ETSIDIlib - *Librería para facilitar los juegos de la asignatura Informática Industrial.*

prof.: Miguel Hernando

http://mhernando.github.com/ETSIDIlib

ETSIDIlib es una librería con un concepto de diseño de facilitar operaciones de por sí un poco tediosas como es la carga de texturas en formato png o la reproducción de sonidos o uso de fuentes.

Por tanto, más que eficiencia, se busca una interfaz cómoda. De ahí que se abuse un poco del uso de Singletons que por medio de maps facilitan la carga aparentemente al vuelo de estos recursos, pero que en verdad son intermamente "cacheados".

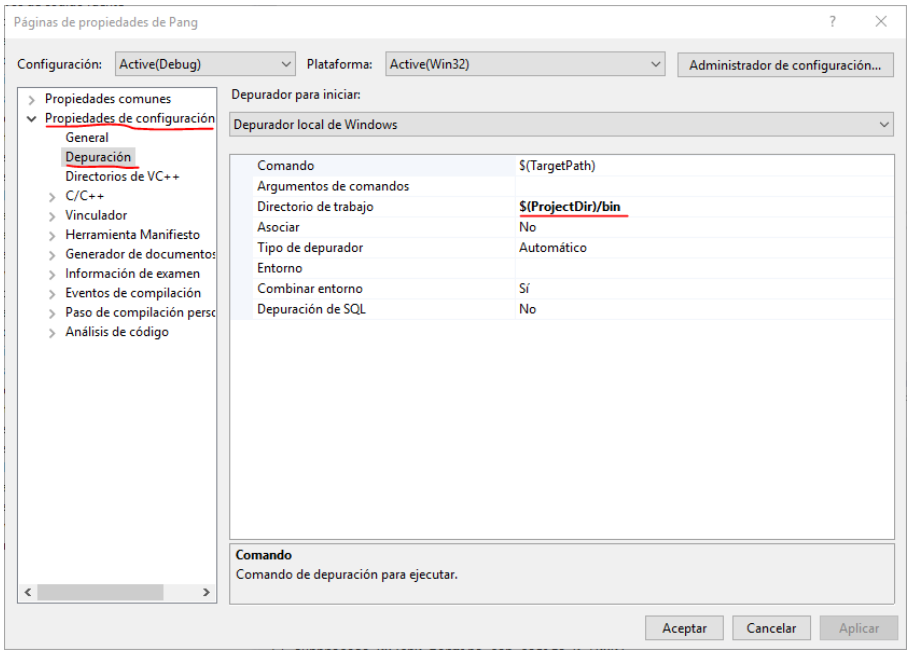
MODO DE USO EN VISUAL STUDIO

Se propone incluir una carpeta lib, y una carpeta bin en el proyecto.

En la carpeta lib, copiar ETSIDIlib.h y ETSIDIlib.lib, y en la carpeta bin copiar ETSIDIlib.dll y FmodL.dll.

En el repositorio se han dejado generadas las versiones para Visual Studio x86 (32bits).

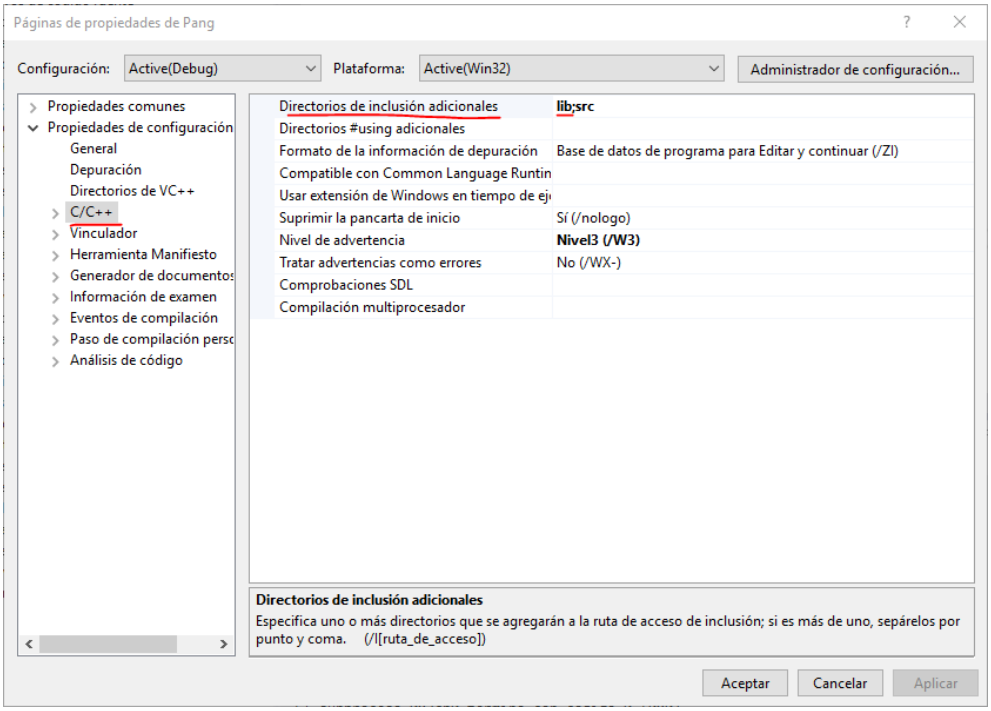
En las propiedades del proyecto se propone (se puede hacer de muchas otras formas) modificar las propiedades de la siguiente forma. Indicar que el directorio de salida y de trabajo es $(ProjectDir)/bin en vez del valor por defecto que es $(ProyectDir):



