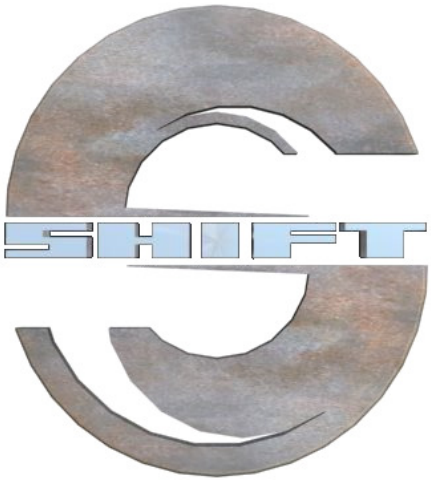

MOTOR DE UTILIDADES
SHIFT videogame

GAMELAB
UNIVERSIDAD DE OVIEDO

Alberto Carlos del Blanco Maraña
Madrid 29 de julio de 2007

Tabla de Contenidos

Lista de Figuras

1. Introducción

El presente documento detalla el sistema de utilidades diseñado para el videojuego SHIFT, como parte del proyecto fin de carrera realizado por Alberto Carlos del Blanco para la E.P.S.I.G. de la Universidad de Oviedo.

El motor de sonidos 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:

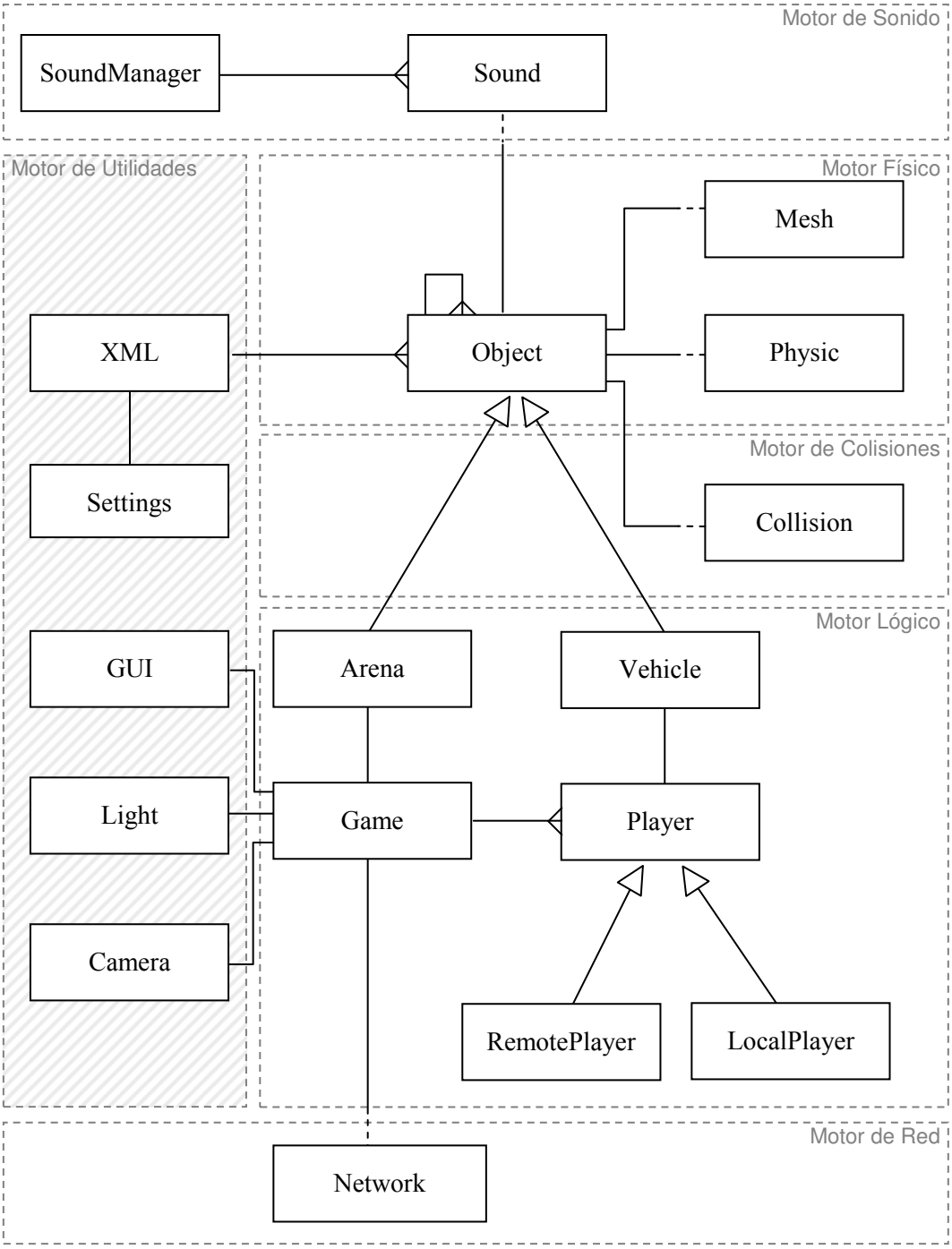

Imagen 1 : Límites del motor de utilidades

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

- **XML**
  Permite cargar o almacenar de forma persistente datos y estructuras de la aplicación con formato XML. Se usará este interfaz para el almacenamiento persistente, tanto para los modelos, la configuración, 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.

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

- **Light**
  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.

- **Camera**
  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.

2. Parámetros de la clase *XML*

Permite cargar o almacenar de forma persistente datos y estructuras de la aplicación con formato XML. Se usará este interfaz para el almacenamiento persistente, tanto para los modelos, la configuración, etc.

2.1. Funcionalidades de la clase XML

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 nodo, modificarlos, añadirlos, borrarlos, cambiar la estructura, guardar ficheros, leerlos, crearlos, modificarlos, etc.

Habrá una instancia global de esta clase que será utilizada por cada constructor dónde sea necesaria la lectura o almacenamiento de datos externos. Las clases identificadas son:

- **Clase Settings**, para cargar o almacenar la configuración de usuario en cada ejecución de la aplicación. De esta forma se asegura mantener las preferencias de usuario tras cada ejecución de la aplicación.
- **Clase Object**, para cargar la construcción de los objetos 3D y sus propiedades, ya sean arenas de juego, vehículos o el resto de objetos de escena.

