Videojuegos, C/C++ y SDL

Rubén Darío Orozco Zapata
drincast@hotmail.com
2008

Creative Commons License Deed

Reconocimiento-No comercial-Compartir bajo la misma licencia 2.5 Colombia

Usted es libre de:

- copiar, distribuir y comunicar públicamente la obra
- hacer obras derivadas

Bajo las condiciones siguientes:

- **Reconocimiento**. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).
- **No comercial**. No puede utilizar esta obra para fines comerciales.
- **Compartir bajo la misma licencia**. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia iéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Advertencia

Este resumen no es una licencia. Es simplemente una referencia práctica para entender el Texto Legal (la licencia completa) es un redactado inteligible por cualquiera de algunos de los términos clave de la licencia. Se trata de una interficie amigable del Texto Legal que hay debajo. Este resumen por si mismo no tiene valor legal, y su contenido no aparece en la auténtica licencia.

Creative Commons no es un bufete de abogados y no ofrece servicios legales. La distribución, la muestra, o el enlace de este Resumen no crea ningun vínculo abogado-cliente.

Los derechos derivados de usos legítimos u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.

Esto es un resumen fácilmente legible del texto legal (la licencia completa).

A todos los desarrolladores de videojuegos,
desde el "pequeño" desarrollador,
a las famosas empresas,

Y a todos los jugadores
de videojuegos del mundo

Gracias por darme una diversión,
que disipa odio que hay en el mundo…

Agradecimientos

A Martin Richards, KenThompson, Dennis Ritchie por que gracias a ellos nació el lenguaje de programación C, a Bjarne Stroustrup por desarrollar C++, a los desarrolladores y colaboradores de la librería SDL por su magnifico trabajo, sin todos ellos hoy no tanta diversión. A mis amigos que siempre dan alegría a la vida y a mi familia por aguantarme tanto y hacerme la vida agradable y más fácil de soportar.

Índice

Introducción

El presente texto, es una guía para los que están interesados en el mundo de la programación y más en la programación de videojuegos, con esto no quiero decir que sea un libro extenso sobre la programación de videojuegos, ni que trate todos los temas que hay sobre ellos y de cómo desarrollar un excelente videojuego, ni las mejores técnicas de desarrollo para realizar un videojuego.

Lo que pretendo con este texto es presentarles unas bases para que se encaminen en este maravilloso mundo lleno de imaginación y creatividad, para esto parto de dos ejes fundamentales para que puedan seguir este texto, estos ejes son, la programación en lenguaje C/C++ y el manejo de una librería que nos facilitara la programación gráfica y muchas cosas más que tengan que ver con los videojuegos, esa es la librería multiplataforma SDL (Simple Directmedia Layer).

Supondré que el interesado o sea tu ya posees conocimientos básicos de programación estructurada y orientada a objetos, así que no me centrare mucho en detalles extensos sobre la programación, además también supondré que sabes que es un videojuego, algo de su historia, etc. así que no entrare en muchos detalles sobre esto tampoco, lo primordial de este texto es encaminarte un poco sobre la programación de un videojuego utilizando una librería como SDL para facilitar un poco las cosas, valga aclarar que este es un corto camino de los muchos y más extensos que hay en el mundo del desarrollo de video juegos, pero por algo se comienza.

Espero que este texto le agrade y les sea de utilidad, hay de antemano pido disculpas si no doy a entender todo, parte o algo de lo que se encuentra en este texto, pero si tienen alguna duda, comentario, sugerencia o critica háganmela saber, escribiendo a drincast@hotmail.com, pues me interesa mucho su opinión y aporte ya que la realimentación es la que nos ayuda a identificar los problemas.

Algo de historia e información técnica

"Los secretos de nuestras épocas están codificados en nuestros juegos"
No me acuerdo del autor, disculpas.

1.1 Historia de los videojuegos

¿De donde nacen los videojuegos?, pues como muchas cosas buenas que hay en este mundo, que nacen de algo tenebroso y oscuro, los videojuegos nacen a partir de la guerra fría por hay en los cincuenta, Willian o Willy Nighinbotthan en 1958, creo la primera forma de entretenimiento interactivo en un aparato electrónico, eran solo dos líneas rudimentarias y una "pelota", esto era una especie de tenis, después apareció Steve Russell, creo un jueguito de naves en un maravilloso PDP llamado "spacewar" y gracias a este personaje es que tenemos el poder en nuestra manos el maravilloso y todo poderos, el control de mando, si, esa cajita con botones y cursor que utilizas en tu XBOX, PlayStatio, etc. y luego viene Ralph Baer y crea la primera consola casera la "magnavox odyssey", después Nolan Bushnell funda en 1972 la empresa Atari, este personaje fue el que creo el negocio del videojuego "y por ello, los videojuegos son tan caros ja ja ja" (una pequeña broma?). Bueno creo que hasta aquí lo de la historia, creo que hay que destacar algunos juegos que aparecieron como famoso PONG (Al Alcorn), Space invader todo un clásico, Pac-MAN el primer héroe, el entretenido e intelectual Tetris.

1.2 Sobre SDL

SDL (Simple DirectMedia Layer), es una API creada por Sam Lantinga de Locky Games para el desarrollo de videojuegos, demos, emuladores, reproductores y más, algo de lo que te permite hacer SDL es establecer un modo de video y utilizar la memoria de video o de sistema para trabajar con gráficos, que en términos de SDL se conocen como superficies (Surfaces), te permite el manejo o detección de eventos para saber si se movió el mouse o se presiono alguna tecla del teclado, etc., te permite trabajar o manipular sonido en tus programas, manejo de la unidad de CD-ROM, manejo de hilos, temporizadores, además de ofrecernos lo anterior esta API es multiplataforma ya que se puede hacer desarrollos para SO (sistemas operativo) Windows, Linux, MacOS, entre otros y además de esto también la podemos utilizar en muchos lenguajes de programación como Perl, C, C++, PHP, entre otros y otro punto a favor más es que esta bajo licencia

LPGL (licencia publica general menor) o sea que tu aplicación creada con SDL no necesita ser de código abierto, pero hombre todo es conocimiento y como tal deberíamos aportar con algo.

Esta API esta compuesta pos subsistemas separados, en general son cinco: sistema de video, sistema de sonido, de CD-ROM, Joystick y temporizadores, cada una puede ser inicializada independiente de las demás, SDL permite o fomenta la utilización de librerías que adicionan funcionalidad o mejor nos reducen el trabajo, entre estas librerías tenemos SDL_image, para cargar imágenes de diferentes formatos (png, jpg, pcx, etc.), SDL_mixer, que nos permite la manipulación de audio de diferentes formatos, SDL_ttf, que nos permite el manejo de tipografía true type texto.

Para comenzar a trabajar con SDL, primero que todo debemos descargar la API, esta la encuentras en el sitio oficial, en Linux en distribuciones como debian solo necesitas utilizar el comando “apt-get” y instalamos los paquetes libsdll.2debian, libsdll.2-dev y listo, pero en nuestro caso vamos a utilizar el SO Windows (en mi caso XP), y uno de los 2 IDE’s, DevC++ o Visual Studio C++ Express 2005, para instalar esta librería en DevC++ existe un paquete que viene para añadírselo por medio del gestor de paquetes de DevC++, pero realice una guía de instalación de SDL en estos 2 IDE’s, así que te recomiendo que le eches un vistazo en http://www.unicauca.edu.co/rorozco (si por algún no encontraste la guía o no se puede visualizar escribe un correo a mi e-mail)., puede estar en la sección de tutoriales o en la sección de noticias, para Linux no he realizado una guía de instalación pero muy pronto, eso si paso a paso, o si prefieres puedes buscar en la red, hay suficiente material referente a este tema, de todos modos el apéndice A esta dedicado un poco al tema, así que échale un vistazo, pero te recomiendo la guía es un poco más completa; ha me olvidaba la pagina oficial de SDL http://www.libsdl.org , aquí podes encontrar en la sección de descarga los binarios, yo descargue la versión SDL-1.2.13, eso si baje los binarios y las librerías de desarrollo, también en la sección de librerías, por medio de su buscador podes buscar las librerías SDL_image, SDL_mixer, SDL_ttf, también bájate los binarios y las de desarrollo.

Es importante que ya tengas descargados e instalado la API, para seguir con este texto si eres inexperto utilizando SDL y quieres realizar los ejemplos que encontraras más adelante, pero si no es así, bien puedes continuar.

1.3 Otras librerías

Existen muchas librerías para el desarrollo de videojuegos, algunas robustas otras muy simples, describiré muy breve algunas de ellas.

Allegro: esta librería también es desarrollada en C como SDL, Allegro es diseñada para el desarrollo de videojuegos, es parecida a SDL, pero difieren en mucho, mi experiencia se basa en la instalación para el IDE DevC++, pues es solamente bajar el paquete para el IDE e instalarla con el gestor de paquetes de DevC++, creo que tengo un poco más de experiencia en esta librería que utilizando SDL, he probado algunos juegos que han hecho con la librería allegro y me han parecido geniales a pesar de que solamente son demos.

Yo realice un simple juego con esta librería, es un pong, el cual es un desastre, nada de gráficos bonitos ni sonido, pero el fuerte de el es el código.

El objetivo de haber desarrollado el juego era probar la librería Allegro y generar un estructura básica que me permitiera crear juegos a partir de unas clases "generales", realmente creo que cumplí el objetivo propuesto, pues desarrolle una serie de clases que se pueden reutilizar para crear diferentes juegos, hago una aclaración y es que el las clases no solo fueron creadas por mi, me base en un articulo que hay en la red que pertenece a Daniel Acuña Norambuena (CURSO DE PROGRAMACIÓN DE VIDEJUEGOS CON C++ Y ALLEGRO), si por casualidad le quieres echar un vistazo descárgalo de esta url: http://www.unicauca.edu.co/~rorozco/archivos/tutoriales/videojuegos/allegro/pon_A.rar el archivo es un comprimido que contiene, el proyecto para DevC++ y los documentos que genera a medida que desarrollaba el juego (si por algún motivo no se puede descargar el archivo escribe un correo a mi e-mail).

OpenGL (Open Graphics Library): Es una API grafica multiplataforma creada por Silicon Graphics, esta solamente se basa en el aspecto grafico de una aplicación, las otras son la librería GLU (OpenGL Utility Library) una librería de utilidades, GLX (OpenGL Extension to the X Windows Systems) para interactuar con sistemas de ventanas X Windows. Esta librería nos facilita la creación de videojuegos en 3D, tengo muy poca experiencia manejando esta librería, solamente he hecho un simple ejemplo, pero promete mucho.