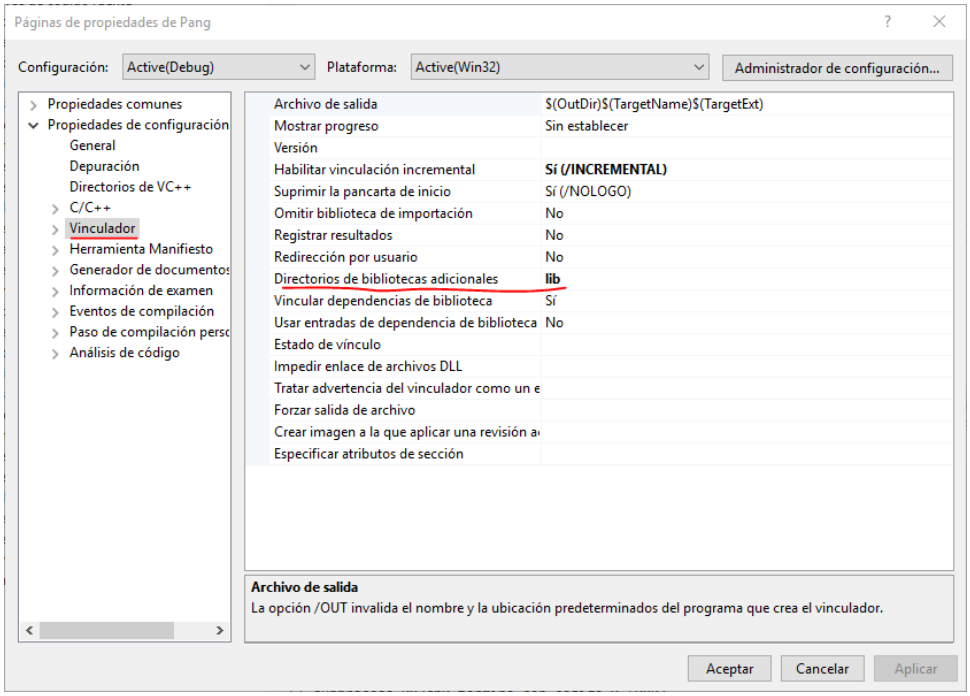
Con esto indicamos que el proyecto utilizará durante la ejecución este directorio como base a la hora de lanzarlo desde visual. De esta forma el ejecutable podrá encontrar los ficheros de imágenes o de sonidos ubicados en este punto así como las dos dll requeridas que ya hemos copiado en el directorio.

Además de eso es necesario informar al proyecto de donde se encuentra el código de enlazado (lib) así como la cabecera de la librería (ETSIDI.h)

En las propiedades de C/C++ agregar lib a los Directorios de inclusión adicionales:



Y por último en el Vinculador, agregar lib a los directorios de bibliotecas adicionales:



Con esto ya tenemos el marco minimo para funcionar. En cualquier lugar bastará con escribir #include “ETSIDI.H” para poder disponer de las funcionalidades siguientes:

SONIDO

Solo se incluyen 3 funciones globales, que a continuación se pone como se usan a modo de ejemplo:

```
ETSIDI::play("mis_sonidos/bang.wav");
ETSIDI::playMusica("mis_musicas/fondo.mp3", true);
ETSIDI::stopMusica();
```

- `void play(const char *  soundPath)`
  Reproduce por el primer canal libre disponible el sonido introducido por el fichero. El sonido solo se cargará la primera vez quedándose en memoria disponible por si se utiliza más veces. El ejecutarse cada sonido en un canal será posible superponer sonidos distintos o incluso el mismo sonido varias veces mientras no se agote el número de canales disponibles en el sistema. Esta función es la habitual para sonidos de bombas, explosiones, pasos, impactos… etc… sonidos cortos y sin comprimir (normalmente wav). Admite multiples formatos pero este es el que se recomienda.
- `void playMusica(const char *  soundPath, bool repite=false)`
  Es util para tener sonido de ambiente o de fondo que apenas reacciona a lo que va ocurriendo. Habitualmente es un mp3 dado que pueden ser ficheros demasiado grandes si no están comprimidos. Admite un parámetro adicional opcional que es si hay o no repetición contínua de la música. Solo se puede poner una música cada vez (tiene asignado un canal)
- `void stopMusica()`
  Para la reproducción de la música.