A continuación veremos en detalle la definición de elementos y atributos de cada XML utilizado en la aplicación, tanto para la configuración de usuario como para la definición de objetos 3D.

2.2. XML para la configuración (*Settings*)

Se utiliza un único fichero XML dónde se almacenan todos los datos referentes a la configuración de usuario de la aplicación. Estos datos de usuario se manejan a través de la clase Settings que se verá 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, como por ejemplo:

- *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, tanto para la música como para los efectos de sonido.

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

El elemento *game* permite mantener los parámetros de usuario dentro de la lógica del juego, tales como el nombre del jugador, la IP del servidor o los últimos parámetros con los que se ha jugado.

```
<!ELEMENT game (player server multi1 multi2)>
<!ELEMENT player EMPTY>
<!ATTLIST player name CDATA #REQUIRED>
<!ELEMENT server EMPTY>
<!ATTLIST server name CDATA #REQUIRED>
<!ELEMENT multi1 EMPTY>
<!ATTLIST multi1 counter CDATA #REQUIRED>
<!ELEMENT multi2 EMPTY>
<!ATTLIST multi2 counter CDATA #REQUIRED>
```

2.3. XML para modelos 3D (*Objects*)

La lectura de ficheros XML también se realiza en el constructor de la clase Object, 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.

Según ya se ha visto, existe una jerarquía de objetos que queda definida de forma casi directa con XML. En resumen los objetos pueden ser:

- **Arenas**, que son los escenarios dónde se desarrolla el juego
- **Vehículos**, que representan a cada uno de los jugadores
- **Objetos**, que componen a los anteriores, arenas o vehículos

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

2.3.1. Objetos (incluidos vehículos y arenas)

Para todos los objetos podemos definir un tipo, nombre y descripción:

```
<!ELEMENT object (mesh?, position?, rotation?, scale?,
                  physic?, collision?, sound?, object)>
<!ATTLIST object type (arena | vehicle) #REQUIRED>
<!ATTLIST object name CDATA #REQUIRED>
<!ATTLIST object description CDATA #REQUIRED>
```