DirectX: Es una API multimedia creada por Microsoft, que consta básicamente de cuarto partes Direct3d para gráficos, DirectSound y DirectMusic para los sonidos y DirectInput para el control del teclado, Mouse, etc., esta también no sirve para crear juegos es espacios 3D pero solamente se utiliza en SO Windows, tengo poca experiencia en esta, solamente he realizado un sencillo ejemplo, pero es realmente potente.

Un Videojuego

"la llave que siempre se usa está brillante"
Benjamín Franklin.

Primero que todo y a manera de resumen, quiero explicar para los mas nuevos que se necesita para la creación de un juego, en nuestro caso un juego de bajo perfil más no bajo por malo, si no por que no tenemos lo que deben de tener las compañías que desarrollan juegos como capcom, además lo principal para el desarrollo de un buen videojuego es la creatividad.

2.1 ¿Qué Necesito para mi video juego?

No quiero entrar en discusiones sobre esta pregunta, lo que escribo aquí no es más que una guía de lo que necesitas para crear un videojuego, no son los pasos generales y un estándar a seguir solamente es mi opinión, y como lo dije en la introducción, este texto es una guía o mejor dicho, una idea de cómo se puede, no de así es, lo que espero con esto es despertarles la chispa de imaginación y que ustedes sigan sus propias reglas, como por ejemplo nadie le dijo a Nolan como hacer de los videojuegos una industria.

Primero lo primero, y esto es la idea de lo que quieres convertir en un juego, tu idea por simple que sea es la fuente de tu poder, las cosas sencillas los las que mejor funcionan, o, nunca te divertiste con pac-man todavía da batalla, como segundo y esto es tu gasolina para ver realizado tu proyecto es tener las ganas y la convicción de que si se puede, después de esto debes preguntarte según tu idea, en que clase de videojuego se enmarca tu idea, existen muchos tipos de juegos yo creo que ni los conos todos, solamente me baso en los básicos, a continuación describo algunos.

Arcade: Pac-Man, Tetris, etc.

RPG: como el famoso Final Fantasy, mucha acción, aventura y estrategia, este genero es el que a consumido mayor parte de mi tiempo, algo triste, pero no los puedo dejar.

Simulación: como los juegos de Formula 1, los de aviones como EagleOne, el reconocido Ned for Speed.

SideScrollers: como SuperTux, el viejo pero divertido Mario Bross, Naruto Ninja Council de GBA.

Juegos de estrategia: estos juegos son como los de Age of Empires, Wesnhost, Civ, etc, al parecer lo más importante de este tipo es la programación de la inteligencia artificial, ya que realmente este es el fuerte que atrae a los jugadores de este genero, por otra parte, según los teóricos dicen que no se necesita muchos efectos y calida en gráficos para el desarrollo de este tipo de juegos, pero como la comida, lo primero es la impresión.

Shooter's: Estos son los famosos juegos de primera persona (DOOM, Quake, no se cual más, etc.) o los de tercera persona (Max Payne, Super Mario 64, Tomb Raider, etc.), en este tipo de juegos si se necesitan gráficos de buena calidad, personalmente me gustan más los juegos en tercera persona.

Deporte: FIFA 2006 y de más, los de baloncesto, los de tenis como virtual tenis, realmente no se si es un genero aparte me queda difícil definirlo dentro de alguno de los anteriores.

Hay muchos más pero creo que estos son los básicos.

Después de esto se necesitan habilidades como por ejemplo, saber programar en algún lenguaje como C++, C#, Java (para este libro nos basaremos en C/C++), los conocimientos generales de este mundo como lo son las matemáticas y la física, manejo de sonidos y gráficos, creo que la parte de gráficos es muy importante, para mi cuenta más esto que otros aspectos menos el aspecto de la trama y jugabilidad este para mi es el más importante. Pero si estas en la misma situación que yo que para gráficos como que regular y además de esto no tengo amigos cercanos enrollados con el cuento del desarrollo de videojuegos (jugarlos si, pero desarrollarlos ¡que aburrido!), deberemos trabajar un poco más duro, por que nos toca todo a nosotros solos, pero, si por el contrario cuentas con amigos que también desean crear un videojuego y además de esto saben dibujar gráficos digitales, pues hombre, si que tienes una herramienta poderosa para que tu proyecto se finalice bien y con éxitos.

Una regla muy importante y que la realizaron muchos de los duros en desarrollo de videojuegos en sus inicios es, el juego que estas desarrollando termínalo, no comiences otro antes de terminar el actual.

2.2 El corazón del juego

Entrando en término de desarrollo o en código, lo importante y el esqueleto de un juego es su ciclo, describiré una opinión sobre en ciclo de un juego, pero primero ¿en que se basa un juego?, pues un juego debe tener entradas las cuales sirven para que el usuario interactué con el nuestro, luego de esto viene procesar estas entradas para ejecutar las acciones correspondientes, luego viene la actualización de los gráficos (mejor dicho de los actores o personajes y objetos de nuestro juego), luego procesamos o actualizamos el

sonido, después de esto nos queda es volcar o mandar los gráficos actualizados a la pantalla, sería algo así.

```
Mientras (no finalice){
      ProcesarEntrtadas();
      ActualizarGraficos();
      ActualizarSonido();
      MostrarGraficos();
      ReproducirSonidos();
}
```

Esto es solo un ejemplo, existen diferentes formas de realizar el ciclo, este solo es para que te hagas una idea de cómo es el ciclo de un videojuego.

En el siguiente capitulo ya entraremos al código en si para esto, vamos a utilizar:

- Lenguaje de programación C/C++.
- Librería SDL
- IDE de Visual C++ 2005 Express.

Los dos primeros requisitos si son obligatorios, el IDE lo puedes escoger a tu gusto, yo utilizo visual C++ por que me facilita un poco más el trabajo, pero tu puedes utilizar DevC++ o otro que sea para Windows o si prefieres utilizar los que existen para Linux, no hay problema, aun que el primer ejemplo que se realice lo desarrollare en Visual C++ y luego lo pasaremos a DevC++, para que veas que no es mucha la diferencia.

Comenzando con la Librería SDL

"emprendiendo el camino, pero adonde …"

En este capitulo veremos como utilizar la librería SDL en nuestras aplicaciones, esto es realmente sencillo, me basare en un ejemplo corto, el cual lo crearemos primero en Visual C++ 2005 Express y luego lo pasaremos a DevC++.

3.1 Manos a la obra

Empecemos por abrir nuestro entorno, seleccionamos el menú ***Archivo – Nuevo – Proyecto*** y seleccionamos el tipo de proyecto ***Aplicación de consola Win32***, le damos un nombre "Ejemplo1" y seleccionamos el directorio don de queramos guardar nuestro proyecto, en la figura 1, puedes ver la ventana de selección de proyecto.

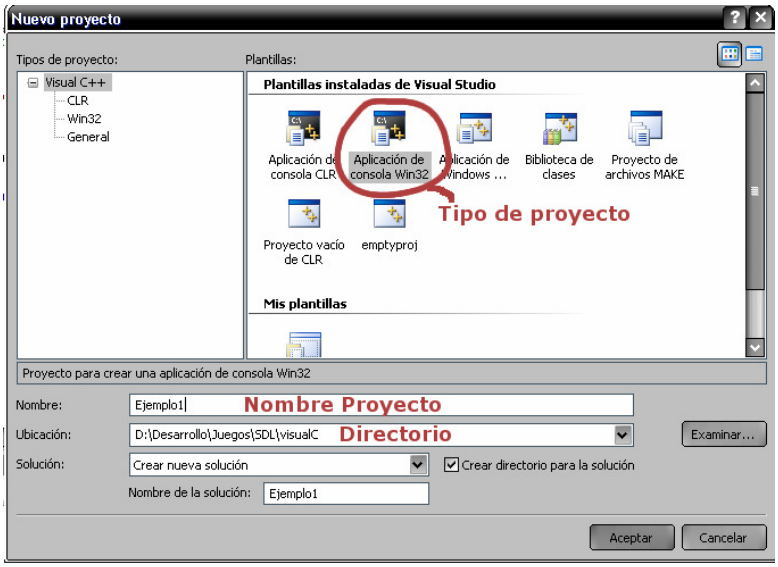

Figura 1. Selección del proyecto.

Ahora configuramos nuestro proyecto para que incluya las librerías de SDL, claro que para hacer esto ya debes de haber instalado SDL y decirle al IDE (VisualC++) donde encontrar las librerías, seleccionamos el menú ***Proyecto – Propiedades de Ejemplo1…*** desplegamos ***propiedades de configuración*** luego ***vinculador*** y seleccionamos ***Línea de comandos*** y el la sección de opciones adicionales escribimos lo siguiente SDL.lib SDLmain.lib, SDL_image.lib, SDL_mixer.lib, SDL_ttf.lib, en la figura 2 puedes ver la configuración.

