



PROGRAMACIÓN
SHIFT videogame

GAMELAB
UNIVERSIDAD DE OVIEDO

Tabla de Contenidos

1. Introducción	3
2. Tecnologías Utilizadas	3
3. Estructura de Directorios.....	4
4. Documentación de clases	6
4.1. Referencia de la Clase C3DCamera	6
4.2. Referencia de la Clase C3DCollision.....	10
4.3. Referencia de la Clase C3DLight.....	12
4.4. Referencia de la Clase C3DMesh.....	16
4.5. Referencia de la Clase C3DObjArena.....	18
4.6. Referencia de la Clase C3DObject.....	20
4.7. Referencia de la Clase C3DObjVehicle	25
4.8. Referencia de la Clase C3DPhysics	31
4.9. Referencia de la Clase C3DSound	34
4.10. Referencia de la Clase C3DSoundMng.....	36
4.11. Referencia de la Clase CGame.....	38
4.12. Referencia de la Clase CGUI	39
4.13. Referencia de la Clase CMenu	44
4.14. Referencia de la Clase CPlayer	47
4.15. Referencia de la Clase CPlayerLocal	51
4.16. Referencia de la Clase CPlayerRemote.....	52
4.17. Referencia de la Clase CSettings	53
4.18. Referencia de la Clase CXMLIO	55
4.19. Referencia de la Estructura SCustomVertex	57
4.20. Referencia de la Estructura SDataGame	58
4.21. Referencia de la Estructura SDataSettings.....	59
4.22. Referencia de la Estructura SMeshEntry	60
5. Documentación de archivos.....	62
5.1. Referencia del Archivo 3DCamera.h	62
5.2. Referencia del Archivo 3DCollision.h.....	63
5.3. Referencia del Archivo 3DLight.h.....	63
5.4. Referencia del Archivo 3DMesh.h.....	64
5.5. Referencia del Archivo 3DObjArena.h.....	64
5.6. Referencia del Archivo 3DObject.h.....	65
5.7. Referencia del Archivo 3DObjVehicle.h	65
5.8. Referencia del Archivo 3DPhysics.h	65
5.9. Referencia del Archivo 3DSound.h	66
5.10. Referencia del Archivo 3DSoundMng.h	66
5.11. Referencia del Archivo Game.h	67
5.12. Referencia del Archivo GUI.h.....	68
5.13. Referencia del Archivo Main.h	69
5.14. Referencia del Archivo Menu.h	70
5.15. Referencia del Archivo Player.h	71
5.16. Referencia del Archivo PlayerLocal.h	71
5.17. Referencia del Archivo PlayerRemote.h.....	71
5.18. Referencia del Archivo Settings.h.....	72
5.19. Referencia del Archivo XMLIO.h.....	72

1. Introducción

En el presente documento se detallarán los aspectos más relevantes de la programación del 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.

Lo que aquí se incluye es la documentación del código auto generada mediante la herramienta Doxygen. Todo el código se encuentra comentado para que sea procesado por esta herramienta, lo que ofrece gran versatilidad de cara a gestiones posteriores del código.

2. Tecnologías Utilizadas

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

- Lenguaje C++, dónde todo el código sigue una estricta orientación a objetos impuesta por la propia naturaleza del videojuego. Esto es debido a que toda la jerarquía de clases diseñada modela de forma muy simple 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 documento de ANEXOS, 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. Su licencia (EULA) que permite el desarrollo y distribución de software puede verse también en el documento de ANEXOS.

3. 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 **3DObjVehicle.cpp**
 - archivo **3DObjVehicle.h**
- Para manejar los vehículos de la escena.*
-
- archivo **3DPhysics.cpp**
 - archivo **3DPhysics.h**
- Para manejar la física de los objetos de la escena.*

- 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 **PlayerRemote.cpp**
- archivo **PlayerRemote.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.

4. Documentación de clases

4.1. 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 (D3DXVECTOR3 dPosition, float dSpeed)**
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.
- **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 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.
- **int m_iType**
Tipo de cámara.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3Dcamera.h**
- **3Dcamera.cpp**

4.2. Referencia de la Clase C3DCollision

```
#include <3DCollision.h>
```

Métodos públicos

- **bool initialize** (**C3DObject** *Object, float dmass, **D3DXVECTOR4** *dv)
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.
- **void setV** (**D3DXVECTOR4** *dv)
Establece los cuatro vértices que determinan el área de colisión 2D (ejes X,Z).
- **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 transformación actual del objeto.

