dispositivo definido en la configuración.

Métodos privados

- void **LoadFile** (void)
  Carga los datos de configuración del fichero externo.
- void **SaveFile** (void)
  Guarda los datos de configuración del fichero externo.

Atributos privados

- **CXMLIO** * **m_pXMLIO**
  Puntero a una instancia del parseador XML.
- **SDataSettings m_DataSettings**
  Estructura con todos los datos de configuración de la aplicación.
- bool **m_bChanges**
  Flag que indica si hay cambios aún no aplicados en la configuración interna.
- bool **m_bChangesFile**
  Flag que indica si hay cambios aún no aplicados en la configuración del fichero externo.

45.21. Referencia de la Clase CXMLIO

`#include <XMLIO.h>`

Métodos públicos

- **CXMLIO** ()
  Constructor.
- **~CXMLIO** ()
  Destructor.
- bool **LoadFile** (LPCWSTR FileName)
  Carga un fichero.
- bool **SaveFile** (LPCWSTR FileName)
  Guarda el fichero actual.
- bool **CreateFile** (LPCWSTR FileName, LPCWSTR Root)
  Crea un fichero con el elemento raíz.
- bool **GetAttribute** (LPCWSTR Attribute, LPWSTR Value)
  Toma el valor de un atributo del elemento y fichero actuales.
- bool **SetAttribute** (LPCWSTR Attribute, LPCWSTR Value)
  Fija el valor de un atributo del elemento y fichero actuales.
- bool **GetElement** (LPWSTR Value)
  Toma el valor del elemento actual del fichero actual.
- bool **SetElement** (LPCWSTR Value)
  Fija el valor del elemento actual del fichero actual.
- bool **GetSubAttribute** (LPWSTR Element, LPCWSTR Attribute, LPWSTR Value, long num=0)
  Toma el valor de un atributo de un sub-elemento del elemento y fichero actuales.
- bool **SetSubAttribute** (LPWSTR Element, LPCWSTR Attribute, LPCWSTR Value, long num=0)
  Fija el valor de un atributo de un sub-elemento del elemento y fichero actuales.
- bool **GetSubElement** (LPWSTR Element, LPWSTR Value, long num=0)
  Toma el valor de un sub-elemento del elemento y fichero actuales.
- bool **SetSubElement** (LPWSTR Element, LPCWSTR Value, long num=0)
  Fija el valor de un sub-elemento del elemento y fichero actuales.
- bool **MoveSubElement** (LPWSTR Element, long num=0)
  Moverse al elemento hijo indicado del elemento actual y fichero XML actual.
- bool **MoveParent** (void)
  Moverse al elemento padre del elemento actual y fichero XML actual.
- bool **MoveRoot** (void)
  Moverse al elemento raíz del fichero XML actual.

Métodos privados

- bool **SubElement** (LPWSTR Element, bool create=false, long num=0)
  Almacena en 'm_pSubElm' el elemento hijo indicado del elemento actual y fichero XML actual.

Atributos privados

- IXMLDOMDocument * **m_pXMLDoc**
  Objeto COM para el manejo de XML.
- CComPtr< IXMLDOMElement > **m_pThisElm**
  Puntero al nodo actual del fichero.
- CComPtr< IXMLDOMElement > **m_pSubElm**
  Puntero para accesos a subnodos del nodo actual.
- bool **m_Opened**
  Flag que indica si hay un fichero abierto.

45.22. Referencia de la Estructura Rank

`#include <3DObjArena.h>`

Atributos públicos

- WCHAR **m_Player** [MAX_STRING+1]
  Nombre del jugador.
- WCHAR **m_Vehicle** [MAX_STRING+1]
  Nombre del vehículo.
- float **m_Time**
  Tiempo en el que ha realizado la arena.

45.23. Referencia de la Estructura SDataGame

`#include <Game.h>`

Atributos públicos

- int **m_iType**
  Tipo de juego.
- std::basic_string< wchar_t > **m_awArena**
  Path de la arena.
- std::basic_string< wchar_t > **m_awLVehicle1**
  Path del vehículo del jugador 1.
- std::basic_string< wchar_t > **m_awLVehicle2**
  Path del vehículo del jugador 2.
- std::basic_string< wchar_t > **m_awLName**
  Nombre del jugador local.

45.24. Referencia de la Estructura SDataSettings

`#include <Settings.h>`

Atributos públicos

- int **m_iQuality**
  Calidad del display.
- int **m_iWinFull**
  Pantalla completa activada o desactivada.
- int **m_iStadistics**
  Estadísticas gráficas activadas o desactivadas.
- unsigned long **m_ulResolution**
  Resolución de pantalla.
- int **m_iFilter**
  Tipo de filtro de texturas.
- int **m_iMipMapping**
  MipMapping activado o desactivado.
- int **m_iEffects**
  Volumen de los efectos sonoros.
- WCHAR **m_awNomPlayer** [MAX_STRING+1]
  Nombre por defecto para el jugador local.

45.25. Referencia de la Estructura SMeshEntry

`#include <3DMesh.h>`

Atributos públicos

- LPWSTR **MeshPath**
  Path dónde se encuentra el modelo gráfico.
- LPD3DXMESH **Mesh**
  Geometria.
- std::vector< D3DMATERIAL9 > * **Mtrls**
  Lista de Materiales.
- std::vector< IDirect3DTexture9 * > * **Textures**
  Lista de Texturas.
- unsigned short **num**
  Número de referencias que tiene la geometría del objeto.

46. Documentación de archivos

46.1. Referencia del Archivo 3DCamera.h

Descripción detallada

Fichero con la especificación de los atributos y funciones de las cámara 3D que se pueden situar en la escena

Clases

- class **C3DCamera**
  Clase para manejar la cámara 3D de la escena.

Definiciones

- #define **CAMERA3D_SPEED** 4.0f
  Velocidad de movimiento lineal de la cámara.
- #define **CAMERA3D_A_SPEED** 0.1f
  Velocidad de movimiento angular de la cámara.
- #define **CAMERA3D_USER** -1
  Tipo de cámara de usuario.
- #define **CAMERA3D_FREE** 0
  Tipo de cámara libre.
- #define **CAMERA3D_GAME_1** 1
  Tipo de cámara persecutoria del juego (tipo 1).
- #define **CAMERA3D_GAME_2** 2
  Tipo de cámara persecutoria del juego (tipo 2).
- #define **CAMERA3D_GAME_3** 3
  Tipo de cámara persecutoria del juego (tipo 3).
- #define **CAMERA3D_GAME_4** 4
  Tipo de cámara persecutoria del juego (tipo 4).
- #define **CAMERA3D_GAME_5** 5
  Tipo de cámara persecutoria del juego (tipo 5).
- #define **CAMERA3D_GAME_6** 6
  Tipo de cámara persecutoria del juego (tipo 6).
- #define **CAMERA3D_GAME_7** 7
  Tipo de cámara persecutoria del juego (tipo 7).

46.2. Referencia del Archivo 3DCollision.h

Descripción detallada

Cada objeto de la escena puede tener asociado un área de colisión, todas las operaciones relacionadas con las colisiones se manejan desde este módulo

Clases

- class **C3DCollision**
  Clase para manejar el área de colisión de los objetos de la escena.

Definiciones

- #define **COLLISION_EPSILON** 0.1f
  Distancia adicional para evitar efecto pegado en la recolocación tras colisiones.
- #define **COLLISION_E_MF** 0.2f
  Coeficiente de elasticidad ante colisiones móvil-fijo.
- #define **COLLISION_E_MM** 0.2f
  Coeficiente de elasticidad ante colisiones móvil-móvil.
- #define **COLLISION_TYPE_SRO1** 1
  Tipo de colisión detectada en SRO(1) dónde (2) mete esquina en (1).
- #define **COLLISION_TYPE_SRO2** 2
  Tipo de colisión detectada en SRO(2) dónde (1) mete esquina en (2).

46.3. Referencia del Archivo 3DLight.h

Descripción detallada

Fichero con la especificación de los atributos y funciones de las luces 3D que se pueden situar en la escena

```
#include <d3d9.h>
```

Clases

- class **C3DLight**
  Clase para manejar las luces 3D de la escena.