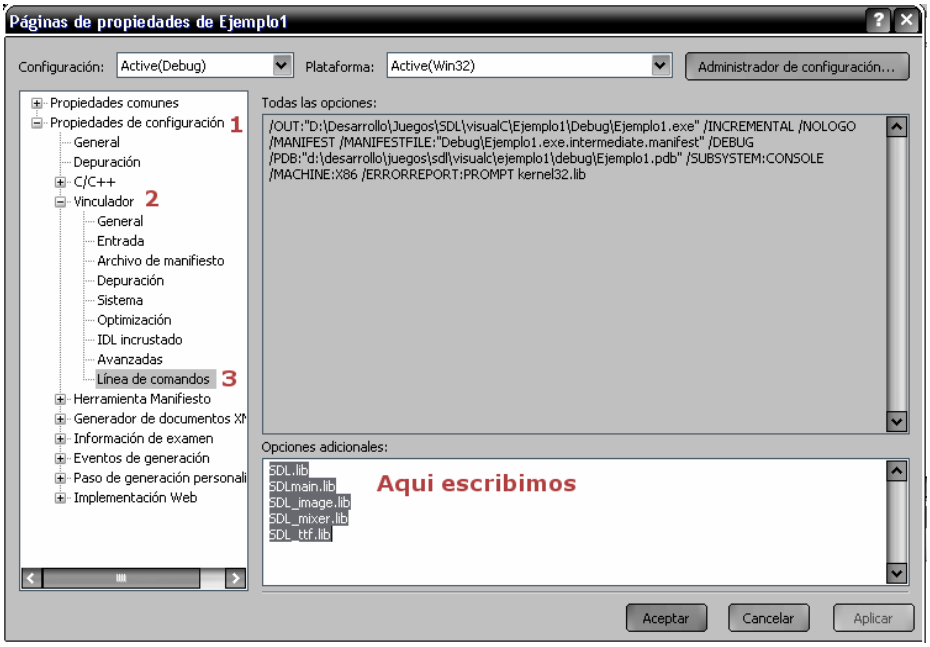

Figura 2. Configuración del Proyecto

Lo primero que tenemos que hacer es incluir los archivos de cabecera a nuestro archivo cpp “Ejemplo1.cpp”, Lugo compilamos con F7 o en el menú ***Generar – Generar Solución***, si todo ha salido bien nos saldrá este mensaje en la ventana de resultados.

```
------ Operación Generar iniciada: proyecto: Ejemplo1, configuración: Debug Win32 ------
Compilando...
stdafx.cpp
Compilando...
Ejemplo1.cpp
Compilando manifiesto en los recursos...
Vinculando...
Incrustando manifiesto...
El registro de generación se guardó en el
"file://d:\Desarrollo\Juegos\SDL\visualC\Ejemplo1\Ejemplo1\Debug\BuildLog.htm"
Ejemplo1 - 0 errores, 0 advertencias
========== Generar: 1 correctos, 0 incorrectos, 0 actualizados, 0 omitidos ==========
```
Mensaje 1. Mensaje de éxito

La parte importante de este mensaje es la ultima que nos dice que hay 1 correcto, 0 incorrectos, si aparece 1 incorrectos, seguro es por que no has adicionado bien la librería SDL, date un tiempo y revisa que todo lo hayas hecho bien, si es necesario vuelve a vincular la librería al entorno de desarrollo (VisualC++).

Bueno continuemos, ahora vamos a iniciar la librería SDL; con la función ***SDL_init()*** se carga e inicializa dinámicamente la librería, como se dijo anterior mente SDL se maneja

por medio de sistemas o subsistemas, para activar los sistemas se utilizan unos flags, vamos a activar el sistema de video y el de sonido, pero primero veamos los flags:

- **SDL_INIT_AUDIO**: activa el subsistema de audio para el manejo de sonidos.
- **SDL_INIT_VIDEO**: activa el sistema de video, para poder desplegar gráficos en pantalla.
- **SDL_INIT_CDROM**: activa el subsistema de CD-ROM, para el manejo de la unidad.
- **SDL_INIT_TIMER**: activa los temporizadores, para el manejo de tiempo.
- **SDL_INIT_EVERYTHING**: activa todos los subsistemas.

La función ***SDL_init(Uint32 flags)***, recibe como un flan de estos o varios, veamos el ejemplo

SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);

Aqui le decimos a la libreria que el subsistema de video y el de audio, por medio del operador "o" unario ***"|"***, podemos añadir los subsistemas que queremos se inicialicen, la función ***SDL_init()*** retorna un tipo de dato int si este valor es menor a 0 es por que a ocurrido un error en la inicialización de los subsistemas.

Una función, muy importante para nosotros obtener información d algun error ocurrido con SDL, es la función ***SDL_GetError()***, esta función lo que hacec es retornar un puntero a char, esta cadena de caracteres es el mensaje de error que ha ocurrido.

Siempre que inicializamos la librería, al finalizar la aplicación debemos desactivarla, esto se hace por medio de la función ***SDL_Quit()***, por lo general llamanos a esta función al final de nuestro programa.

En SDL un concepto importante son las superficies (surfaces), una superfice a manera simple de explicar es un grafíco el cual queremos desplegar en pantalla, pero en realidad en SDL una superficies es una estructura la cual puede almacenar una imagen o un conjunto de imágenes formadas por varias superficies, este estructura se llama SDL_Surface, acontinuacion veremos la definición de esta estructura:

```
typedef struct SDL_Surface{
      Uint32 flags;            /*sololectura*/
      SDL_PixelFormat *format; /*sololectura*/
      int w, h;                /*sololectura*/
      Uint16 pitch;            /*sololectura*/
      Void *pixels;            /*lectura y escritura*/
      /*clipping information*/
      SDL_Rect clip_rect;     /*sololectura*/
      int refcount;           /*solo lectura*/
}SDL_Surface;
```

Código 1. estructura SDL_Surface

Explicación de los campos:

flags: son unos flags que indican el tipo de superficie.

format: un puntero a una estructura que indica el formato de cada pixel, ya lo veremos.

w, h: ancho y alto de la superficie.

pitch: corresponde a la longitud de una línea de escaneo (scanline) de la superficie, medido
en bytes.

pixels: puntero al comienzo de los datos (pixeles) de la superficie.

clip_rect: estructura que indica el área (un simple rectángulo), el cual indica el area en donde se puede pintar en la superficie.

refcount: contador de referencia, usado cuando se libera la superficie.

Ahora veamos la estrtuctura SDL_PixelFormat:

```
typedef struct{
  SDL_Palette *palette;  //es un puntero a la paleta si
                         //utiliza bpp de 8 si no es NULL
  Uint8  BitsPerPixel;   //bpp número de bits para representar
                         //cada pixel (8, 16, 24, 32)
  Uint8  BytesPerPixel;  //bytes por pixel (1, 2, 3, 4);

  Uint32 Rmask, Gmask, Bmask, Amask; //componentes de colos
                                     //RGBA (rojo, verde, azul y alpha)

  Uint8  Rshift, Gshift, Bshift, Ashift; //desplazamiento binario
                        //hacia la izquierda de cada componente de
                        //color en el valor del pixel.

  Uint8  Rloss, Gloss, Bloss, Aloss; //pérdida de precisión de
                              //cada componente de color

  Uint32 colorkey; //aquí se almacena el color que es transparente en
                 //la superficie. un color que no queremos que se vea en
                 //pantalla

  Uint8  alpha; //el valor del canal alpha de la superficie
} SDL_PixelFormat;
```

Código 2. estructura SDL_PixelFormat

Otra estructura importate es SDL_Rect:

```
typedef struct{
  Sint16 x, y; //posición superior izquierda del rectangulo
  Uint16 w, h; //alto y ancho o esquina inferior dedrecha
} SDL_Rect;
```

Código 3, Estructura SDL_Rect

Esta estructura nos permite definir una coordenadas que conforman un regtangulo, se utiliza para definir el regtangulo o los limites dee una superficie, una de las funciones que cumple es indicar, en donde se pinta una superficie y que parte de la superficie quermos mostrar.

Ahora veamos como establecemos un modo de video, esto lo hacecmos por medio de la función ***SDL_SetVideoMode()***, esta función retorna una suprficie, la cual será nuestro modo de video o mejor nuestra pantalla.

```
SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags);
```

Width: ancho de la pantalla.
Height: alto de la pantalla.
Bpp: bits por pixel, o profundidad de pantalla
Flags: unos modos los cuales indican varias cosas sobre la superficie, miremoslos.

SDL_SWSURFACE	Con esto obligamos que la superfice se cree en la memoria del sistema o sea la RAM
SDL_HWSURFACE	Crea la superficie en la memoria de video (Tarjeta de video).
SDL_ANYFORMAT	Funciona cuando una profundidad (bpp) no es soportada por la tarjeta de video, y "obliga" a SDL a utilizar la superficie de video.
SDL_HWPALETTE	Permite el acceso a la paleta de colores, util para obtener colores con la función ***SDL_SetColors()***
SDL_DOUBLEBUF	Habilita el doble buffer, este es un concepto muy utilizazdo en los videojuegos.
SDL_FULLSCREEN	Activara el modo de pantalla completa, no funciona siempre.
SDL_RESIZABLE	Crea una ventana de tamaño modificable.

Tabla 1. Flags de una superficie o modo de video.

Los que más se utilizan cuando se inicializa un modo de video son SDL_SWSURFACE o SDL_HWSURFACE, junto a SDL_DOUBLEBUF.

La función ***SDL_SetVideoMode()***, retorna NULL cuando ha ocurrido algun error en la inicialización del modo de video.

Bueno te preguntaras para que te hice abrir el IDE, si no estamos codificando nada, pues primero como en todo deebemos ver algo de teoria, que se me estaba escapando antes de realizar nuestro primer programa, aquí va el código.

```
#include "stdafx.h"
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>

int _tmain(int argc, _TCHAR* argv[])
{
      SDL_Surface *screen;          //esta será nuestra pantalla

      //por el momento solo el subsistema dee video
      if (SDL_Init(SDL_INIT_VIDEO) == -1){
            printf("Error al iniciar SDL: %s\n", SDL_GetError());
        return 1;
    }

      //obtenemos un modo de video
      screen = SDL_SetVideoMode(320, 240, 16, SDL_HWSURFACE);
```

```
      //verificamos si ha ocurrido algun error
      if (screen == NULL){
            printf("Error estableciendo el modo de video: %s\n",
                  SDL_GetError());
        return 1;
    }

      SDL_Flip(screen);       //mostrtamos screen en pantalla

      SDL_Delay(5000);        //paramos el programa por 5 segundos arox

      SDL_FreeSurface(screen); //liberamos la superficie

      SDL_Quit();

      system("pause"); //esta función es para visualC++, pausa la consola
      return 0;
}
```

Código 4. La primera prueba.

Copia este código, compila (F7) y si no hay errores, ejecutalo (F5), te debera mostrar dos pantallas una ventana con llena de color negro y una consola cmd detrás de la esta, ahora veremos las funciones nuevas que se presentaron en este código.

SDL_Flip(SDL_Surface *): Se uiliza despues de haber iniciado un modo grafíco, la utilizamos para mkostrtar en pantalla una superficie, en este caso la superficie que tiene el modo de video que solicitamos anteriormente.

SDL_Dealy(Uint32 ms): Se utiliza para bloquear el programa, un determinado tiempo, el parametro de entrada equivale a los milisegundos que deseamos bloquear el programa.

Este pequeño ejemplo era para quitarte el deserpero de programar, ahora escribiremos el ejemplo de este capitulo, este mostrara en pantalla una imagen y reproducira un sonido, primero veamos el ejemplo, y luego explicaremos lo nuevo de este ejemplo, ha puedes reemplazar el código anterior para que no crees otro proyecto.