La geometría, que se carga mediante un fichero .X externo (formato DirectX para definir vértices, normales y texturas)

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

La posición X, Y, Z en el espacio:

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

La rotación en los ejes X, Y, Z:

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

El escalado del objeto en los ejes X, Y, Z:

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

Los parámetros físicos de velocidad, aceleración y fricción, tanto lineales (con direcciones) como angulares (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>
```

Si el objeto va a ser colisionable o no. El área de colisión se tomará de la geometría cargada del fichero .X externo:

```
<!ELEMENT collision EMPTY>
<!ELEMENT sound EMPTY>
```

Sonido externo .WAV asociado al objeto:

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

2.3.2. Vehículos

Además de los anteriores, para los vehículos podemos definir varios nodos XML que especializan la descripción del vehículo dentro del videojuego.

Tenemos un nodo de características, dónde podemos indicar atributos como la capacidad, vida, velocidad máxima, aceleración, y diferentes parámetros que definen cualidades físicas del vehículo en escena:

```
<!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>
```

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

2.3.3. Arenas

Al igual que en los vehículos, con las arenas tenemos un nodo adicional que determina el firmamento a usar. En este caso el tratamiento de este objeto de la arena tiene un tratamiento diferente al resto.

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

3. 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, estilos de juego, etc.

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:

- *music* para la música
- *sound* para los efectos de sonido.

El *juego* permite mantener:

- *player_name* nombre del jugador
- *server_name* IP del servidor
- *counter* Último contador escogido para los diferentes modos de juego

4. 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 diseño del interfaz se realiza ofreciendo diferentes opciones tal y como se comentó en el análisis inicial y tal y cómo se detallará a continuación.