Métodos protegidos

- **C3DCollision** ()
Constructor privado, sólo se podrá crear desde una instancia 3DObject.
- **~C3DCollision** ()
Destructor.
- **bool CheckCollision** (**C3DCollision** *Collision)
Función para comprobar colisiones de la instancia que invoca respecto otra instancia.

Atributos protegidos

- **bool m_bInit**
Flag que indica si el objeto está inicializado.
- **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).
- **D3DXMATRIX m_transInv**
Inversa de la matriz de transformacion del objeto.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DCollision.h**
- **3DCollision.cpp**

4.3. 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, bool Atomic=false)**
Apagar las luces lentamente.
- **void FallIn (float Spd, bool Atomic=false)**
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 atenuación0.
- void **setAttenuation1** (float v)
Establece la atenuación1.
- void **setAttenuation2** (float v)
Establece la atenuación2.
- 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.
- **bool m_bFallAtomic**
Indica si el fall actual es atómico.
- **float m_FallSpd**
Velocidad del Falloff o FallIn actual.
- **D3DCOLORVALUE m_oldDiffuse**
Luz difusa antes de apagar las luces.
- **D3DCOLORVALUE m_oldSpecular**
Luz especular antes de apagar las luces.
- **D3DCOLORVALUE m_oldAmbient**
Luz ambiental antes de apagar las luces.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DLight.h**
- **3DLight.cpp**

4.4. Referencia de la Clase C3DMesh

```
#include <C3DMesh.h>
```

Métodos públicos

- **bool initialize (C3DObject *dObject, LPCWSTR MeshPath)**
Inicializa el modelo gráfico.
- **void render ()**
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.
- **LPD3DXMESH m_pMesh**
Geometria.
- **std::vector< D3DMATERIAL9 > * m_aMtrls**
Lista de Materiales.
- **std::vector< IDirect3DTexture9 * > * m_aTextures**
Lista de Texturas.

- **LPWSTR m_awMeshPath**
Path dónde se encuentra el modelo gráfico.
- **SMeshEntry * m_pOriginal**
Puntero al modelo abierto en memoria.
- **C3DObject * m_pObject**
Objeto asociado a este modelo gráfico.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DMesh.h**
- 3DMesh.cpp