Pero antes una aclaración, para ejecutar este ejemplo vamos a utilizar dos archivos externos, una imagen, y un archivo de sonido, estos dos archivos deben de estar dentro de la carpeta de proyecto donde se encuentra el archivo Ejemplo1.cpp, por defecto los he nombrado a la imagen *imagen.png* y al archivo de sonido *wav.wav*.

```
#include "stdafx.h"
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>

int _tmain(int argc, _TCHAR* argv[])
{
      SDL_Surface *screen,     //esta será nuestra pantalla
                  *img;        //aqui cargaremos la imagen

      Mix_Music *sonido;       //aqui cargaremos el archivo de sonido
```

```
    SDL_Event eventSDL;     //lo utilizamos para saber si ocurre
                            //algun evento externo, como movimiento
                            //del raton, o presion dee algun botón,
                            //en este ejemplo lo utilizamos para
                            //verificar si se cierra la ventana

    SDL_Rect posDestino; //lo utilizamos para indicar
                //en que parte de screen vamos a colocar la imagen

    int salir = 0;  //es una bandera que nos indicara
                //si salinos de un ciclo

    //iniciamos el subsistema de video y el de sonido
    if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) == -1){
          printf("Error al iniciar SDL: %s\n", SDL_GetError());
      return 1;
}

    //obtenemos un modo de video
    screen = SDL_SetVideoMode(320, 240, 16, SDL_HWSURFACE);

    //verificamos si ha ocurrido algun error
    if (screen == NULL){
          printf("Error estableciendo el modo de video: %s\n",
                SDL_GetError());
      return 1;
}

    //inicializamos un formato de audio
    if (Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 1024) == -1){
          printf("Error en Mix_OpenAudio: %s\n", Mix_GetError());
      return 1;
}

    //colocamos un mensaje o el nombre de la ventana
    SDL_WM_SetCaption("Ejemplo1 - imagen y video", NULL);

    //cargamos la imagen en la suprficie
    img = IMG_Load("imagen.png");
    //verificamos si ha ocurrido algun error cargando
    //la imagen
  if (img == NULL){
          printf("Error en IMG_Load= %s\n", SDL_GetError());
          exit(1);
}

    //cargamos el archivo de sonido
    sonido = Mix_LoadMUS("wav.wav");
    //verificamos si ocurrio algun error
    //al cargar el archivo dee sonido
    if (!sonido)
          printf("Error en Mix_LoadMUS: %s\n", Mix_GetError());

    //si no es null sonido
    if(sonido){
          //verificamos si lo podemos reproducir
          if (Mix_PlayMusic(sonido, -1) == -1)
```

```
                printf("Error en Mix_PlayMusic: %s\n", Mix_GetError());
      }

      posDestino.x = 320/2 - img->w/2;    //la posición eje x
      posDestino.y = 240/2 - img->h/2;    //la posició eje y
      posDestino.w = posDestino.h = 0;

      SDL_BlitSurface(img, NULL, screen, &posDestino); //copiamos img en screen

      SDL_Flip(screen);       //mostrtamos screen en pantalla

      //mientras salir no sea 1 continua
      while (!salir)    {
            SDL_PollEvent(&eventSDL); //escuchamos un evento externo

            //si evento es cerrar ventana
            if (eventSDL.type == SDL_QUIT)
                  salir = 1;

            //esperamos 10 ms
            SDL_Delay(10);
    }

      //si sonido no es null, lo apagamos y liberamos memoria
      if (sonido){
            Mix_HaltMusic();
        Mix_FreeMusic(sonido);
    }

      Mix_CloseAudio(); //apagamos el audio

      SDL_FreeSurface(img);    //liberamos img
      SDL_FreeSurface(screen); //liberamos la superficie

      SDL_Quit();

      system("pause"); //esta función es para visualC++, pausa la consola

      return 0;
}
```

Código 5. Ejemplo1.cpp

Compilalo y ejecutalo de nuevo, te debe aparecer la ventana con la imagen y la consola cmd detrás de la ventana, y escuhar el sonido (figura 3), el programa se finaliza cuando cierres la ventana.

Ahora expliquemos ***Mix_Music** ** es el tipo de dato que nos ofrese la librería SDL_mixer, para cargar archivos de audio, estos pueden ser de diferentes formatos mp3, wav, mid.

SDL_Event es el tipo de dato que nos ofrese SDL para obtener eventos de los dispositivos como el teclado el mouse y del sistema de ventanas, en este almacenaremos el evento “que ha ocurrido”. Un evento en pocas palabra es un suceso que pasa en la aplicación, por ejemplo presionar la tecla “T” del teclado o mover el mouse.

Figura 3. Run de Ejemplo1

SDL_Rect posDestino: Es un tipo de dato que nos ofrece SDL, para "crear" regtangulos, mejor dicho aquí vamos a almacenar, una posicion para la esquina superrior izquierda y la esquina inferior derecha que nos formara un regtangulo virtual, ya sabemos que su estructura tiene cuatro datos enteros x, y, w, h, que nos indican el regtangulo, en el programa lo utilizamos por que la función ***SDL_BlitSurface()*** necesita como parametro de entrada.

La función ***Mix_OpenAudio(44100, MIX_DEFAULT_FORMAT, 2, 1024)***, la utilizamos para solicitar un formato de sonido, el primer parametro es la frecuencia de sonido normalmente es de 44.1 mhz, el segundo parametrto es el formato, por el momento no entro en detalles basta con decir que el que se utilizo es el formato por defecto, el tercer parametro indica los canales o modo stereo o nomo 2 para stereo y 1 para mono.

La función ***SDL_WM_SetCaption("Ejemplo1 – imagen y video", NULL)***, la vamos a utilizar para colocar un nombre a la ventana en donde se ejecutara nuestro juego, el primer parametro es una cadena de caracteres que indicara el mensaje que se desplegara en la ventana, el segundo parametro, por el momento no es importante, pero este indica una cadena de caracteres que representan un icono que se mostrara en la parte superior izquierda ede la ventana.

La linea de código ***img = IMG_Load("imagen.png")***, nos permite cargar una imagen, esta función no la ofrese la librería ***SDL_image***, esta función nos permite cargar diferentes formatos de imágenes por ejemplo jpg, png, pcx, entre otras, el parametro que resibe es una cadena de caracteres, que le indicara la ruta y nombre del archivo a cargar, esta función retorna un NULL si ha ocurrido algun error al cargar el archivo.

La linea de código ***sonido = Mix_LoadMUS("wav.wav"),*** nos la ofrese ***SDL_mixer***, con esta función podemos cargar un archivo de audio, el parametro que resibe indica la ruta y nombre del archivo a cargar, esta función retorna NULL si no se ha podido cargar el archivo de audio. Tambien la librería ***SDL_mixer*** tiene su propia función de obtención de erroes, esta es ***Mix_GetError()***.

La función ***Mix_PlayMusic(sonido, 1)***, nos permite reproducir un sonido prebiamente cargado, el primer parametro es el tipo de dato ***Mix_Music ****, que en el ejemplo es la variable ***sonido***, el segundo parametro es un tipo de dato entero, este representa el ciclo de reproducción, en el ejemplo de mandamos -1 para que se reprodusca indefinidamente hasta que finalice la aplicación, si le colocas el valos de 1 reproducira una sola vez el sonido, esta función retorna un entero negativo si ha ocurrido algun error.

Las linea de código: ***posDestino.x = 320/2 - img->w/2;*** estamos estableciendo la variable ***posDestino*** en la posición x, ***posDestino.y = 240/2 - img->h/2;*** establecemos el valor de la posicion en y, y ***posDestino.w = posDestino.h = 0;*** establecemos el ancho y alto a cero. Te preguntaras por que si ***posDestino*** es del tipo ***SDL_Rect***, pues si te fijaste bien, solamente le estamos diciendo que cree un "punto" solamente no todo el rectangulo, acontinuación veras por que.

La función ***SDL_BlitSurface(img, NULL, screen, &posDestino)***, lo que realiza esta función realizar un volcado de la superficie ***img*** a ***screen***, según unas posiciones estas posiciones se las indicamos mediante un dato deel tipo ***SDL_Rect***, el primer paramero es la superficie que deseamos volcar (*surOrigen*) o pegar, el segundo parametro es el area que realmente deseamos volcar, osea el rectangulo, aquí utilizamos NULL para indicarle que queremos volcar toda la superficie, el tercer parametro es la superficie (*surDestino*) en donde volcaremos *surOrigen*, el cuarto y ultimo parametro es el rectangulo que le indicara en que posición sera volcada *surOrigen* en *surDestino*, aquí solamente es necesario los datos x e y de ***SDL_Rect***, por que solo neceesita un punto o pixel, que es desde el que empezara a volcar a *surOrigen*, ahora si ves por que no era necesario crear todo el rectangulo para ***posDestino***.

La función ***SDL_PollEvent(&eventSDL)***, se utiliza para leer un evnto de la cola de eventos, con este capturamos el evento que esta al inicio de la cola, el parametro que resibe es un dato del tipo ***SDL_Event***, en el cual se almacenara el evento inicial de la cola, y ese evnto se eliminara dee la cola, esta función retor 1 si hay eventos pendientes y 0 si no los hay.

El atributo ***eventSDL.type*** del evento nos indca el tipo de evento que ocurre, por ejemplo ***SDL_QUIT***, nos indica que el evento es de cierre de ventana, si fuera ***SDL_KEYDOWN***, nos indicaria si se ha presionado una tecla.

La funcion ***Mix_HaltMusic()***, detiene la reproducción del sonido cargado previamente en estado de reproduccion, la función ***Mix_FreeMusic(Mix_Music *)***, libera la memoria que se solicito para carcar el archivo de audio, finalmete la función ***Mix_CloseAudio()***, nos detiene el sistema o mejor es formato de audio solicitado con anterioridad, mejor dicho desactiva el audio, más no el subsistema de audio de SDL.

Como viste, realmente es facil utilizar la librería SDL, ahora vamos a pasar este código al IDE DevC++, para ver en que se diferncia con el de VisualC++, en cuanto a código.