4.1. Diagrama de navegación

En principio vamos a detallar el diagrama de navegación de pantallas y más adelante se describirá cada una por separado.

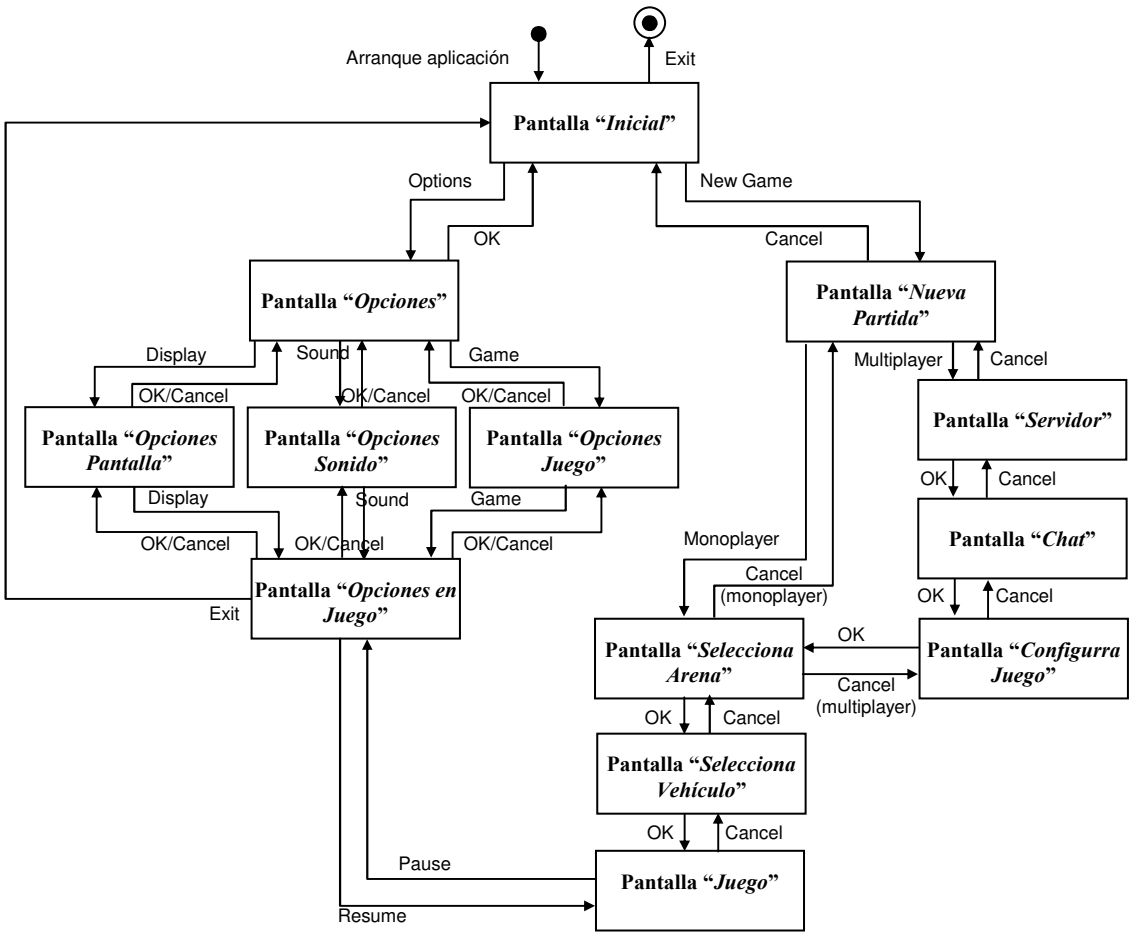

Imagen 2 : Diagrama de pantallas del GUI

4.2. Pantallas

4.2.1. Pantalla "*Inicial*"

En la pantalla inicial aparecen las tres opciones disponibles: iniciar una nueva partida, opciones y salir.

Imagen 3 : Pantalla "*Inicial*"

4.2.2. Pantalla "*Nueva Partida*"

Permite asignar un nombre al jugador y seleccionar entre partida monojugador o multijugador

Imagen 4 : Pantalla "*Nueva Partida*"

4.2.3. Pantalla "*Servidor*"

Para escoger un servidor de una partida multijugador o crear uno nuevo

Imagen 5 : Pantalla "*Servidor*"

4.2.4. Pantalla "*Chat*"

Para hablar con otros usuarios que se van conectando a la partida multijugador

Imagen 6 : Pantalla "*Chat*"

4.2.5. Pantalla "*Configurar Juego*"

Para configurar los aspectos propios del tipo de juego escogido

Imagen 7 : Pantalla "*Configurar Juego*"

4.2.6. Pantalla "*Seleccionar Arena*"

Permite seleccionar una arena entre las múltiples disponibles:

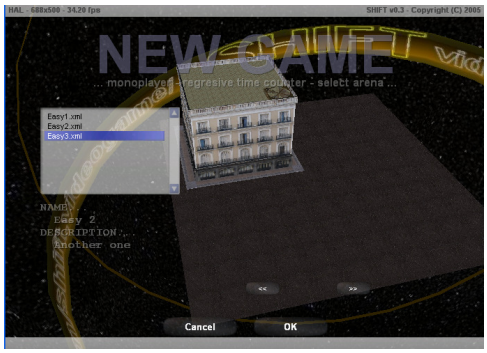

Imagen 8 : Pantalla "*Seleccionar Arena*"

4.2.7. Pantalla "*Seleccionar Vehículo*"

Permite seleccionar un vehículo entre los múltiples disponibles:

Imagen 9 : Pantalla "*Seleccionar Vehículo*"

4.2.8. Pantalla "*Juego*"

GUI durante el juego, en la parte inferior se permite pausar el juego, en modo multijugador esto no pausará la partida

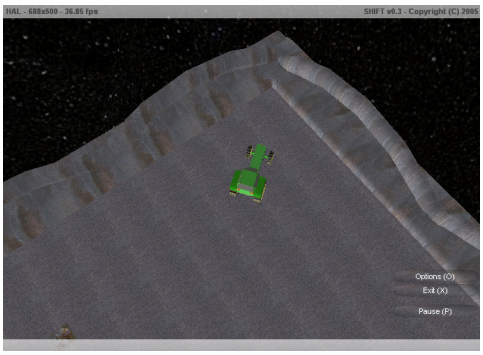

Imagen 10 : Pantalla "*Juego*"

4.2.9. Pantallas "*Opciones*" y "*Opciones en Juego*"

Pantallas de opciones desde el menú principal y desde el propio juego

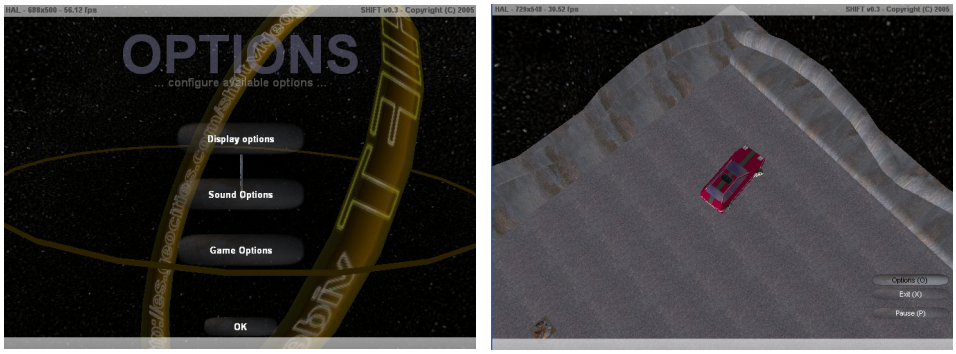

Imagen 11 : Pantallas "*Opciones*" y "*Opciones en Juego*"

4.2.10. Pantalla "*Opciones de Pantalla*"

Para configurar los aspectos propios del dispositivo de video:

Imagen 12 : Pantalla "*Opciones de Pantalla*"

4.2.11. Pantalla "*Opciones de Sonido*"

Para configurar los aspectos propios del sonidos y música

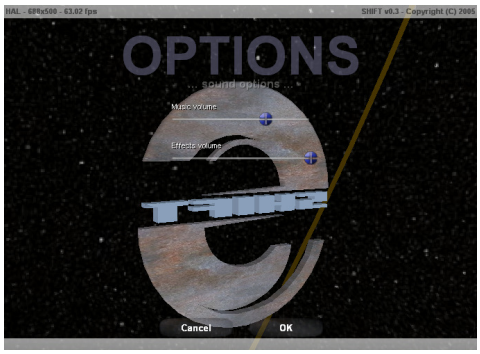

Imagen 13 : Pantalla "*Opciones de Sonido*"

4.2.12. Pantalla "*Opciones de Juego*"