46.4. Referencia del Archivo 3DMesh.h

Descripción detallada

Cada objeto de la escena puede tener asociado un modelo gráfico, todas las operaciones relacionadas con los modelos gráficos se manejan desde este módulo

```
#include <vector>
```

Clases

- struct **SMeshEntry**
  Estructura para almacenar por cada geometría de objeto en memoria, para evitar cargar dos veces la misma.
- class **C3DMesh**
  Clase para manejar los modelos gráficos de la escena.

Definiciones

- #define **MESH_ALPHA_NON** 0
  Tipo de geometría no transparente (tipo 0).
- #define **MESH_ALPHA_CAM** 1
  Tipo de geometría transparente ante cámara persecutoria (tipo 1).
- #define **MESH_ALPHA_YES** 2
  Tipo de geometría transparente siempre (tipo 2).

46.5. Referencia del Archivo 3DObjArena.h

Descripción detallada

Este módulo ofrece un tratamiento específico a los objetos de tipo arena

```
#include "3DObject.h"
#include <vector>
#include <list>
```

Clases

- struct **Rank**
  Estructura para mantener la información del ranking de mejores tiempos de la arena.
- class **C3DObjArena**
  *Clase hija de **C3DObject** para para manejar las arenas de la escena.*

Definiciones

- #define **OBJARENA_DIR** L".\\models\\arenas\\"
  Directorio dónde se deben buscar los ficheros descriptivos de las arenas del juego.
- #define **OBJARENA_NUM_RANK** 4
  Número máximo de entradas en el ranking.
- #define **OBJARENA_MAX_RANKTXT** (MAX_STRING*OBJARENA_NUM_RANK)
  Tamaño máximo del ranking de la arena.

46.6. Referencia del Archivo 3DObject.h

Descripción detallada

Cada objeto de la escena puede tener asociado: un área de colisión, un sistema físico, un modelo gráfico, una lista de objetos que lo componen. Este módulo implementa todas las operaciones para gestionar el objeto

```
#include <vector>
#include "3DMesh.h"
#include "3DCollision.h"
#include "3DPhysics.h"
#include "3DSound.h"
#include "3DParticle.h"
#include "XMLIO.h"
```

Clases

- class **C3DObject**
  Clase base para manejar los objetos de la escena.

Definiciones

- #define **OBJECT_TYPE_MOBILE** 1
  Tipo de objeto colisionable móvil.
- #define **OBJECT_TYPE_PHYSICS** 2
  Tipo de objeto colisionable fijo con física.
- #define **OBJECT_TYPE_FIXED** 3
  Tipo de objeto colisionable fijo sin física.
- #define **OBJECT_TYPE_NOCOLL** 4
  Tipo de objeto no colisionable.

46.7. Referencia del Archivo 3DObjElement.h

Descripción detallada

Este módulo ofrece un tratamiento específico a los objetos de tipo elemento

```
#include "3DObject.h"
#include "3DObjArena.h"
```

Clases

- class **C3DObjElement**
  *Clase hija de **C3DObject** para para manejar los elementos de la arena.*

46.8. Referencia del Archivo 3DObjVehicle.h

Descripción detallada

Este módulo ofrece un tratamiento específico a los objetos de tipo vehículo

```
#include "3DObject.h"
#include "3DPhyVehicle.h"
```

Clases

- class **C3DObjVehicle**
  *Clase hija de **C3DObject** para para manejar los vehículos de la escena.*

Definiciones

- #define **OBJVEHICLE_DIR** L".\\models\\vehicles\\"
  Directorio dónde se deben buscar los ficheros descriptivos de los vehículos del juego.

46.9. Referencia del Archivo 3DObjWeapon.h

Descripción detallada

Este módulo ofrece un tratamiento específico a los objetos de tipo arma

```
#include "3DObject.h"
#include "3DObjVehicle.h"
```

Clases

- class **C3DObjWeapon**
  *Clase hija de **C3DObject** para para manejar las armas del vehículo.*

46.10. Referencia del Archivo 3DParticle.cpp

Descripción detallada

Este módulo ofrece el tratamiento de los sistemas de partículas de la escena

```
#include "dxstdafx.h"
#include <tchar.h>
#include <vector>
#include "3DParticle.h"
```

46.11. Referencia del Archivo 3DParticle.h

Descripción detallada

Este módulo ofrece el tratamiento de los sistemas de partículas de la escena

```
#include <d3dx9.h>
```

Clases

- struct **Particle**
  Estructura para mantener cada partícula.
- struct **PointVertex**
  Estructura para vértices y FVF personalizados.
- class **C3DParticles**
  Clase C3DParticles para manejar sistemas de partículas.

46.12. Referencia del Archivo 3DPhysics.h

Descripción detallada

Cada objeto de la escena puede tener asociado un sistema físico, todas las operaciones relacionadas con el sistema físico se manejan desde este módulo

Clases

- class **C3DPhysics**
  Clase para manejar el sistema físico de los objetos de la escena.

46.13. Referencia del Archivo 3DPhyVehicle.h

Descripción detallada

Los vehículos tienen una física ligeramente diferente al resto de los objetos
`#include "3DPhysics.h"`

Clases

- class **C3DPhyVehicle**
  Clase para manejar el sistema físico concreto de los vehículos.

Definiciones

- #define **PHYVEHICLE_FRICTION_COEF** 30
  Coeficiente de rozamiento lineal para movimientos adicionales.

46.14. Referencia del Archivo 3DSound.h

Descripción detallada

Cada objeto de la escena puede tener asociado un sonido, todas las operaciones relacionadas con los sonidos se manejan desde este módulo

Clases

- class **C3DSound**
  Clase para manejar los sonidos 3D de los objetos de la escena.

Definiciones

- #define **SOUND3D_DIR** L".\\sounds\\"
  Directorio donde se deben buscar los ficheros de sonido.
- #define **SOUND3D_DOPPLER_FACTOR** 0.0f
  Nivel del efecto doppler del sonido 3D.
- #define **SOUND3D_ROLL_OFF_FACTOR** 0.2f
  Nivel de decaimiento del sonido 3D con la distancia.
- #define **SOUND3D_MIN_DISTANCE** 5.0f
  Distancia mínima de decaimiento del sonido 3D.
- #define **SOUND3D_MAX_DISTANCE** 150.0f
  Distancia máxima de decaimiento del sonido 3D.

46.15. Referencia del Archivo 3DSoundMng.h

Descripción detallada

Esta clase abstrae todas las operaciones de gestión de los sonidos 3D de la escena

```
#include <vector>
#include "3DSound.h"
```

Clases

- class **C3DSoundMng**
  Clase para gestionar los sonidos 3D de la escena.

46.16. Referencia del Archivo Game.h

Descripción detallada

```
#include <string>
#include "PlayerLocal.h"
#include "3DObjArena.h"
```

Clases

- struct **SDataGame**
  Tipo de dato con los datos de un juego, necesarios para la inicialización.
- class **CGame**
  Clase base para manejar los jugadores de la escena.

Definiciones

- #define **GAME_1PLAYER** 1
  Juego un jugador.
- #define **GAME_2PLAYER_FIGHT** 2
  Juego dos jugadores a lucha.
- #define **GAME_2PLAYER_OBJS** 3
  Juego dos jugadores a objetivos.
- #define **GAME_TIME_INI_GAME** -1.0f
  Tiempo de inicio del juego.
- #define **GAME_TIME_OBJ** 1
  Segundos para aparición del objetivo.
- #define **GAME_TIME_ELE** 2
  Segundos para aparición de los elementos.
- #define **GAME_LOW_TIME** 11.0f
  Tiempo bajo dónde se emite el sonido de aviso.

46.17. Referencia del Archivo GUI.h

Descripción detallada

Fichero con la especificación de los atributos y funciones para el manejo de los componentes gráficos referentes al interfaz gráfico de usuario

```
#include <d3d9.h>
#include <vector>
```

Clases

- class **CGUI**
  Clase para manejar el interfaz gráfico de usuario.