Comencemos por abrir el IDE de DevC++, creamos un nuevo proyecto, podemos seleccionar como tipo de proyecto, ***console Application*** o seleccionar la plantilla que

creamos para SDL, por el momento se leccionamos el proyecto tipo ***console Application***, y seleccionamos un nombre por ejemplo “Ejemplo1” (figura 4)

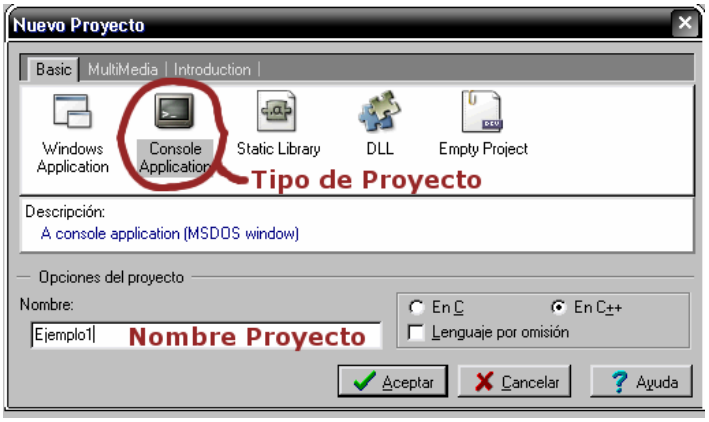

Figura 4. Nuevo proyecto DevC++

Damos click en el botón ***Aceptar***, y seleccionamos la ruta en donde deseamos guardar el proyecto, y damos click en el boton de ***Guardar***.

Ahora configuremos el proyecto para que, funcione con la librería SDL, nos vamos al menú ***Proyecto – Opcione de Proyecto*** y en la sección ***Enlazador(Linker)***, digitamos lo siguiente:
-lmingw32 -lSDLmain -lSDL -lSDL_image -lSDL_mixer, y damos click en ***Aceptar*** (figura 5).

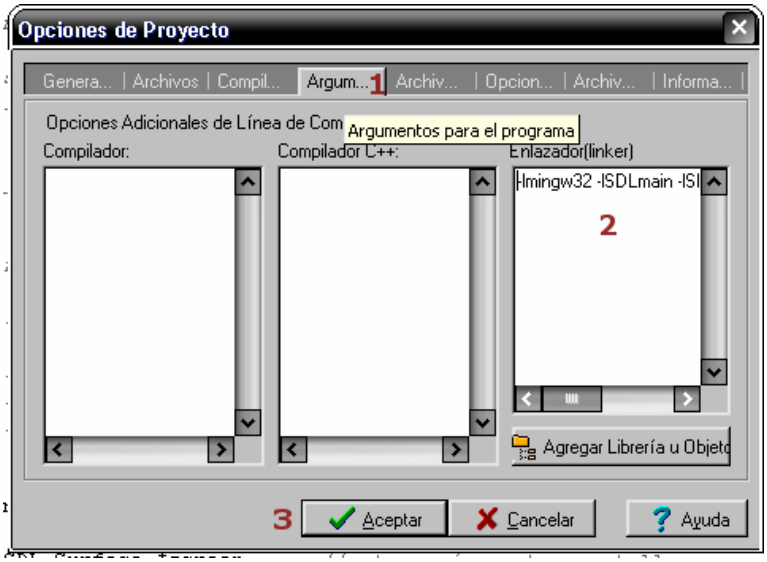

Figura 5. Configuración del proyecto en DevC++

Ahora copiemos tal cual el código que hicimos en visual C++ el de ***Ejemplo1.cpp*** al archivo cpp por defecto del proyecto de DevC++, ahora guardamos el archivo con el mismo nombre “***Ejemplo1,cpp***”, ahora compila, al compilar nos dice que no encuentrta una inclusión, pues facil la quitamos, compilamos de nuevo, y nos muestra un error en la función main (*‘_TCHAR’ has no been declared*), pues facil cambiamos esta al modo

estandar así ***int main(int argc, char *argv[])***, compila de nuevo, y listo sin problemas, no todabia no.

Inclullamos, ***<cstdlib>*** antes ***<stdlib.h>*** y ***<iostream>***, compilemos de nuevo, listo no hay errores, ahora ejecutemos, ha pero primero acuérdate de pasar los archivitos imagen.png y wav.wav, a la ruta donde guardaste el proyecto, ahora si podemos ejecutar nuestra aplicación, listo corrió bien, pero por que no se cierra la consola cmd, y si la cerramos aun continua ejecutándose el programa Ejemplo1 y la cmd, bueno el la lista de procesos activos del administrador de tareas de Windows, pues esto se debe a la función ***system("pause")***, pues sencillo, matados estos dos procesos, y ahora quitamos esta línea de código de nuestro código, compilamos y ejecutamos, y ahora si se ejecuto de forma normal la aplicación, no se por que no funciona ***system("pause")|system("PAUSE")***, si normalmente esta función se incluye en el archivo por defecto de un proyecto tipo consola en DevC++.

Pero te preguntaras como vas ha ver los mensajes que deseas Mostar y los mensajes de error, pues fácil en DevC++, se crean dos archivos de texto que se encuentran dentro de la carpeta en donde tenemos nuestro proyecto, uno se llama ***stdout.txt***, que en el cual se guardan los mensajes que deseas mostrar en la consola y el otro es ***stderr.txt***, en donde se almacenaran los mensajes de error de aplicación, particularmente esta caracteriza es muy buena, pues no dependemos de la consola para ver los mensajes si no que estos se almacenaran en los archivos, en Visual C++, al parecer no se activa esto, pero nosotros podemos crear nuestros propios archivos para guardar mensajes, más adelante mostrare una pequeña forma, para almacenar los mensajes de error.

Si lo notase no es muy difícil pasar el código de Visual C++ a DevC++, como recomendación para hacer tu código portable, es no utilizar ninguna de las clases que proporciona .Net y Visual C++, que no tengan que ver con el estándar ANSI/ISO de C++, pero como herramienta para ver que no es portable pues compilemos el código en DevC++ (compilador Mingw), el nos mostrara las funciones o tipos de datos que no están en el estándar.

Ahora antes de continuar con el siguiente capitulo, practica un poco, cacharrea para que te apropies de las funciones y explora, lo bueno de trabar en Visual C++, es su IntelliSense, el cual nos ayuda a ver todo el pull de funciones que trae SDL y además de esto nos permite ver las funciones que vamos creando, con solo pulsar CTRL+Space, en cualquier parte o después de un objeto o apuntador, DevC++, también viene con esta función, pero en mi caso no me funciona a veces me arroja un error cuando selecciono alguna función, no se si es que lo tengo mal configurado.

El código fuente del programa junto a los proyetos, los encuentras en Cap 3/Ejemplo1/VC/Ejemplo1 para visual C++ y Cap 3/Ejemplo1/DevC/Ejemplo1 para DevC++.

Manejo de imágenes y de Eventos

"Se han lanzado los dados"
Julio César

Vamos a realizar un ejemplo que nos demuestra como cargar imágenes y añadirle ciertas características, como colores trasparentes, también miraremos un poco más como manejar eventos del teclado, este capitulo se basa principalmente en estos dos conceptos, además de introducir un poco de orden para nuestro código, como reunir las sentencias de código que utilizamos más en funciones, estos serán los primeros pasos que daremos para empezar a crear el pequeño proyecto que se pretende realizar en este libro, el cual es un "motor" o mejor un conjunto de clases genéricas para el desarrollo de un juego, hago la aclaración y este el "motor" no es pedroso, ni suficiente para emprender proyectos de desarrollo de videojuegos serios, pero sirve para los propósitos de este texto que es presentarte una guía de desarrollo o mejor una idea para que te encamines en esta área.

4.1 Eventos

Como se menciono anteriormente, un evento es un suceso que ocurre en el programa, en pocas palabras es un acción que particularmente es emprendida por un usuario a través de un periférico como el teclado, los eventos pueden ser producidos por el teclado, el Mouse, eventos internos del sistema operativo (cuando se cierra una ventana o cambia de tamaño), en SDL como vimos anteriormente nos ofrece el manejo de eventos de una manera fácil, para enterarnos un poco mas del manejo de eventos, miraremos algunas estructuras.

Como vimos podemos capturar el tipo de evento que se ha generado por medio de la función ***int SDL_PollEvent(SDL_Event *event)***, para explicar un poco los eventos que generan del teclado, veremos la estructura:

```
typedef struct{
    Uint8 type;
    Uint8 state;
    SDL_keysym keysym;
} SDL_KeyboardEvent;
```

- **type:** pude tomar valores como **SDL_KEYDOWN** (alguna tecla se pulso) o **SDL_KEYUP** (se solto la tecla).
- **State:** puede tomar valores como **SDL_PRESSED** (alguna tecla se pulso) o **SDL_RELEASED** (se solto alguna tecla)**.**
- **Keysym:** este campo nois indica que tecla fue la que se presiono, es una estructura que veremos a continuación.

Tabla 2. Estructura SDL_KeyboardEvent

```
typedef struct{
   Uint8 scancode;
   SDLKey sym;
   SDLMod mod;
   Uint16 unicode;
} SDL_keysym;
```

- **scancode:** por lo general no se utiliza, por que este valor depended mucho del SO.
- **sym:** contiene el código de la tecla pulsada. Esto es un valor definido por SDL, un ejemplo es **SDLK_a** (indica la tecla a), **SDLK_UP** (indica la tecla cursor arriba).
- **mod:** indica el estado de teclas especiales como shift, para indicar si esta presionada o no, lo mismo para la tecla ctrl., también nos indica si es la tecla ctrl. Izquierda o derecha, por ejemplo **KMOD_RCTRL** nos indica que esta presionada la tecla CTRL derecha.
- **unicote:** es el campo unicote de la tecla presionada, no se utiliza por lo general.

Tabla 3. Estructura SDL_keysym.

En el Apéndice B, se presentan unas tablas que indican el código ***SDLKey*** que puede tomar ***sym***, y otra tabla de los valores que puede tomas el parámetro ***mod***. Para ver un ejemplo vamos a ver la sentencia que se utiliza para saber si se ha presionado la tecla escape.

```
SDL_Event eventGame;

SDL_PollEvent(&eventGame);

if(eventGame.type == SDL_KEYDOWN){
      switch(eventGame.key.keysym.sym){
            case SDLK_ESCAPE:
                  exit = true;
      }
}
```

4.2 Técnicas de video

Ahora veremos algunas funciones graficas que nos ofrece SDL, para el manejo de superficies. Primero que todo preguntemos que sabemos, ya sabemos cargar una imagen en una superficie, sabemos copiar una superficie en otra, ya sabemos crear un modo de video y sabemos mostrar una superficie (la que representa nuestra pantalla) en la pantalla o en una ventana, listo esto es lo mas básico, pero existen otras funciones que nos

ayudaran a mostrar, nuestros gráficos de una forma más adecuada, en el apéndice C, se explica algunos conceptos generales de gráficos 2D, así que si encuentras alguna palabra rara que no se explica aquí, es muy posible que se encuentre en el apéndice C.