4.5. Referencia de la Clase C3DObjArena

```
#include <C3DObjArena.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 setEnabledSky (bool enabled)**
Función para cambiar el estado de procesamiento del cielo de la arena, activado/desactivado.
- **bool getEnabledSky (void)**
Función tomar el estado de procesamiento del cielo de la arena, activado/desactivado.
- **LPWSTR getName (void)**
Función observadora del nombre de la arena.

- **LPWSTR getDescription** (void)
Función observadora de la descripción de la arena.
- **C3DObject * getSky** (void)
Función observadora del cielo de la arena.

Atributos privados

- **LPWSTR m_arenaName**
Nombre de la arena.
- **LPWSTR m_arenaDescription**
Descripción textual de la arena.
- **C3DObject * m_pSky**
Puntero al cielo de la arena.

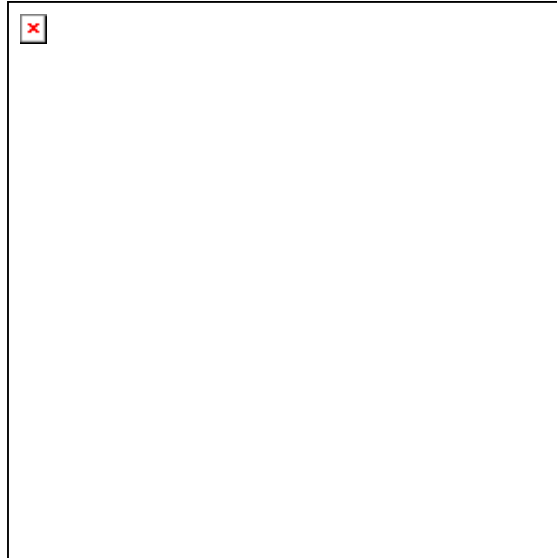
La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DObjArena.h**
- **3DObjArena.cpp**

4.6. 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.
- **C3DSound * getSound** (void)
Función observadora del sonido asociado.
- **std::vector< C3DObject * > * getObjects** (void)
Función observadora de la lista de objetos que componen el objeto.
- **D3DXMATRIX * getTransformation** (void)
Obtiene la transformación actual del objeto.
- **D3DXMATRIX * calculateTransInv** (D3DXMATRIX *result)
Toma o calcula la inversa de la transformación actual del objeto.
- **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.
- **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.

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 transformación final neta.
- **D3DXVECTOR3 * m_pScale**
Vector del escalado.
- **D3DXVECTOR3 * m_pRotation**
Vector de la rotación.
- **D3DXVECTOR3 * m_pPosition**
Vector de la posición.
- **bool m_bModified**

Indica si hay que refrescar la matriz actual.

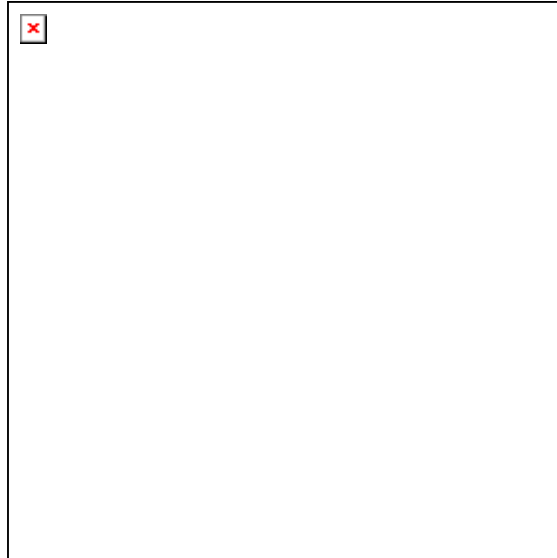
La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DObject.h**
- 3DObject.cpp

4.7. 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)**
Inicializa el vehículo.
- **void update ()**
Actualiza el vehículo.
- **D3DXMATRIX * calculateTransInv (D3DXMATRIX *result)**
Toma o calcula la inversa de la transformación actual del 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.
- 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.
- **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.
- 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 **getAccelerate** (void)
Función observadora del estado del acelerador.
- void **setAccelerate** (bool Value)
Función para activar o desactivar el acelerador.
- void **playSkid** (bool enabled)
Para activar el sonido de derrape del coche.
- void **stopSkid** (void)
Para desactivar el sonido de derrape del coche.
- void **playCrash** (bool enabled)
Para activar el sonido de derrape del coche.

- void **stopCrash** (void)
Para desactivar el sonido de derrape 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

- LPWSTR **m_awsName**
Nombre del vehículo.
- LPWSTR **m_awsDescription**
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.
- 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.
- **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.
- **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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DObjVehicle.h**
- **3DObjVehicle.cpp**

4.8. Referencia de la Clase C3DPhysics