Definiciones

- #define **GUI_MAX_ITEMS_COMBOBOX** 256
  Número máximo permitido de elementos en un combobox.

Funciones

- void CALLBACK **OnGUIEvent** (UINT nEvent, int nControlID, CDXUTControl *pControl)
  Manejador de eventos.

46.18. Referencia del Archivo Main.h

Descripción detallada

Fichero con la función principal de la aplicación y con las funciones manejadoras de eventos del bucle principal del programa gráfico

```
#include "dxstdafx.h"
#include "resource.h"
```

Definiciones

- #define **MAIN_WIN_TITLE** L"SHIFT Videogame"
  Título de la ventana de la aplicación.

Funciones

- INT WINAPI **WinMain** (HINSTANCE, HINSTANCE, LPSTR, int)
  Función principal de la aplicación.
- HRESULT CALLBACK **OnCreateDevice** (IDirect3DDevice9 *pd3dDevice, const D3DSURFACE_DESC *pBackBufferSurfaceDesc)
  Función manejadora del evento 'crear el dispositivo'.
- void CALLBACK **OnDestroyDevice** ()
  Función manejadora del evento 'destruir el dispositivo'.
- HRESULT CALLBACK **OnResetDevice** (IDirect3DDevice9 *pd3dDevice, const D3DSURFACE_DESC *pBackBufferSurfaceDesc)
  Función manejadora del evento 'resetear el dispositivo'.
- void CALLBACK **OnLostDevice** ()
  Función manejadora del evento 'perder el dispositivo'.
- void CALLBACK **OnFrameMove** (IDirect3DDevice9 *pd3dDevice, double fTime, float fElapsedTime)
  Función manejadora del evento 'mover la imagen'.
- void CALLBACK **OnFrameRender** (IDirect3DDevice9 *pd3dDevice, double fTime, float fElapsedTime)
  Función manejadora del evento 'renderizar la imagen'.
- LRESULT CALLBACK **MsgProc** (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam, bool *pbNoFurtherProcessing)
  Función manejadora del evento 'procesar mensajes de windows'.
- void CALLBACK **KeyboardProc** (UINT nChar, bool bKeyDown, bool bAltDown)
  Función manejadora del evento 'procesar mensajes del teclado'.
- bool **IniApp** (void)
  Función de inicialización de la aplicación.
- void **EndApp** (void)
  Función de des-inicialización de la aplicación.

46.19. Referencia del Archivo Menu.h

Descripción detallada

Fichero con la especificación de los atributos y funciones para la representación de la escena 3D que se muestra durante el menú del juego

```
#include "3DLight.h"
#include "3DMesh.h"
#include "3DObjArena.h"
#include "3DObjVehicle.h"
#include "Game.h"
```

Clases

- class **CMenu**
  Clase para manejar la escena 3D del menú.

Definiciones

- #define **MENU_TYPE_MENU** 1
  Tipo de menú normal.
- #define **MENU_TYPE_SELECT_ARENA** 2
  Tipo de menú para seleccionar arena.
- #define **MENU_TYPE_SELECT_VEHICLE1** 3
  Tipo de menú para seleccionar vehículo del jugador 1.
- #define **MENU_TYPE_SELECT_VEHICLE2** 4
  Tipo de menú para seleccionar vehículo del jugador 2.
- #define **MENU_TYPE_PREGAME** 5
  Tipo de menú justo antes del juego.
- #define **MENU_TYPE_GAME** 6
  Tipo de menú para el modo juego.
- #define **MENU_TYPE_PAUSE** 7
  Tipo de menú para el modo pausa.
- #define **MENU_ARENAS_DIR** L".\\models\\arenas\\"
  Directorio dónde se deben buscar los ficheros descriptivos de las arenas del juego.
- #define **MENU_VEHICLES_DIR** L".\\models\\vehicles\\"
  Directorio dónde se deben buscar los ficheros descriptivos de los vehículos del juego.
- #define **MENU_MODELS_EXTENSION** L".xml"
  Extensión de los ficheros descriptivos de los modelos, tanto para arenas como para vehículos.
- #define **MENU_ANIM_SEG** 7.0f
  Periodo de tiempo para cambio de animación.
- #define **MENU_INI_CAMERA** 2.0f
  Tiempo de inicio de la cámara.
- #define **MENU_LIGHTOFF_SPD** 0.3f
  Velocidad de apagado de las luces.
- #define **MENU_LIGHTON_SPD** 0.3f
  Velocidad de encendido de las luces.

46.20. Referencia del Archivo Player.h

Descripción detallada

Cada jugador tiene asociado un vehículo. Este módulo implementa todas las operaciones para gestionar los jugadores

```
#include "3DObjVehicle.h"
#include "3DObjArena.h"
```

Clases

- class **CPlayer**
  Clase base para manejar los jugadores de la escena.

Definiciones

- #define **PLAYER_MAX_SEC_HEALTH** 2.5f
  Máxima pérdida de vida por segundo permitida.

46.21. Referencia del Archivo PlayerLocal.h

Descripción detallada

Este módulo incorpora las operaciones para manejar un vehículo por medio de dispositivos locales, como el teclado

```
#include "Player.h"
```

Clases

- class **CPlayerLocal**
  *Clase hija de **CPlayer** para manejar los jugadores de tipo local.*

Definiciones

- #define **PLAYERLOCAL_1** 1
  Jugador principal.
- #define **PLAYERLOCAL_2** 2
  Jugador adicional.

46.22. Referencia del Archivo Settings.h

Descripción detallada

Fichero con la especificación de los atributos y funciones para el mantenimiento de la configuración personalizada de la aplicación de forma persistente

```
#include "XMLIO.h"
```

Clases

- struct **SDataSettings**
  Tipo de dato con la estructura para mantener la configuración de la aplicación.
- class **CSettings**
  Clase para manejar la configuración de la aplicación.

Definiciones

- #define **SETTINGS_FILE** L".\\config\\settings.xml"
  Fichero externo con los datos de configuración de la aplicación.
- #define **MIN_WIDTH** 640
  Ancho en pixeles mínimo permitido para el modo de video.
- #define **MIN_HEIGHT** 480
  Alto en pixeles mínimo permitido para el modo de video.

46.23. Referencia del Archivo XMLIO.h

Descripción detallada

Se usa para la carga y almacenamiento persistente de datos de configuración, datos de programa o propiedades de objetos.

```
#include <msxml2.h>
#include <atlcomcli.h>
```

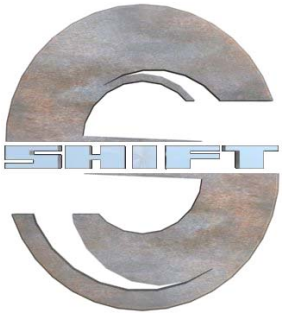
Clases

- class **CXMLIO**
  Clase para manejar la entrada y salida de ficheros XML.

Capítulo 12

Sitio Web

16 de marzo de 2008



Capítulo 12: Sitio Web

Tabla de apartados

Lista de figuras

47. Introducción (*sitio Web*)

Desde un principio, el objetivo del proyecto ha sido el desarrollo completo de un videojuego, tratando de cubrir todos los elementos internos y externos necesarios. En este sentido, la Web del videojuego constituye un elemento externo esencial, haciendo llegar los recursos generados al público objetivo y permitiendo la gestión del contenido del proyecto.