Primero miremos la función ***Uint32 SDL_MapRGB(SDL_PixelFormat *fmt, Uint8 r, Uint8 g, Uint8 b);*** esta función nos sirve para representar un color, el primer parámetro es el formato de píxel que utiliza alguna de las superficies, generalmente la cual vamos a utilizar para pintar algo, el segundo campo representa un valor de 0 a 255 que representa el color rojo (0 menos intenso, 255 más intenso), el tercer parámetro es lo mismo que el anterior pero este representa el color verde, el cuarto representa el color azul, ejemplo.

```
SDL_Surface *su;
SDL_MapRGB(su->format, 0, 0, 255);  //retorna un color azul
```

Existe también ***Uint32 SDL_MapRGBA(SDL_PixelFormat *fmt, Uint8 r, Uint8 g, Uint8 b, Uint8 a);*** esta función es casi igual a la anterior, difiere en que tiene un parámetro de más, el ultimo, el cual representa un color de transparencia o mejor un alpha-blendig, este valor también va de 0 a 255, un valor de 0 hará invisible la superficie y un valor intermedio dará la impresión de que la superficie es transparente como una ventana, y un valor de 255 mostrara tal cual es la imagen. Con esto ya sabemos representar un color, se decidió explicar primero esta función ya que otras funciones solicitan como parámetro de entrada la definición de un color.

Ahora miremos una función un poco más interesante, esta es ***int SDL_SetColorKey(SDL_Surface *surface, Uint32 flag, Uint32 key);*** esta función lo que hace es indicar un color de transparencia o mejor un color que nosotros no queremos que se vea en una superficie, el primer parámetro es la superficie a la cual queremos indicarle el color transparente, el segundo parámetro son unos flags, SDL_SRCCOLORKEY (para indicar que vamos a utilizar un color transparente en la superficie) junto ha SDL_RLEACCEL (para acelerar el blitting (pegar una superficie en la superficie de video)), si por el contrario colocamos 0 le decimos que desactivamos el color transparente, el tercer parámetro es el color el cual no queremos que se visualice cundo presentamos nuestra superficie en pantalla, este color se lo pasamos por medio de la función ***SDL_MapRGB()***.

Esta función hará que un grafico en ves de verse así:

Se vea así:

La superficie sigue siendo el mismo rectángulo pero no nos va a mostrar el color que nosotros decidimos, en este caso un café.

Ahora miremos una función que nos facilita para asignarle a nuestra superficie un canal alpha o alpha-blendig, ***int SDL_SetAlpha(SDL_Surface *surface, Uint32 flag, Uint8 alpha);*** el primer parámetro es la superficie a la cual deseamos aplicarle el canal alpha, la segunda es un flan (0 para desactivar el canal alpha, SDL_SRCALPHA para indicar que vamos a utilizar el canal alpha, junto a SDL_RLEACCEL), y el tercer parámetro es un valor de 0 a 255 como se explico arriba, 0 invisible y 255 opaco (normal), esta función hará que la imagen anterior se vea así con un valor alpha de 52:

Miremos ahora la función ***void SDL_SetClipRect(SDL_Surface *surface, SDL_Rect *rect);*** nos indica un rectángulo dentro de una superficie en el cual podemos realizar operaciones graficas, por ejemplo podemos decirle a una superficie que solo se puede pintar en la mitad superior, el primer parámetro es la superficie, el segundo parámetro es un dato tipo ***SDL_Rect*** que ya lo vimos, el cual nos indicara el área (rectángulo) en el cual podemos utilizar funciones graficas dentro de la superficie.

La función ***void SDL_GetClipRect(SDL_Surface *surface, SDL_Rect *rect);*** nos retorna el clipping de una superficie, el primer parámetro es la superficie, el segundo es un tipo de dato SDL_Rect, en el cual se almacenara el área rectangular en la cual se puede realizar operaciones graficas.

La función ***SDL_Surface *SDL_ConvertSurface(SDL_Surface *src, SDL_PixelFormat *fmt, Uint32 flags);*** nos ayuda a convertir el formato de una superficie en el de otra superficie, esto nos ayuda a facilitarle el trabajo a SDL cuando copiamos una superficie en otra, el primer parámetro es la superficie la cual convertiremos, el segundo parámetro es el formato de la superficie que le deseamos aplicar, el tercer parámetro son flags (SDL_SWSURFACE, SDL_HWSURFACE, SDL_SRCCOLORKEY, SDL_SRCALPHA).

Listo por el momento es suficiente de evento y de funciones graficas, ahora miremos una forma de organizar lo que hemos visto hasta ahora.

Primero sabemos que, vamos a estar continuamente cargando superficies, algunas sin transparencia y otras con transparencia, además de esto tenemos que estar copian superficies dentro de otras superficies, como es el caso de la superficie de video, por que no organizamos esto en un archivo de inclusión el cual podemos llamar RDO_UtilSDL.h (RDO para indicar la compañía, el autor o el nombre de nuestro motor, son iniciales). Ahora es el momento de volver a abrir nuestro editor y crear un nuevo proyecto el cual llamaremos Ejemplo2.

Para ser un poco más organizados, dentro de la carpeta del proyecto creemos dos carpetas, una llamada *lib* para guardar las clases y librerías que creemos (archivos .h y .cpp), y una carpeta *res* en la cual guardaremos las imágenes, archivos de sonido y otros recursos. Dentro de la carpeta lib creemos dos archivos a *RDO_UtilSDL.h* y *RDO_UtilSDL.cpp*, el anterior proceso lo realizamos por medio del explorador de Windows

no por Visual C++, ahora en el explorador de soluciones de visual C++ agreguemos un filtro (una carpeta lógica) en *Archivos de código fuente* y *Archivos de encabezado*, a este filtro lo llamaremos lib, para hacer esto nos paramos con el cursor del Mouse en alguna de estas dos carpeta, hundimos el botón derecho del Mouse y seleccionamos ***Agregar – Nuevo Filtro*** (figura 6) y le damos el nombre, acuérdate que se debe crear en las dos carpetas mencionadas antes, estos filtros son carpetas lógicas ya que en el directorio real en donde se almacena nuestro proyecto esta carpeta no aparece no se crea.

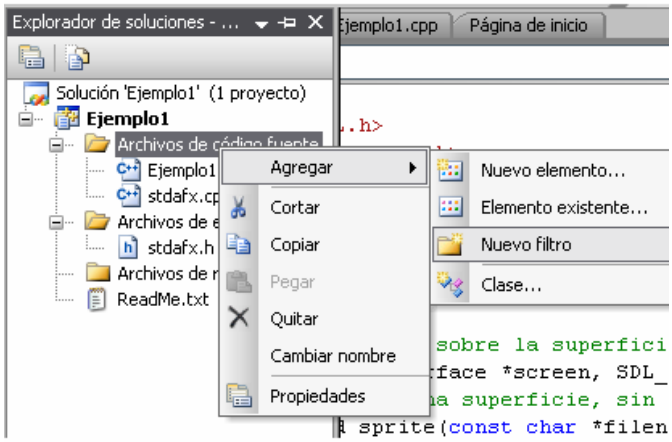

Figura 6, Añadiendo un filtro

Ahora te acuerda que creamos dos archivos anteriormente (*RDO_UtilSDL.h* y *RDO_UtilSDL.cpp)*, pues vamos a añadirlos a nuestro proyecto, en el filtro *lib* dentro de *Archivos de cabecera*, presionamos el botón derecho de Mouse y seleccionamos ***Agregar –Elemento Existente…*** (Figura 7), aquí nos aparece un cuadro de dialogo para seleccionar un archivo, abrimos la carpeta *lib* y seleccionamos *RDO_UtilSDL.h*.

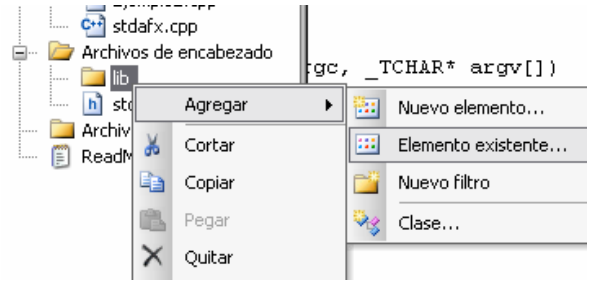

Figura 7. Añadiendo un archivo existente.

El mismo procedimiento lo aplicamos para añadir el archivo *RDO_UtilSDL.cpp* pero en *Archivos de código fuente/lib*, al final nos quedara algo así:

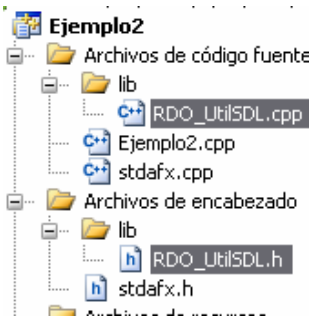

Figura 8. Los nuevos archivos.