```
#include <3DPhysics.h>
```

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 **addLFriction** (float Val)
Añade una fricción lineal a la existente.
- 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_pAcceleration**
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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DPhysics.h**
- **3DPhysics.cpp**

4.9. 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 Despasa la reproducción del sonido.
- void **Stop** (void)
Detiene la reproducción del sonido.
- void **setVolume** (long IVolume)
Cambia el volumen de reproducción del sonido.
- void **setFrequency** (long IFrequency)
Cambia la frecuencia de reproducción del sonido.
- long **getFrequency** (void)
Toma la frecuencia actual de reproducción del sonido.
- long **getInitFrequency** (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 IFrequency, long IVolume)
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**

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DSound.h**
- 3DSound.cpp

4.10. 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 Despasa 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.
- **C3DSoundManager * m_pSoundManager**
Manejador principal de sonidos 3D.
- **LPDIRECTSOUND3DLISTENER m_pDSLlistener**
Entorno 3D para el sonido.

- `long m_IVolume`
Volumen actual de los sonidos.
- `std::vector< C3DSound *> m_aSounds`
Lista de sonidos 3D manejados.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **3DSoundMng.h**
- 3DSoundMng.cpp

4.11. 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 * getLocalPlayer (void)**
Función observadora del jugador local.
- **C3DObjArena * getArena (void)**
Función observadora de la arena del 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_pLPlayer**
Puntero al jugador local.
- **std::vector< CPlayerRemote * > m_aRPlayers**
Vector con los jugadores remotos del juego, indexado por la IP de cada uno.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **Game.h**
- **Game.cpp**

4.12. 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=-1)
Establece la arena con el índice dado, cargando todas antes si fuese necesario.
- **bool setVehicle** (int dIndex=-1)
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 setStatistics** (bool dType)
Muestra u oculta las estadísticas.
- **bool getStatistics** (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.
- **void setIsServer** (bool dServer)
Función para establecer el tipo de aplicación como cliente o servidor.
- **bool getIsServer** (void)
Comprueba si la aplicación actúa como servidor o cliente.
- **void setMaxCountGame** (int dMax)
Funcion para establecer el contador máximo del juego.
- **int getMaxCountGame** (void)
Funcion observadora del máximo valor para el contador del 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 getGaming** (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_bStatistics**
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.
- **int m_iMaxCountGame**
Indica el máximo tamaño para el contador del juego.
- **bool m_bPaused**
Indica si estamos o no en pausa.
- **bool m_bGamming**
Indica si estamos o no jugando (pausados o no).

- **bool m_bIsServer**
Indica si somos o no servidores.
- **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_iVehicle**
Índice del vehículo 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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **GUI.h**
- **GUI.cpp**

4.13. 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** * **getLight** (void)
Retorna la luz utilizada.
- **C3DObjArena** * **getArena** ()
Retorna la arena actualmente seleccionada.
- **C3DObjVehicle** * **getVehicle** ()
Retorna el vehículo actualmente seleccionado.
- **CGame** * **getGame** (void)
Retorna el juego actual.

- **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_pVehicle**
Vehículo seleccionado.
- **CGame * m_pGame**
Juego actual.
- **SDataGame m_sDataGame**
Información sobre el juego (tipo, jugadores y arena).
- **double 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.

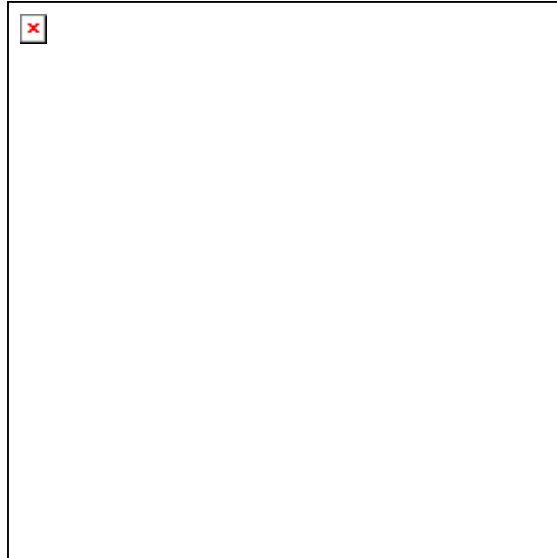
La documentación para esta clase fué generada a partir de los siguientes archivos:

- **Menu.h**
- **Menu.cpp**

4.14. 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.
- **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_awsName**
Nombre del jugador.
- C3DObjVehicle * **m_pVehicle**
Puntero al vehículo del jugador.
- 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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **Player.h**
- **Player.cpp**

4.15. 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)
Inicializa el jugador local.
- **void update ()**
Actualiza el jugador local.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **PlayerLocal.h**
- **PlayerLocal.cpp**

4.16. Referencia de la Clase CPlayerRemote

```
#include <PlayerRemote.h>
```

Diagrama de herencias de CPlayerRemote



Métodos públicos

- **CPlayerRemote ()**
Constructor.
- **~CPlayerRemote ()**
Destructor.
- **bool initialize** (LPCWSTR dIP, LPCWSTR dName, LPCWSTR dPathVehicle, **C3DObject** *dParent)
Inicializa el jugador remoto.
- **void update ()**
Actualiza el jugador remoto.

Atributos privados

- **LPWSTR m_awIP**
Nombre del jugador.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **PlayerRemote.h**
- **PlayerRemote.cpp**

4.17. 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 getHeight** (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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **Settings.h**
- **Settings.cpp**

4.18. 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.

La documentación para esta clase fué generada a partir de los siguientes archivos:

- **XMLIO.h**
- **XMLIO.cpp**

4.19. Referencia de la Estructura SCustomVertex