Teniendo en cuenta esto, se ha diseñado y mantenido una Web con las siguientes cualidades:

- Tecnologías Web estándar
- Distribución correcta y diseño cuidado
- El español es el idioma principal, aunque gran parte del contenido se traduce al inglés
- Debe facilitarse la gestión del contenido para futuras amplicaciones
- Diseño propio, no basado en ninguna plantilla ni gestor de contenido

En los siguientes apartados se va a detallar el diseño de esta Web, que reside bajo la siguiente URL:

http://shift.delblanco.es

48. Características

En cuanto a las **tecnologías utilizadas**:

- **XHTML 1.0 Estricto** Se debe hacer un uso correcto del estándar, validando el contenido mediante la herramienta online del WC3
- **CSS** Al igual que antes, usando correctamente el estándar y validando las hojas de estilo creadas.
- **RSS** Se añaden semillas para noticias relacionadas con el desarrollo, actualizaciones, versiones, etc.
- **JavaScript** Permite derivar parte de la carga de presentación al cliente

En cuanto al **contenido**:

- **Diseño** Diseño propio y creativo, bajo el mismo criterio y tonalidades que el propio videojuego.
- **Idiomas** Se permite el español y el inglés. Sin embargo se mantiene el español como idioma principal y no todo el contenido se encuentra traducido.
- **Recursos** Cualquier recurso utilizado para el desarrollo del proyecto estará disponible en esta Web, ya sean documentos, binarios, fuentes, modelos, etc.
- **Gestión** El contenido de la Web se almacena en ficheros XML locales, que serán traducidos a XHTML mediante un XLS propio y trasmitidos al servidor mediante un programa de sincronización. La gestión del sitio se se reduce por tanto a la gestión local del contenido XML.

49. Recursos disponibles

49.1. Apartado "*Principal*"

Desde el apartado inicial se mostrará una pequeña descripción del proyecto y las últimas noticias acerca del mismo.

Figura 94 : Web (página principal)

Las noticias podrán detallarse en una segunda página dónde se incluye todo el histórico desde el inicio del proyecto. Esta pantalla constituye una bitácora real del desarrollo del videojuego.

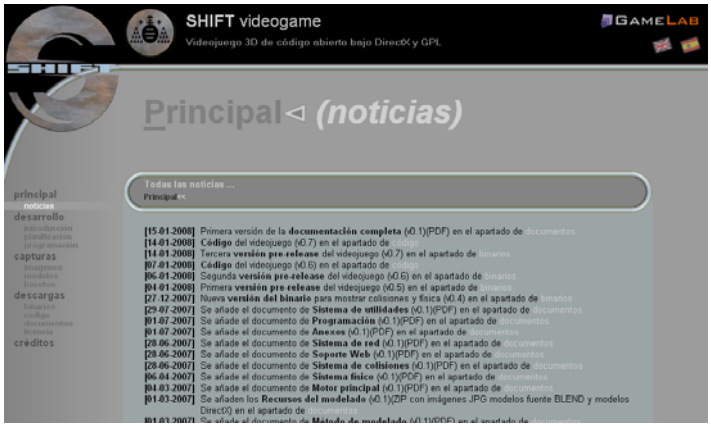


Figura 95 : Web (bitácora de noticias)

49.2. Apartado "*Desarrollo*"

Este es un apartado importante, dónde se muestra la documentación del código autogenerada con la herramienta Doxigen.

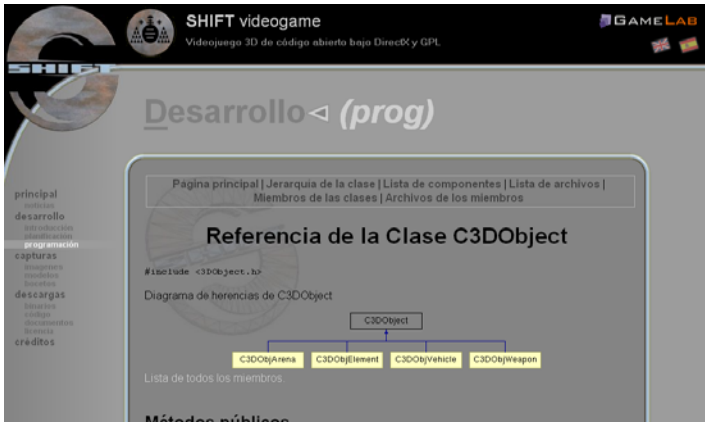


Figura 96 : Web (introducción, planificación y desarrollo del proyecto)

La salida de esta herramienta está correctamente adaptada a los estilos de la Web, por lo que visitar este apartado no supone un cambio en la navegación de la Web.

49.3. Apartado "*Capturas*"

En esta sección se muestran los primeros bocetos que dieron la idea del videojuego, los modelos creados e imágenes reales del videojuego, comparando de este modo la idea inicial con los resultados finales obtenidos.

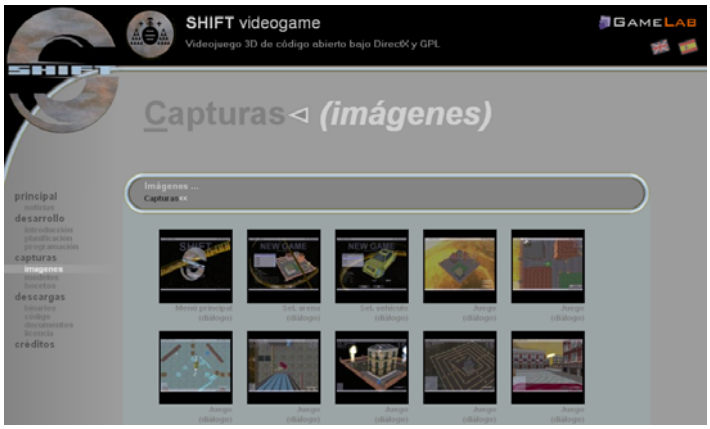


Figura 97 : Web (capturas, modelos y bocetos iniciales)

49.4. Apartado "*Descargas*"

En este apartado tenemos disponibles todos los recursos del proyecto, desde los documentos de desarrollo en PDF, las diferentes versiones de código, las diferentes versiones de los binarios, etc.

Figura 98 : Web (descargas de binarios, código y documentos)

Además se detalla la licencia que legisla el uso de los recursos ofrecidos (GPLv3). Para más detalles sobre esta licencia puede consultar su definición legal en el apartado *54. ANEXO: Licencia GPLv3*.

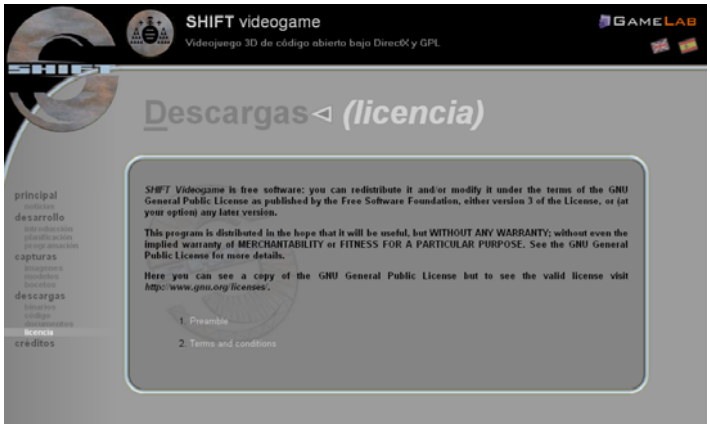
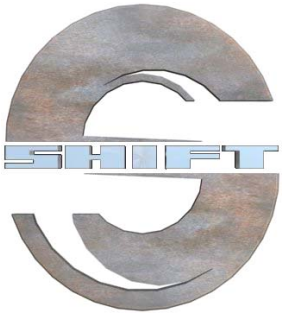


Figura 99 : Web (licencia de los recursos del proyecto)

Capítulo 13

Mejoras propuestas

16 de marzo de 2008



Capítulo 13: Mejoras propuestas

Tabla de apartados

50. Introducción (*mejoras*)

Desde el inicio del proyecto no se ha planteado un alcance ambicioso, más aún considerando la complejidad que supone el desarrollo del software de entretenimiento y todos sus recursos asociados.

En este sentido, el proyecto puede ser extendido en múltiples aspectos y, de hecho, la intención a futuro es desarrollarlos fuera del ámbito que describe el presente proyecto.

A continuación se describirán algunas de estas extensiones que suponen una mejora sustancial en el proyecto y con una integración directa en el diseño actual.

51. Mejoras propuestas

51.1. Sistema multijugador

La mejora más sustancial de cara a la jugabilidad del videojuego es la integración de un módulo de red que permita conexiones remotas y la integración de un sistema multijugador. Esta posibilidad fue considerada en el inicio del proyecto, pero descartada más adelante debido a los tiempos planteados.