Para configurar los aspectos propios de la lógica del videojuego

Imagen 14 : Pantalla "*Opciones de Juego*"

4.3. Manejo del GUI

Los controles para el GUI y para el juego son los típicos de cualquier juego de PC. Además, considerando las pocas operaciones que se pueden realizar, no se considera necesario permitir la configuración personal de los controles, por lo que serán fijos para todas las partidas.

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 por las opciones situándonos sobre ellas, y seleccionar una pulsando el botón izquierdo. Con el teclado se puede navegar por las opciones con las teclas del cursor, y seleccionar una pulsando intro.

4.4. Manejo del juego

Durante el juego sólo podremos tomar el control del coche por medio del teclado. Los controles son los siguientes:

- **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 dos ruedas traseras, con los que el coche se vuelve más inestable, pero permite ladear el coche rápidamente.
- **Tecla 'Z'** Para colocar una mina
- **Tecla 'X'** Para lanzar un cohete
- **ESC** Para pausar el juego y lanzar el menú de juego.

5. Parámetros de la clase *Light*

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 cambos de escena en la aplicación. Sin embargo no hay muchas más características añadidas al objeto, que simplemente ofrece un interfaz estándar para el tratamiento de la luz como un objeto más en el motor gráfico de la aplicación.

6. Parámetros de la clase *Camera*

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 de hacen focalizando un objeto, que en nuestro caso siempre será un vehículo y moviendo la cámara de manera retardada al movimiento del objeto (para crear un efecto muelle en el seguimiento de la cámara).

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

La cámara se puede situar a tres alturas diferentes, la primera desde el nivel del suelo.

Imagen 15 : Cámara con vista a nivel de suelo

La segunda a nivel medio:

Imagen 16 : Cámara con vista media

Y la tercera desde un nivel superior:

Imagen 17 : Cámara con vista superior

Sin embargo las tres perspectivas comparten el mismo algoritmo de seguimiento, cambiando únicamente la altura final de la cámara.

Alberto Carlos del Blanco Maraña
Madrid 29 de julio de 2007