```
#include <3DMesh.h>
```

Atributos públicos

- float **x**
Coordenada X de la posición del vértice.
- float **y**
Coordenada Y de la posición del vértice.
- float **z**
Coordenada Z de la posición del vértice.
- float **nx**
Coordenada X de la normal al vértice.
- float **ny**
Coordenada Y de la normal al vértice.
- float **nz**
Coordenada Z de la normal al vértice.
- float **tu**
Escalado U de la textura para el vértice.
- float **tv**
Escalado V de la textura para el vértice.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **3DMesh.h**

4.20. 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_awLVehicle**
Path del vehículo del jugador local.
- **std::basic_string< wchar_t > m_awLName**
Nombre del jugador local.
- **std::vector< std::basic_string< wchar_t > > m_aRIPs**
Vector con los nombres de los jugadores remotos.
- **std::vector< std::basic_string< wchar_t > > m_aRVehicles**
Vector con los path de los vehículos de los jugadores remotos.
- **std::vector< std::basic_string< wchar_t > > m_aRNames**
Vector con los nombres de los jugadores remotos.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **Game.h**

4.21. 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_iMusic**
Volumen de la música.
- **int m_iEffects**
Volumen de los efectos sonoros.
- **WCHAR m_awNomPlayer [MAX_STRING+1]**
Nombre por defecto para el jugador local.
- **WCHAR m_awNomServer [MAX_STRING+1]**
Dirección por defecto del servidor remoto.
- **int m_iMulti1**
Contador por defecto para el modo multijugador 1.
- **int m_iMulti2**
Contador por defecto para el modo multijugador 2.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **Settings.h**

4.22. Referencia de la Estructura SMeshEntry

```
#include <3DMesh.h>
```

Atributos públicos

- **C3DMesh * Mesh**
Puntero a la geometría del objeto.

- unsigned short **num**
Número de referencias que tiene la geometría del objeto.

La documentación para esta estructura fué generada a partir del siguiente archivo:

- **3DMesh.h**



5. Documentación de archivos

5.1. 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_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_MIN_ANGLE 0.2f`
Mínima diferencia de la cámara de persecución por la que ya no se rotará.



5.2. 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_L_COEF** 0.6f
Coefficiente de transferencia de cantidad de movimiento lineal ante colisiones.
- #define **COLLISION_A_COEF** 0.8f
Coefficiente de transferencia de cantidad de movimiento angular ante colisiones.

5.3. 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.



5.4. 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.
- struct **SCustomVertex**
Estructura para almacenar un vértice de una geometría de objeto en memoria.
- class **C3DMesh**
Clase para manejar los modelos gráficos de la escena.

5.5. Referencia del Archivo 3DObjArena.h

Descripción detallada

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

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

Clases

- 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.



5.6. 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 "XMLIO.h"
```

Clases

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

5.7. Referencia del Archivo 3DObjVehicle.h

Descripción detallada

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

```
#include "3DObject.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.

5.8. 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.



5.9. 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** 1.5f
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** 5000.0f
Distancia máxima de decaimiento del sonido 3D.

5.10. 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.



5.11. Referencia del Archivo Game.h

Descripción detallada

```
#include <vector>
#include <string>
#include "PlayerLocal.h"
#include "PlayerRemote.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_MONO_1** 1
Juego monojugador.
- #define **GAME_MULTI_1** 2
Juego multijugador por contador de objetivos.
- #define **GAME_MULTI_2** 3
Juego multijugador por contador de muertes.
- #define **GAME_MULTI_3** 4
Juego multijugador partida a muerte.



5.12. 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.



5.13. 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 - Alberto Carlos del Blanco"`
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.



5.14. 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_VEHICLE** 3
Tipo de menú para seleccionar vehículo.
- #define **MENU_TYPE_GAME** 4
Tipo de menú para el modo juego.
- #define **MENU_TYPE_PAUSE** 5
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\\vehiculos\\"
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 **MAIN_ANIM_SEG** 7.0f
Periodo de tiempo para cambio de animación.



5.15. 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"
```

Clases

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

5.16. 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.*

5.17. Referencia del Archivo PlayerRemote.h

Descripción detallada

Este módulo incorpora las operaciones para manejar un vehículo por medio de los datos recibidos por la red.

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

Clases

- class **CPlayerRemote**
*Clase hija de **CPlayer** para manejar los jugadores de tipo remoto.*



5.18. 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.

5.19. 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.

Alberto Carlos del Blanco Maraña
Madrid 01 de julio de 2005