Actualmente, el motor gráfico se encuentra preparado para la integración de un motor de red. Tan sólo sería necesario integrar una clase *Network* con el protocolo de comunicación y una clase *RemotePlayer* que especializase la clase *Player* para el jugador remoto (al igual que *LocalPlayer* implementa al jugador local).

El protocolo debería poder mantener los datos de la escena entre las diferentes instancias de red. Para ello se podría tomar una **estructura lógica de anillo**, dónde se trasmitiese la información global de la escena junto con el testigo de la red. En cada momento, cada estación que reciba el testigo, debería:

- Actualizar sus datos locales del resto de jugadores con la información global que reciba
- Actualizar los datos globales de él mismo con los datos locales que tiene
- Pasar el testigo a la siguiente estación del anillo

En la red se deberían distinguir dos tipos de procesos, un **proceso administrador** que se encarga de la administración del anillo y varios **procesos estaciones** que lo forman. Desde el punto de vista del videojuego:

- El que crea la partida multijugador arrancaría un proceso administrador y un proceso estación, es decir actuaría como servidor y cliente al mismo tiempo.
- El resto de jugadores que se uniesen a la partida, arrancarían sólo un proceso estación que se uniría al anillo por medio del proceso administrador.

Existen varias posibilidades para implementar este protocolo, pero para seguir con la escencia del resto del proyecto sería conveniente seguir las siguientes premisas:

- Se descartarían librerías de comunicaciones para videojuegos (como por ejemplo las que incluye DirectX) debido al carácter educativo del proyecto.
- Se descartaría implementar las comunicaciones a nivel de aplicación TCP, ya que de cara a los videojuegos es más conveniente el nivel de transporte UDP. Esto complicaría el protocolo, pero aumentaría la fluidez de la comunicación.

La capa UDP dispone una comunicación extremo a extremo multiplexada mediante puertos, pero no proporciona mecanismos de acuse de recibo, secuenciación de mensajes, control de flujo, detección de errores, etc. Por tanto, el protocolo debería disponer de ciertos servicios para ofrecer un nivel razonable de fiabilidad en la conexión.

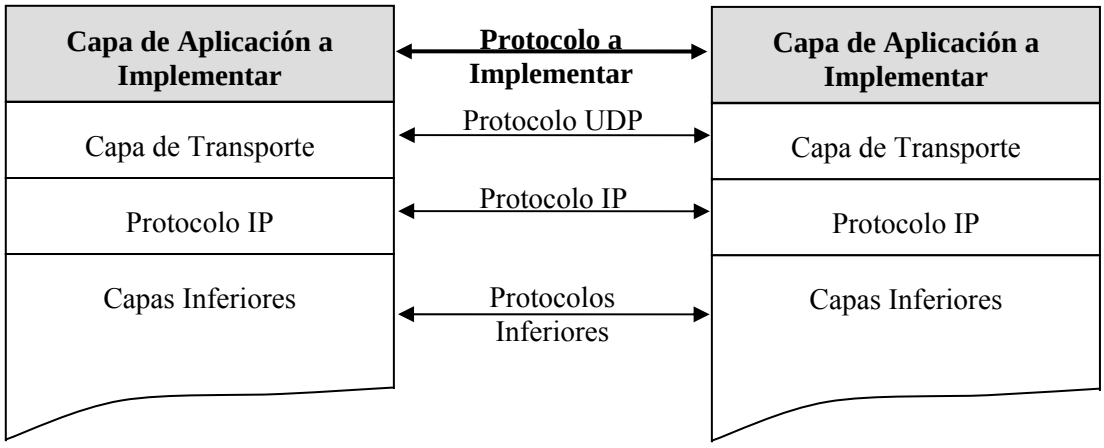


Figura 100 : Arquitectura del posible motor de red

51.2. Editor de arenas y vehículos

Como ya se comentaba en los apartados *6.1. Arenas* y *6.2. Vehículos*, el videojuego debe permitir la incorporación de nuevas arenas y vehículos aparte de los existentes.

Tanto es así que al inicio del videojuego se escanean los directorios en busca de los ficheros XML con los vehículos y arenas a mostrar. El motor gráfico es el encargado de parsear cada fichero XML e instanciar los objetos correctamente.

Por lo tanto tenemos un interfaz externo para comunicar al videojuego las características y composición de cada elemento, de forma que incluir nuevas arenas o vehículos se traduce a crear ficheros XML que los definan.

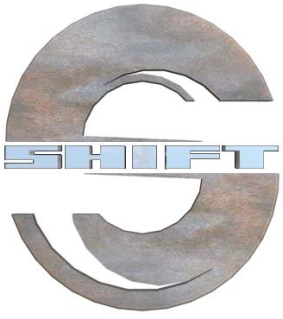
Los vehículos y arenas actuales han sido creados editando manualmente los ficheros XML, pero sería interesante desarrollar una aplicación externa que permitiese generarlos de forma gráfica.

Con este editor resultaría muy cómodo componer nuevas arenas o definir nuevos vehículos, pudiendo extender o personalizar la aplicación a gusto del usuario.

Capítulo 14

Bibliografía

16 de marzo de 2008



Capítulo 14: Bibliografía

El proyecto ha tratado de ser autocontenido, lo que ha permitido plantear un enfoque autodidáctico durante su desarrollo.Debido a esto, se han reducido considerablemente las fuentes externas utilizadas, haciendo uso casi en exclusiva de recursos propios de la Universidad de Oviedo:

La bibliografía que aquí se expone son por tanto meras referencias a información y recursos que han permitido el desarrollo de la aplicación:

- **Información general sobre informática gráfica**

  Recursos de la asignatura "Informática Gráfica"

  Impartida por Ricardo Ramos, Universidad de Oviedo (2003)

- **Información específica sobre DirectX:**

  Recursos del curso "Programación Gráfica para Videojuegos sobre DirectX9 / C++"

  Impartido por Iván Fernández Lobo, Universidad de Oviedo (2004)

- **Sistema de colisiones:**

  Información tomada de "http://www.myphysicslab.com"

- **Otros recursos copyleft:**

  Texturas: "http://www.3dcafe.com/"

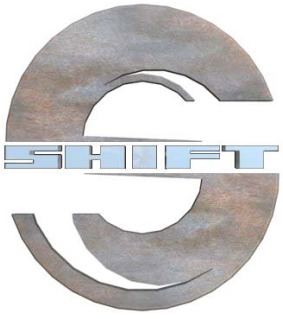
  Sonidos: "http://www.findsounds.com"

  Blueprints de vehículos comerciales: "http://www.onnovanbraam.com/"

Capítulo 15

Anexos

16 de marzo de 2008



Capítulo 15: Anexos

Tabla de apartados

52. ANEXO: Licencia DirectX 9.0c SDK

Se añade aquí el contrato de uso EULA (End-User License Agreement) del paquete de desarrollo DirectX 9.0c SDK. Este contrato permite el uso del paquete para el desarrollo y distribución de software:

END-USER LICENSE AGREEMENT FOR MICROSOFT SOFTWARE

DirectX 9.0 Software Development Kit Update (Summer 2004)

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation ("Microsoft") for the Microsoft software that accompanies this EULA, which includes computer software and may include associated media, printed materials, "online" or electronic documentation, and Internet-based services ("Software"). An amendment or addendum to this EULA may accompany the Software. YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE. IF YOU DO NOT AGREE, DO NOT INSTALL, COPY, OR USE THE SOFTWARE; YOU MAY RETURN IT TO YOUR PLACE OF PURCHASE (IF APPLICABLE) FOR A FULL REFUND.

1. GRANTS OF LICENSE. Microsoft grants you the following rights provided that you comply with all terms and conditions of this EULA:

1.1 General License Grant. Microsoft grants to you as an individual, a personal, nonexclusive license to make and use copies of the Software for the purposes of designing, developing, and testing your software product(s), provided that you are the only individual using the Software.

If you are an entity, Microsoft grants to you a personal, nonexclusive license to make and use copies of the Software, provided that for each individual using the Software within your organization, you have acquired a separate and valid license for each such individual.