NUMEROS ALEATORIOS

Simplifica la generación de números aleatorios con una serie de funciones globales muy sencillas. Internamente hace uso del reloj del ordenador para usar una semilla del algoritmo que evite repeticiones en diversas ejecuciones.

```
valor=ETSIDI::lanzaDado(); //valor valdra entre 0.0 y 1.0 double
valor=ETSIDI::lanzaDado(10.0); //valor valdra entre 0.0 y 10.0 double
valor=ETSIDI::lanzaDado(10); //valor valdra entre 0 y 10 entero
valor=ETSIDI::lanzaMoneda(); //valor valdra o true o false
```

- `double lanzaDado(double max=1.0, double min=0.0F)`
  Genera un número real aleatorio entre max y min
- `int lanzaDado(int max, int min=1)`
  Genera un número entero aleatorio entre max y min
- `bool lanzaMoneda()`
  Genera un número aleatorio binario

TEXTURAS

Básicamente consiste en una caché externa a OPENGL de forma que leerá los ficheros png con canal o sin canal alpha de forma que sean entendibles por OPENGL. Es la función básica para

cargar texturas y después usarlas por medio del identificador interno de openGL. Consiste en una sola función:

```
GLtextura mitextura=ETSIDI::getTexture(“imágenes/fondo.png”);
```

- GLTexture getTexture(const char * texturePath)
  La función retorna una estructura que contiene tres campos: ancho y alto de la imagen, y un identificador que es el identificador de la textura cargada en el Sistema gráfico. Si la función es capaz de leer la textura pero no ha podido generarla en OpenGL el campo id contendrá un cero. Esto es importante, porque las texturas se pueden cargar sólo una vez que hay un contexto de OpenGl activo. La caché solo almacenará las texturas que se han podido crear en openGl. Ese identificador puede usarse normalmente como cualquier textura generada a través de un BMP o lo que sea. Por ejemplo:

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, ETSIDI::getTexture("imagenes/fondo.png").id);
glDisable(GL_LIGHTING);
glBegin(GL_POLYGON);

glColor3f(1,1,1);
glTexCoord2d(0,1);          glVertex3f(-10,0,-0.1);
glTexCoord2d(1,1);          glVertex3f(10,0,-0.1);
glTexCoord2d(1,0);          glVertex3f(10,15,-0.1);
glTexCoord2d(0,0);          glVertex3f(-10,15,-0.1);
glEnd();

glEnable(GL_LIGHTING);
glDisable(GL_TEXTURE_2D);
```

LOS SPRITES

Se incluyen en la librería dos clases relacionadas, que son los Sprites y las Secuencias de Sprites. Los primeros permiten dibujar imágenes que se pueden mover y rotar cómodamente así como invertir su dirección (flip) de pintado, y escalar. Se incluye además un código muy básico de detección de colisión entre dos sprites (el rectánculo que lo define). Aunque la imagen cargada sea de unas dimensiones determinadas es posible adaptarla al tamaño que se desee por medio de la redifinición de su ancho y alto. Notese que para poder crear un Sprite e necesario basarse en un fichero de imagen.

Las secuencias de Sprites están pensadas para trabajar con ficheros png que contienen distintos estados de un mismo objeto (normalmente animaciones). Básicamente cuartea una imagen en filas y columnas (debe definir el número de ellas el programador) y después trabaja con ellas como si fueran índices. Es especialmente cómodo el hecho de que se autotemporiza de forma que podemos cargar un fichero de una explosión y olvidarnos de ir actualizando su estado.

```
class Sprite
{
public:
	Sprite(const char *texturePath, float x=0, float y=0,
		float width=-1, float height=-1);
	void draw();
	void loop();
	void setPos(float x, float y);
	void setVel(float vx, float vy);
	void setSize(float w, float h);
	void setCenter(float cx, float cy);
	void flip(bool h,bool v);
	bool isFlippedH() const;
	bool isFlippedV() const;
	void setAngle(double angle);
	double getAngle() const;
	bool collides(const Sprite &s) const;
};
class SpriteSequence: public Sprite
{
public:
	SpriteSequence(const char *texturePath, int cols, int rows=1,
			int ms_step = 50, bool repeat = true,
			float x=0, float y=0,
			float w=-1.0F, float h=-1.0F,
			int initState=0);
	void draw();
	void loop();
	void setState(int index, bool pause=true);
	int getState();
	void pause(bool stop=true){_pause=stop;}
};
```