Empecemos a editar el archivo *RDO_UtilSDL.h*.

```
//RDO_UtilSDL.h libreria de utlidades
//(c) Rubén Dario Orozco 2008
#pragma once
#ifndef RDO_UTILSDL_H
#define RDO_UTILSDL_H

#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
/****************  Variables Globales  *****************************/
extern SDL_Surface *screen;         // Pantalla
/****************  Prototipo de Funciones  *************************/
//pinta una superficie, sobre otra superficie
//SDL_Surface * en donde copio (destino), SDL_Surface la que copiara (fuente),
//int pos x, int pos y
void blitting_surface(SDL_Surface *destiny, SDL_Surface *source, int x, int y);

//carga una imagen en una superficie, sin ninguna transparencia
//const char * es la ruta en donde esta el archivo
SDL_Surface *load_img(const char *filename);

//carga una imagen en una superficie, con un color de transparencia
//constante (255, 0, 255).
//const char * es la ruta en donde esta el archivo,
//Uint8 cantidad de rojo, Uint8 cantidad dee verde, Uint8 cantidad de azul
SDL_Surface *load_imgT(const char *filename);

//carga una imagen en una superficie, con un color de transparencia
//definido por el usuario.
//const char * es la ruta en donde esta el archivo,
//Uint8 cantidad de rojo, Uint8 cantidad dee verde, Uint8 cantidad de azul
//estos valores van de 0 a 255
SDL_Surface *load_imgT(const char *filename, Uint8 r, Uint8 g, Uint8 b);

#endif
```
Código 6. RDO_UtilSDL.h

Ahora editemos el archivo *RDO_UtilSDL.cpp*.

```
//RDO_UtilSDL.cpp definición de libreria de utlidades
//(c) Rubén Dario Orozco 2008
#include "stdafx.h"
#include "RDO_UtilSDL.h"
```

```
/***************  Variables Globales  ****************************/
SDL_Surface *screen;  // Pantalla

/***************  Prototipo de Funciones  ************************/
void blitting_surface(SDL_Surface *destiny, SDL_Surface *source, int x, int y){
      SDL_Rect rect;

      rect.x=x;
      rect.y=y;

      SDL_BlitSurface(source, 0, destiny, &rect);
}

SDL_Surface *load_img(const char *filename){

      SDL_Surface *tmp;

      tmp = IMG_Load(filename);
      if(!tmp)
            return 0;

      return tmp;
}

SDL_Surface *load_imgT(const char *filename){
      SDL_Surface *tmp, *bmp;

      tmp=IMG_Load(filename);
            if(!tmp) return 0;

      SDL_SetColorKey(tmp, SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGB(tmp->format,
255, 0, 255));
      bmp = SDL_DisplayFormat(tmp);
      SDL_FreeSurface(tmp);
      if(!bmp)
            return 0;

      return bmp;
}

SDL_Surface *load_imgT(const char *filename, Uint8 r, Uint8 g, Uint8 b){
      SDL_Surface *tmp, *bmp;

      tmp=IMG_Load(filename);
            if(!tmp) return 0;

      SDL_SetColorKey(tmp, SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGB(tmp->format,
r, g, b));
      bmp=SDL_DisplayFormat(tmp);
      SDL_FreeSurface(tmp);
      if(!bmp)
            return 0;

      return bmp;
}
```

Código 7. RDO_UtilSDL.cpp

Compilemos, para ver si hay errores y listo ya hemos definido una librería de utilidades, en la cual podemos ir adicionando más funciones las cuales sean de propósito general.

Ahora, si te has dado cuenta, los pasos que siempre vamos a requerir es inicializar SDL, y establecer un modo de video, vamos a crear una clase que nos maneje esta funcionalidad, además de esto vamos a crear un sistema pobre que nos almacene los errores de inicio de SDL y de video en un archivo, es una forma sencilla y fácil aun que no es la mejor.

Creemos una clase nueva para el proyecto, esto lo hacemos en el explorador de soluciones, dando click derecho en *Archivos de encabezado,* luego seleccionamos ***Agregar – Clase…*** seleccionamos ***Clase de C++*** y presionamos el botón *Agregar*, nos abre un cuadro de dialogo, en el cual solo vamos a colocar el nombre "*RDO_initSDL*" y por ultimo presionamos el botón *Finalizar*, y automáticamente se nos crea el .h en Archivos de Encabezado y el .cpp en Archivos de código fuente.

Esta es otra forma de crear clases o archivos .cpp o .h, pero estos archivos se crean en el directorio del proyecto, así los hubiéramos creado dentro del filtro *lib* (carpeta lógica), seguirían por fuera, lo que hicimos anteriormente para los archivos *RDO_UtilSDL*, fue crear primero los archivos .h y .cpp dentro de la carpeta *lib* física no lógica, pero sin utilizar este asistente de creación de clases, para luego adicionarlos al proyecto, de esta manera creo yo que se organiza un poco los archivos de código, y no tenemos todos los archivos .h y .cpp en el directorio del proyecto, esto nos facilita la búsqueda de algún archivo en especial.

Ahora creemos una carpeta física en el directorio del proyecto llamada *log*, la cual almacenara archivos de texto, estos archivos serán mensajes de error que ocurran en la ejecución de la aplicación, a modo de ejemplo esta técnica solamente se implementara en la clase RDO_initSDL, estos mensajes se guardaran en un archivo llamado log_initSDL.txt, ahora editemos RDO_initSDL.h y RDO_utilSDL.cpp.

```
//RDO_UtilSDL.h clase encargada de iniciar SDL y crear un modo de video
//(c) Rubén Dario Orozco 2008

#pragma once
#ifndef RDO_INITSDL_H
#define RDO_INITSDL_H

#include "lib/RDO_UtilSDL.h"
#include <fstream>      //definiciones de manejo de E/S dee archivos
using namespace std;    //para hacer referencia a fstream

class RDO_initSDL
{
private:
      Uint8 sysInit;   //opción para saber que sistemas de SDL se iniciaron
      int H_SCREEN;    //atributo int para el alto de la pantalla
      int W_SCREEN;    //atributo int para el ancho de la pantalla
      int BPP;    //atributo int para la profundidad de color (Bit por pixel)
      ofstream archivoLog;//objeto ofstream, para el manejo de archivos.

public:
      RDO_initSDL(void);
```

```
	//los posibles valores, segun los sistemas que se desenen inicializar
	//0-Video y Timer, 1-Video, Timer y Audio, 2- Video, Timer,
	//Audio, Joystick, 3-TODO
	RDO_initSDL(Uint8);

	//des: inicializa el modo grafico de la libreria SDL de una
	//       forma generica sin flags.
	//ParEntrada: int: ancho.
	//            int: alto.
	//            int: bpp, profundidad.
	//ParSalida:  SDL_Surface *: esta sera la superficie que
	//            representa la pantalla.
	SDL_Surface *initModGraphic(int, int , int);

	//retorna un Uint8, este valor especifica que sistemas de
	//SDL se han iniciado
	Uint8 get_sysInit(void){return sysInit;}
	//establece el atributo sisIniciados, tipo de dato Uint8
	void set_sysInit(Uint8 valor){sysInit = valor;}

	//retorna un int, este valor es el alto de la pantalla
	int get_H_SCREEN(void){return H_SCREEN;}
	//establece el atributo ALTO_PANTALLA, tipo de dato int
	void set_H_SCREEN(int valor){H_SCREEN = valor;}

	//retorna un int, este valor es el ancho de la pantalla
	int get_W_SCREEN(void){return W_SCREEN;}
	//establece el atributo ALTO_PANTALLA, tipo de dato int
	void set_W_SCREEN(int valor){W_SCREEN = valor;}

	//retorna un int, este valor es la profundidad del sistema
	//grafico
	int get_BPP(void){return BPP;}
	//establece el atributo ALTO_PANTALLA, tipo de dato int
	void set_BPP(int valor){BPP = valor;}

	//guarda un mensaje en el archivo log_iniSDL.txt
	void saveMsgFileLog(const char *, const char *);
	//inicializa archivoLog, prepara el archivo log_iniSDL.txt
	void iniFileLog(void);

	~RDO_initSDL(void);
};
#endif
```

Código 8. Clase RDO_initSDL definición.

```
//RDO_UtilSDL.cpp implementación de clase
//(c) Rubén Dario Orozco 2008
#include "StdAfx.h"
#include "RDO_initSDL.h"

RDO_initSDL::RDO_initSDL(void){
	sysInit = -1;
	H_SCREEN = W_SCREEN = BPP =0;

	initFileLog();

	if (SDL_Init(SDL_INIT_VIDEO) < 0) {
```

```
            printf("Error al iniciar SDL (sistema de video)\n");

            fileLog.open("logs/log_iniSDL.txt", ios::out);
            if(fileLog){
                  fileLog <<"Error al iniciar SDL (sistema de video)\n";
                  fileLog.close();
            }

            system("pause");
            exit(-1);
      }
}

RDO_initSDL::RDO_initSDL(Sint8 opc){
      sysInit = opc;
      W_SCREEN = H_SCREEN = BPP =0;

      initFileLog();

      switch(opc){
            case 0:
                  if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER) < 0) {
                        printf("Error al iniciar SDL (sistema de video y
                                    Timer)\n");
                        //abre el archivo para añadir al final de archivo
                        fileLog.open("logs/log_iniSDL.txt", ios::app);
                        if(fileLog){

                              fileLog <<"Error al iniciar SDL (sistema de video
                                    y Timer)\n";
                              fileLog.close();
                        }

                        system("pause");
                        exit(-1);
                  }
                  break;
            case 1:
            if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER | SDL_INIT_AUDIO)<0) {
                        printf("Error al iniciar SDL (sistema de video, Timer y
                                audio)\n");

                        fileLog.open("logs/log_iniSDL.txt", ios::app);
                        if(fileLog){
                              fileLog <<"Error al iniciar SDL (sistema de video,
                                  Timer y audio)\n";
                              fileLog.close();
                        }

                        system("pause");
                        exit(-1);
                  }
                  break;
            case 2:
            if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_TIMER | SDL_INIT_AUDIO |
                SDL_INIT_JOYSTICK) < 0) {
                        printf("Error al iniciar SDL (sistema de video, Timer,
                              audio y Joystick)\n");
```

```
                        fileLog.open("logs/log_iniSDL.txt", ios::app);
                        if(fileLog){
                              fileLog <<"Error al iniciar SDL (sistema de video,
                                    Timer, audio y Joystick)\n";
                              fileLog.close();
                        }

                        system("pause");
                        exit(-1);
                  }
                  break;
            case 3:
                  if (SDL_Init(SDL_INIT_EVERYTHING) < 0) {
                        printf("Error al iniciar SDL (error en todos los
                             sistemas)\n");

                        fileLog.open("logs/log_iniSDL.txt", ios::app);
                        if(fileLog){
                              fileLog <<"Error al iniciar SDL (error en todos
                                   los sistemas)\n";
                              fileLog.close();
                        }

                        system("pause");
                        exit(-1);
                  }
                  break;
            default:
                  printf("La opción no es valida, las opciones son 1 a 3");
                  system("pause");
      }
}

RDO_initSDL::~RDO_initSDL(void){
      SDL_Quit();
      //archivoLog.~basic_ofstream();
      printf("Destructor de iniSDL, se desactivo la libreria SDL\n");
}

SDL_Surface *RDO_initSDL::initModGraphic(int w, int h, int bpp){
      //SDL_Surface *temp;

      W_SCREEN = w;
      H_SCREEN = h;
      BPP = bpp;

      //asigna un modo de video a la supeficie global
      //dedclarada en RDO_UtilSDL
      screen = SDL_SetVideoMode(W_SCREEN,H_SCREEN,BPP,0);

      if(!screen){
            printf("No se pudo iniciar el modo grafico: %s\n",SDL_GetError());

            saveMsgFileLog("No se pudo iniciar el modo grafico: ",
SDL_GetError());

            system("pause");
```

```
            fileLog.~basic_ofstream(); //libera memoria
            SDL_Quit();
            exit(-1);
      }

      return screen;
}

void RDO_initSDL::initFileLog(void){
      fileLog.open("logs/log_iniSDL.txt", ios::out);
      fileLog <<"archivo de registro de la clase RDO_initSDL\n" ;
      fileLog <<"************************************************\n\n" ;
      fileLog.close();
}

void RDO_initSDL::saveMsgFileLog(const char *msg, const char *msgSDL){
      fileLog.open("logs/log_iniSDL.txt", ios::app);
      if(fileLog){
            fileLog <<msg<<": "<<msgSDL<<"\n";
            fileLog.close();
      }
}
```

Código 9. Implementación de la clase RDO_initSDL

Creo que ya te es familiar algo del código de los archivos como la creación de un modo de video y iniciar la librería SDL, por eso solo se explicara la parte de manejo del archivo, lo demás esta documentado en forma de comentarios en la definición de la clase (RDO_initSDL.h).