1.2 Documentation. You may make and use an unlimited number of copies of any documentation, provided that such copies shall be used only for personal purposes and are not to be republished or distributed (either in hard copy or electronic form) beyond your premises.

1.3 Storage/Network Use. You may also store or install a copy of the Software on a storage device, such as a network server, used only to install or run the Software on computers used by a licensed end user in accordance with Section 1.1. A single license for the Software may not be shared or used concurrently by multiple end users.

2. ADDITIONAL LICENSE RIGHTS - REDISTTIBUTABLE CODE.

In addition to the rights granted in Section 1, certain portions of the Software, as described in this Section 2, are provided to you with additional license rights. These additional license rights are conditioned upon your compliance with the distribution requirements and license restrictions described in Section 3.

2.1 Sample Code.

(a) Microsoft grants you the right to: (i) use and modify the source code version of those portions of the Software identified as "Samples" in the Software ("Sample Code") for the sole purposes of designing, developing, and testing your software product(s), and (ii) a limited, nonexclusive, royalty-free right to reproduce and distribute the Sample Code, along with any modifications thereof, in object and/or source code form. For applicable redistribution requirements for Sample Code, see Section 3 below.

(b) Sample Code is identified as all of the files in the following directories and subdirectories of the Software:

(i) <Installed SDK Location>\Samples\C++

<Installed SDK Location>\Samples\Managed

<Installed SDK Location>\Samples\Media

<Installed SDK Location>\Utilities\MView

<Installed SDK Location>\Utilities\Content Creation Tool Plug-Ins

(c) Additional Sample Code may be available by separate download at www.microsoft.com/msdn/directx and may be identified as all of the files sint eh following directories and subdirectires:

(i) <SDK CD ROOT>\Extras\Direct3D\Plug-Ins

<SDK CD ROOT>\Extras\DirectShow

2.2 Redistributable Code. Microsoft grants you a nonexclusive, royalty-free right to reproduce and distribute the object code form of any portion of the Software listed in <Installed SDK Location>\Documentation\License Agreements\DirectX Redist.TXT ("Redistributable Code"). For general redistribution requirements for Redistributable Code, see Section 3 below.

3. DISTRIBUTION REQUIREMENTS AND OTHER LICENSE LIMITATIONS.

If you choose to exercise your rights under Section 2, any redistribution by you is subject to your compliance with Section 3.

3.1 General Distribution Requirements.

(a) If you choose to redistribute Sample Code, or Redistributable Code (collectively, the "Redistributables") as described in Section 2, you agree: (i) except as otherwise noted in Section 2.1 (Sample Code), to distribute the Redistributables only in object code form and in conjunction with and as a part of a software application product developed by you that adds significant and primary functionality to the Redistributables ("Licensee Software"); (ii) that the Redistributables only operate in conjunction with Microsoft Windows platforms; (iii) that if the Licensee Software is distributed beyond Licensee's premises or externally from Licensee's organization, to distribute the Licensee Software containing the Redistributables pursuant to an end user license agreement (which may be "break-the-seal", "click-wrap" or signed), with terms no less protective than those contained in this EULA; (iv) not to use Microsoft's name, logo, or trademarks to market the Licensee Software; (v) to display your own valid copyright notice which shall be sufficient to protect Microsoft's copyright in the Software; (vi) not to remove or obscure any copyright, trademark or patent notices that appear on the Software as delivered to you; (vii) to indemnify, hold harmless, and defend Microsoft from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of the Licensee Software; (viii) to otherwise comply with the terms of this EULA; and (ix) that Microsoft reserves all rights not expressly granted.

You also agree not to permit further distribution of the Redistributables by your end users except you may permit further redistribution of the Redistributables by your distributors to your end-user customers if your distributors only distribute the Redistributables in conjunction with, and as part of, the Licensee Software, you comply with all other terms of this EULA, and your distributors comply with all restrictions of this EULA that are applicable to you.

(b) If you use the Redistributables, then in addition to your compliance with the applicable distribution requirements described for the Redistributables, the following also applies. Your license rights to the Redistributables are conditioned upon your not (i) creating derivative works of the Redistributables in any manner that would cause the Redistributables in whole or in part to become subject to any of the terms of an Excluded License; or (ii) distributing the Redistributables (or derivative works thereof) in any manner that would cause the Redistributables to become subject to any of the terms of an Excluded License. "Excluded License" means any license that requires as a condition of use, modification and/or distribution of software subject to the Excluded License, that such software or other software combined and/or distributed with such software be (x) disclosed or distributed in source code form; (y) licensed for the purpose of making derivative works; or (z) redistributable at no charge.

4. RESERVATION OF RIGHTS AND OWNERSHIP. Microsoft reserves all rights not expressly granted to you in this EULA. The Software is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Software. The Software is licensed, not sold.

5. LIMITATIONS ON REVERSE ENGINEERING, DECOMPILATION, AND DISASSEMBLY. You may not reverse engineer, decompile, or disassemble the Software, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

6. NO RENTAL/COMMERCIAL HOSTING. You may not rent, lease, lend or provide commercial hosting services with the Software.

7. CONSENT TO USE OF DATA. You agree that Microsoft and its affiliates may collect and use technical information gathered as part of the product support services provided to you, if any, related to the Software. Microsoft may use this information solely to improve our products or to provide customized services or technologies to you and will not disclose this information in a form that personally identifies you.

8. ADDITIONAL SOFTWARE/SERVICES. This EULA applies to updates, supplements, add-on components, or Internet-based services components, of the Software that Microsoft may provide to you or make available to you after the date you obtain your initial copy of the Software, unless we provide other terms along with the update, supplement, add-on component, or Internet-based services component. Microsoft reserves the right to discontinue any Internet-based services provided to you or made available to you through the use of the Software.

9. UPGRADES/DOWNGRADES.

Upgrades. To use Software identified as an upgrade, you must first be licensed for the software identified by Microsoft as eligible for the upgrade. After upgrading, you may no longer use the software that formed the basis for your upgrade eligibility.

Downgrades. Instead of installing and using the Software, you may install and use one copy of an earlier version of the Software, provided that you completely remove such earlier version and install the original Software within a reasonable time. Your use of such earlier version shall be governed by this EULA, and your rights to use such earlier version shall terminate when you install the Software.

10. NOT FOR RESALE SOFTWARE. Software identified as "Not For Resale" or "NFR," may not be sold or otherwise transferred for value, or used for any purpose other than demonstration, test or evaluation.

11. ACADEMIC EDITION SOFTWARE. To use Software identified as "Academic Edition" or "AE," you must be a "Qualified Educational User." For qualification-related questions, please contact the Microsoft Sales Information Center/One Microsoft Way/Redmond, WA 98052-6399 or the Microsoft subsidiary serving your country.

12. EXPORT RESTRICTIONS. You acknowledge that the Software is subject to U.S. export jurisdiction. You agree to comply with all applicable international and national laws that apply to the Software, including the U.S. Export Administration Regulations, as well as end-user, end-use, and destination restrictions issued by U.S. and other governments. For additional information see <http://www.microsoft.com/exporting/>.

13. SOFTWARE TRANSFER. The initial user of the Software may make a one-time permanent transfer of this EULA and Software to another end user, provided the initial user retains no copies of the Software. This transfer must include all of the Software (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity). The transfer may not be an indirect transfer, such as a consignment. Prior to the transfer, the end user receiving the Software must agree to all the EULA terms.

14. TERMINATION. Without prejudice to any other rights, Microsoft may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the Software and all of its component parts.

15. DISCLAIMER OF WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE.

16. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL MICROSOFT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, PUNITIVE, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF PROFITS OR CONFIDENTIAL OR OTHER INFORMATION, FOR BUSINESS INTERRUPTION, FOR PERSONAL INJURY, FOR LOSS OF PRIVACY, FOR FAILURE TO MEET ANY DUTY INCLUDING OF GOOD FAITH OR OF REASONABLE CARE, FOR NEGLIGENCE, AND FOR ANY OTHER PECUNIARY OR OTHER LOSS WHATSOEVER) ARISING OUT OF OR IN ANY WAY RELATED TO THE USE OF OR INABILITY TO USE THE SOFTWARE, THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE, OR OTHERWISE UNDER OR IN CONNECTION WITH ANY PROVISION OF THIS EULA, EVEN IN THE EVENT OF THE FAULT, TORT (INCLUDING NEGLIGENCE), MISREPRESENTATION, STRICT LIABILITY, BREACH OF CONTRACT OR BREACH OF WARRANTY OF MICROSOFT OR ANY SUPPLIER, AND EVEN IF MICROSOFT OR ANY SUPPLIER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. LIMITATION OF LIABILITY AND REMEDIES. NOTWITHSTANDING ANY DAMAGES THAT YOU MIGHT INCUR FOR ANY REASON WHATSOEVER (INCLUDING, WITHOUT LIMITATION, ALL DAMAGES REFERENCED HEREIN AND ALL DIRECT OR GENERAL DAMAGES IN CONTRACT OR ANYTHING ELSE), THE ENTIRE LIABILITY OF MICROSOFT AND ANY OF ITS SUPPLIERS UNDER ANY PROVISION OF THIS EULA AND YOUR EXCLUSIVE REMEDY HEREUNDER SHALL BE LIMITED TO THE GREATER OF THE ACTUAL DAMAGES YOU INCUR IN REASONABLE RELIANCE ON THE SOFTWARE UP TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE OR US$5.00. THE FOREGOING LIMITATIONS, EXCLUSIONS AND DISCLAIMERS SHALL APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, EVEN IF ANY REMEDY FAILS ITS ESSENTIAL PURPOSE.

18. U.S. GOVERNMENT LICENSE RIGHTS. All Software provided to the U.S. Government pursuant to solicitations issued on or after December 1, 1995 is provided with the commercial license rights and restrictions described elsewhere herein. All Software provided to the U.S. Government pursuant to solicitations issued prior to December 1, 1995 is provided with "Restricted Rights" as provided for in FAR, 48 CFR 52.227-14 (JUNE 1987) or DFAR, 48 CFR 252.227-7013 (OCT 1988), as applicable.

19. APPLICABLE LAW. If you acquired this Software in the United States, this EULA is governed by the laws of the State of Washington. If you acquired this Software in Canada, unless expressly prohibited by local law, this EULA is governed by the laws in force in the Province of Ontario, Canada; and, in respect of any dispute which may arise hereunder, you consent to the jurisdiction of the federal and provincial courts sitting in Toronto, Ontario. If you acquired this Software in the European Union, Iceland, Norway, or Switzerland, then local law applies. If you acquired this Software in any other country, then local law may apply.

20. ENTIRE AGREEMENT; SEVERABILITY. This EULA (including any addendum or amendment to this EULA which is included with the Software) is the entire agreement between you and Microsoft relating to the Software and the support services (if any) and it supersedes all prior or contemporaneous oral or written communications, proposals and representations with respect to the Software or any other subject matter covered by this EULA. To the extent the terms of any Microsoft policies or programs for support services conflict with the terms of this EULA, the terms of this EULA shall control. If any provision of this EULA is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.

53. ANEXO: Licencia Visual C++ Toolkit 2003

Se añade aquí el contrato de uso EULA (End-User License Agreement) que se incluía con las herramientas en línea de comandos de Microsoft Visual C++ Toolkit 2003. Este contrato permite el uso del paquete para el desarrollo y distribución de software:

END-USER LICENSE AGREEMENT FOR MICROSOFT SOFTWARE

MICROSOFT VISUAL C++ TOOLKIT 2003

IMPORTANT-READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Microsoft Corporation ("Microsoft") for the Microsoft software that accompanies this EULA, which includes computer software and may include associated media, printed materials including best practices, white papers, templates, "online" or electronic documentation, and Internet-based services ("Software"). An amendment or addendum to this EULA may accompany the Software. YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA BY INSTALLING, COPYING, OR OTHERWISE USING THE SOFTWARE. IF YOU DO NOT AGREE, DO NOT INSTALL, COPY, OR USE THE SOFTWARE.

MICROSOFT Software LICENSE

1. GRANTS OF LICENSE. Microsoft grants you the rights described in this EULA provided that you comply with all terms and conditions of this EULA.

1.1 General License Grant. Microsoft grants to you as an individual, a personal, nonexclusive license to make and use copies of the Software (i) for your internal use; (ii) for designing, developing, testing and demonstrating your software product(s); and (iii) for evaluation of the Software.

1.2 Documentation. You may make and use an unlimited number of copies of any documentation, provided that such copies shall be used only for personal internal purposes and are not to be republished or distributed (either in hard copy or electronic form) beyond your premises except as otherwise specifically provided herein.

2. ADDITIONAL LICENSE RIGHTS -- REDISTRIBUTABLES. In addition to the rights granted in Section 1, certain portions of the Software, as described in this Section 2, are provided to you with additional license rights. These additional license rights are conditioned upon your compliance with the distribution requirements and license restrictions described in Section 3.

2.1 Sample Code. Microsoft grants you the right to use and modify the source code version of those portions of the Software identified as "Samples" in REDIST.TXT or elsewhere in the Software ("Sample Code") for the sole purposes of designing, developing, and testing your software product(s), and to reproduce and distribute the Sample Code along with any modifications thereof, in object and/or source code form. For applicable redistribution requirements for Sample Code, see Section 3.1 below.

2.2 Redistributable Code-General. Microsoft grants you a nonexclusive, royalty-free right to reproduce and distribute the object code form of any portion of the Software listed in REDIST.TXT ("Redistributable Code"). For general redistribution requirements for Redistributable Code, see Section 3.1, below.

3. LICENSE RESTRICTIONS -- DISTRIBUTION REQUIREMENTS. If you choose to exercise your rights under Section 2, any redistribution by you is subject to your compliance with the following terms.

3.1 If you are authorized and choose to redistribute Sample Code or Redistributable Code (collectively, the "Redistributables") as described in Section 2, you agree: (i) except as otherwise noted in Section 2.1 (Sample Code) to distribute the Redistributables only in object code form and in conjunction with and as a part of a software application product developed by you that adds significant and primary functionality to the Redistributables ("Licensee Software"); (ii) that the Redistributables only operate in conjunction with Microsoft Windows platforms; (iii) to distribute the Licensee Software containing the Redistributables pursuant to an end user license agreement (which may be "break-the-seal", "click-wrap" or signed), with terms no less protective than those contained in this EULA; (iv) not to use Microsoft's name, logo, or trademarks to market the Licensee Software; (v) to display your own valid copyright notice which shall be sufficient to protect Microsoft's copyright in the Software; (vi) not to remove or obscure any copyright, trademark or patent notices that appear on the Software as delivered to you; (vii) to indemnify, hold harmless, and defend Microsoft from and against any claims or lawsuits, including attorney's fees, that arise or result from the use or distribution of the Licensee Software; (viii) otherwise comply with the terms of this EULA; and (ix) agree that Microsoft reserves all rights not expressly granted.

You also agree not to permit further distribution of the Redistributables by your end users except you may permit further redistribution of the Redistributables by your distributors to your end-user customers if your distributors only distribute the

Redistributables in conjunction with, and as part of, the Licensee Software and you and your distributors comply with all other terms of this EULA.

3.2 If you use the Redistributables, then in addition to your compliance with the applicable distribution requirements described for the Redistributables, the following also applies. Your license rights to the Redistributables are conditioned upon your not (a) creating derivative works of the Redistributables in any manner that would cause the Redistributables in whole or in part to become subject to any of the terms of an Excluded License; and (b) distributing the Redistributables (or derivative works thereof) in any manner that would cause the Redistributables to become subject to any of the terms of an Excluded License. An "Excluded License" is any license which requires as a condition of use, modification and/or distribution of software subject to the Excluded License, that such software or other software combined and/or distributed with such software (x) be disclosed or distributed in source code form; (y) be licensed for the purpose of making derivative works; or (z) be redistributable at no charge.

4. RESERVATION OF RIGHTS AND OWNERSHIP. Microsoft reserves all rights not expressly granted to you in this EULA. The Software is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Software. The Software is licensed, not sold.

5. LIMITATIONS ON REVERSE ENGINEERING, DECOMPILATION, AND DISASSEMBLY. You may not reverse engineer, decompile, or disassemble the Software, except and only to the extent that such activity is expressly permitted by applicable law notwithstanding this limitation.

6. NO RENTAL/COMMERCIAL HOSTING. You may not rent, lease, lend, or provide commercial hosting services with the Software.

7. CONSENT TO USE OF DATA. You agree that Microsoft and its affiliates may collect and use technical information gathered as part of the product support services provided to you, if any, related to the Software. Microsoft may use this information solely to improve our products or to provide customized services or technologies to you and will not disclose this information in a form that personally identifies you.

8. LINKS TO THIRD PARTY SITES. You may link to third party sites through the use of the Software. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

9. ADDITIONAL SOFTWARE/SERVICES. This EULA applies to updates, supplements, add-on components, or Internet-based services components, of the Software that Microsoft may provide to you or make available to you after the date you obtain your initial copy of the Software, unless we provide other terms along with the update, supplement, add-on component, or Internet-based services component. Microsoft reserves the right to discontinue any Internet-based services provided to you or made available to you through the use of the Software.

10. EXPORT RESTRICTIONS. You acknowledge that the Software is subject to U.S. export jurisdiction. You agree to comply with all applicable international and national laws that apply to the Software, including the U.S. Export Administration Regulations, as well as end-user, end-use, and destination restrictions issued by U.S. and other governments. For additional information, see .

11. SOFTWARE TRANSFER. The initial user of the Software may make a one-time permanent transfer of this EULA and Software to another end user, provided the initial user retains no copies of the Software. This transfer must include all of the Software (including all component parts, the media and printed materials, any upgrades to this EULA, and, if applicable, the Certificate of Authenticity). The transfer may not be an indirect transfer, such as a consignment. Prior to the transfer, the end user receiving the Software must agree to all the EULA terms.

12. TERMINATION. Without prejudice to any other rights, Microsoft may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the Software and all of its component parts.

13. DISCLAIMER OF WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, MICROSOFT AND ITS SUPPLIERS PROVIDE THE SOFTWARE AND SUPPORT SERVICES (IF ANY) AS IS AND WITH ALL FAULTS, AND HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY (IF ANY) IMPLIED WARRANTIES, DUTIES OR CONDITIONS OF MERCHANTABILITY, OF FITNESS FOR A PARTICULAR PURPOSE, OF RELIABILITY OR AVAILABILITY, OF ACCURACY OR COMPLETENESS OF RESPONSES, OF RESULTS, OF WORKMANLIKE EFFORT, OF LACK OF VIRUSES, AND OF

LACK OF NEGLIGENCE, ALL WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT OR OTHER SERVICES, INFORMATION, SOFTWARE, AND RELATED CONTENT THROUGH THE SOFTWARE OR OTHERWISE ARISING OUT OF THE USE OF THE SOFTWARE. ALSO, THERE IS NO WARRANTY OR CONDITION OF TITLE, QUIET ENJOYMENT, QUIET POSSESSION, CORRESPONDENCE TO DESCRIPTION OR NON-INFRINGEMENT WITH REGARD TO THE SOFTWARE.

14. EXCLUSION OF INCIDENTAL, CONSEQUENTIAL AND CERTAIN OTHER DAMAGES. To the maximum extent permitted by applicable law, in no event shall Microsoft or its suppliers be liable for any special, incidental, punitive, indirect, or consequential damages whatsoever (including, but not limited to, damages for loss of profits or confidential or other information, for business interruption, for personal injury, for loss of privacy, for failure to meet any duty including of good faith or of reasonable care, for negligence, and for any other pecuniary or other loss whatsoever) arising out of or in any way related to the use of or inability to use the SOFTWARE, the provision of or failure to provide Support OR OTHER Services, informatIon, software, and related CONTENT through the software or otherwise arising out of the use of the software, or otherwise under or in connection with any provision of this EULA, even in the event of the fault, tort (including negligence), misrepresentation, strict liability, breach of contract or breach of warranty of Microsoft or any supplier, and even if Microsoft or any supplier has been advised of the possibility of such damages.

15. LIMITATION OF LIABILITY AND REMEDIES. NOTWITHSTANDING ANY DAMAGES THAT YOU MIGHT INCUR FOR ANY REASON WHATSOEVER (INCLUDING, WITHOUT LIMITATION, ALL DAMAGES REFERENCED HEREIN AND ALL DIRECT OR GENERAL DAMAGES IN CONTRACT OR ANYTHING ELSE), THE ENTIRE LIABILITY OF MICROSOFT AND ANY OF ITS SUPPLIERS UNDER ANY PROVISION OF THIS EULA AND YOUR EXCLUSIVE REMEDY HEREUNDER SHALL BE LIMITED TO THE GREATER OF THE ACTUAL DAMAGES YOU INCUR IN REASONABLE RELIANCE ON THE SOFTWARE UP TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE OR US$5.00. THE FOREGOING LIMITATIONS, EXCLUSIONS AND DISCLAIMERS SHALL APPLY TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, EVEN IF ANY REMEDY FAILS ITS ESSENTIAL PURPOSE.

16. APPLICABLE LAW. If you acquired this Software in the United States, this EULA is governed by the laws of the State of Washington. If you acquired this Software in Canada, unless expressly prohibited by local law, this EULA is governed by the laws in force in the Province of Ontario, Canada; and, in respect of any dispute which may arise hereunder, you consent to the jurisdiction of the federal and provincial courts sitting in Toronto, Ontario. If you acquired this Software in the European Union, Iceland, Norway, or Switzerland, then local law applies. If you acquired this Software in any other country, then local law may apply.

17. U.S. GOVERNMENT LICENSE RIGHTS. All Software provided to the U.S. Government pursuant to solicitations issued on or after December 1, 1995 is provided with the commercial license rights and restrictions described elsewhere herein. All Software provided to the U.S. Government pursuant to solicitations issued prior to December 1, 1995 is provided with "Restricted Rights" as provided for in FAR, 48 CFR 52.227-14 (JUNE 1987) or DFAR, 48 CFR 252.227-7013 (OCT 1988), as applicable.

18. ENTIRE AGREEMENT; SEVERABILITY. This EULA (including any addendum or amendment to this EULA which is included with the Software) are the entire agreement between you and Microsoft relating to the Software and the support services (if any) and they supersede all prior or contemporaneous oral or written communications, proposals and representations with respect to the Software or any other subject matter covered by this EULA. To the extent the terms of any Microsoft policies or programs for support services conflict with the terms of this EULA, the terms of this EULA shall control. If any provision of this EULA is held to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.
Should you have any questions concerning this EULA, or if you desire to contact Microsoft for any reason, please use the address information enclosed in this Software to contact the Microsoft subsidiary serving your country or visit Microsoft on the World Wide Web.

54. ANEXO: Licencia GPLv3

SHIFT Videogame es software libre: usted puede redistribuirlo y/o modificarlo bajo los términos de la Licencia Pública General GNU publicada por la Fundación para el Software Libre, ya sea la versión 3 de la Licencia, o (a su elección) cualquier versión posterior.

Este programa se distribuye con la esperanza de que sea útil, pero SIN GARANTÍA ALGUNA; ni siquiera la garantía implícita MERCANTIL o de APTITUD PARA UN PROPÓSITO DETERMINADO.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

“This License” refers to version 3 of the GNU General Public License.

“Copyright” also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

“The Program” refers to any copyrightable work licensed under this License. Each licensee is addressed as “you”. “Licensees” and “recipients” may be individuals or organizations.

To “modify” a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a “modified version” of the earlier work or a work “based on” the earlier work.

A “covered work” means either the unmodified Program or a work based on the Program.

To “propagate” a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's “contributor version”.

A contributor's “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in

connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

Madrid a 16 de marzo de 2008
http://shift.delblanco.es