Para manejar archivos utilizamos el archivo de inclusión ***<fstream>***, el cual nos ayuda en el manejo de entada y salida de flujo para archivos, si te has fijado en los atributos de clase se ha declarado un objeto ***ofstream*** llamado *fileLog*, pues este objeto es el que manipularemos para que se abra el archivo, se almacene algún mensaje y cerremos el archivo cuando hemos terminado las operaciones en el.

Analicemos la función ***RDO_initSDL(void)***, esta función es el constructor de la clase, aquí se inicializan los atributos de clase y llámanos a la función ***initFileLog()***, en esta función, se abre el archivo "*log_iniSDL.txt*" por medio de la función ***open()***, el primer parámetro que recibe está función, es una cadena de texto, la cual representara la ruta en donde se encuentra el archivo, y el segundo parámetro es un modo de apertura, en este caso es ***ios::out***, estos modos de apertura están definidos en el archivo de inclusión ***<fstream>***, ***ios:out*** lo que indica es que se abra un archivo para escritura pero que el contenido que se encuentre en ese archivo se borre, luego la línea de código:

fileLog <<"archivo de registro de la clase RDO_initSDL\n" ;
le dice al objeto fileLog que almacene el mensaje que se encuentra entre dobles comillas, y despues utilizamos ***fileLog.close();*** para cerrar el archivo ya que por el momento no vamos a guarmas nada más.

De nuevo en el constructor, pedimos que se inicialice el subsistema de video de SDL, si por algun motivo ocurre un error al iniciar el subsistema, se entta en la sentencia *if* y aquí se abre de nuevo el archivo "*log_iniSDL.txt*" pero con otro modo de apertura el ***ios:app***,

este modo lo que indica es, que se abra el archivo pero que todo se almacene al final de archivo, sin borrar el contenido que tenga en ese momento, y de nuevo guardamos el mensaje y cerramos el archivo, si ves que no es tan complicado manejar archivos de log o registro, pues te recomiendo que hagas esto para que toda tu aplicación funcione con este sistema de almacenamiento de mensajes, pues es una ayuda grandísima a la hora de depurar el programa, claro que el sistema presentado no es muy robusto, así que mejóralo un poco.

Bueno llego la hora de compilar, y mirar si no ha ocurrido algún error, si hay algún error fíjate bien el los archivos de inclusión, no creo que los errores se han de mayor problema.

Bueno te preguntaras que hace la función ***saveMSgFileLog()***, pues sencillo se utiliza para guardar un mensaje en el archivo de log, si te fijas en la función ***initModGraphic()***, aquí se hace un llamado a ***saveMSgFileLog()***, mandándole como parámetros una cadena de caracteres (mensaje), y el mensaje de error que se produce de la función ***SDL_GetError()***, podemos cambiar, las líneas de código en las cuales abrimos, guardamos y cerramos el archivo por esta función, claro pasándole los parámetros apropiados.

Ahora veamos el ejemplo, editemos Ejemplo2.cpp. Para este ejemplo se cargara unas imágenes que deberán encontrarse dentro de la carpeta *res* en nuestro proyecto. Ahora ya saber para que es que funciona el código que escribimos con anterioridad, El ejemplo, muestra una imagen de fondo con un, muñequito que por el momento solo lo podemos mover a la derecha, con la tecla del cursor del teclado.

```
//Ejemplo2.cpp: define el punto de entrada de la aplicación de consola.
//© Rubén Dario Orozco 2008

#include "stdafx.h"
#include "RDO_initSDL.h"

int _tmain(int argc, _TCHAR* argv[])
{
      //se inicia SDL con los subsistemas de vido y temporizaadores
      //RDO_initSDL *initSdl = new RDO_initSDL(0);
      RDO_initSDL initSdl(0);

      SDL_Event eventGame;
      SDL_Surface *img,       //para un fondo
                  *fill_d[4];     //simular frames

      //SDL_Rect origen, destino;
      int endgame = 0;

      Uint16 posXFill = 0, //se utiliza para la posición X de una superficie
               posYFill = 0; //posicion y de la superficie
      Uint8 cd=0; //para cambiar el indice del vector
                  //fill_d

      //asignamos un modo de vido a la superficie global
      screen = initSdl.initModGraphic(640, 480, 16);

      SDL_WM_SetCaption("Ejemplo2", NULL);
```

```
	//cargamos una superficie sin transparencia
	img = load_img("res/img.png");

	//copiamos img a screen
	blitting_surface(screen, img, 0, 0);

	fill_d[0] = load_imgT("res/fill_d1.png", 255, 255, 255);
	fill_d[1] = load_imgT("res/fill_d2.png", 255, 255, 255);
	fill_d[2] = load_imgT("res/fill_d3.png", 255, 255, 255);
	fill_d[3] = load_imgT("res/fill_d4.png", 255, 255, 255);

	posYFill = screen->h/2;

	blitting_surface(screen, fill_d[0], 0, posYFill);

	while(endgame == 0){
		SDL_Flip(screen);  //mostramos la pantalla

		//mientra exista un evento
		while(SDL_PollEvent(&eventGame)){
			//Cerrarlaventana
			if(eventGame.type == SDL_QUIT){endgame = 1;}

			//Pulsando una tecla
			if(eventGame.type == SDL_KEYDOWN){
				switch(eventGame.key.keysym.sym){
					case SDLK_RIGHT:
						posXFill++;
						blitting_surface(screen,fill_d[cd],
							posXFill, posYFill);
						cd++;
						if(cd >= 4)
							cd = 0;
				}
			}
		}
	}

	SDL_FreeSurface(img);

	for(int i=0; i<=3; i++){
		SDL_FreeSurface(fill_d[i]);
	}
	SDL_FreeSurface(screen);

	//SDL_Quit();

	printf("Todo ha salido bien.\n");

	return 0;
}
```

Código 10. Ejemplo2.cpp

Expliquemos un poco el código, la línea ***RDO_initSDL initSdl(0)***; nos inicia los subsistemas de video y temporizadores de SDL, esto es por el parametrro que le enviamos, el ventor de superficies ****fill_d[4];*** se utilizara para almacenar 4 superficies, este es una especie de Frame, noes lo mejor pero si lo más a la mano que hay, aquí

guardaremos las representaciones graficas que produciran despues la sensación dee movimiento.

posXFill y ***posYFill***, se utilizaran para manejar las posiciones de x e y de un muñeco que aparece en pantalla, ***cd*** se utiliza como indice "dinamico" para manejar el vector de superficies, la línea ***screen = initSdl.initModGraphic(640, 480, 16);*** es el llamdo ala función que creamos en *RDO_initSDL*, la cual nos inicia un modo de video sin ningun flag, y este modo se lo retorna a la variable global screen, que esta definida en *RDO_UtilSDL*, ***img = load_img("res/img.png");*** en esta líbea de código, llamanos a nuestra función de utileria que creamos anteriormente y cargamos una imagen sin aplicarle ninguna transparencia.

blitting_surface(screen, img, 0, 0); en esta línea, hacemos una operación de blitting, con la función de utilidad que se creo, ***fill_d[0] = load_imgT("res/fill_d1.png", 255, 255, 255);*** cargamos una imagen en el vector ***fill_d***, posicion 0, indicando un color de transparencia definido por nosotros en este caso blanco.

En ***switch(eventGame.key.keysym.sym)***, capturamos el tipo de evento de teclado que se produce, en este caso lo vamos a utilizar para ver si se presiona la tecla fecha derecha del teclado, si esto es así, entoncees actualizamos ***posXFill*** en uno y ejecutamos una operación de blitting, y bueno el while se ejecuta hasta que cerremos la ventana, pero te preguntaras que paso si no hemos finalizado a SDL, pues sencillo, como el objeto ***initSDL***, se ha creado como un objeto no como un apuntador, pues su destrtuctor se llama al finalizazr el programa, y ya sabemos que en el destrructor de este se hacee el llamado a ***SDL_Quit.***

Ahora compilemos y ejecutemos nuestro programa, al ejecutarlo y mover el muñequito con la tecla del cursor fecha derecha, este nos esta dejando una sombra detras de el, ¿porque será?, pues el vector de superficies se esta repintando, en nuestra superficie de video osea ***screen***, esta no se actualiza si no que muestra lo viejo y lo nuevo que hemos pintado, en el siguiente capitulo veremos como mejorar esto, por el momento creo que nos merecemos un descanzo, ha perote dejo un reto hay una función de SDL que nos sirbe para guarda lo que vemos en pantalla en un archivo BMP, esta función es:

SDL_SaveBMP(SDL_Surface *, char *ruta);

El primer parametro es la superficie que deseamos guardar y el segundo parametro es el nombre con el cual guadaremos el archivo.

Create una función para que se la añadas a RDO_UtilSDL, para que nos capture la pantalla en un archivo.

Si por algun motivo he presentado alguna función de SDL en el código que no le haya echo una explicación, es muy probable que encuentres una explicación corta en el apendice D. El proyecto y el código fuente del ejemplo número 2, lo encuntras en Cap 4\Ejemplo2.rar

Creando el diminuto Motor

"Por algo se empieza …"

En el capitulo 4 vimos algunas funciones de SDL para el manejo de superficies, eventos, por el momento creo que es lo más básico para realizar nuestro primer juego de computador, en este capitulo nos centraremos es escribir algunas clase que nos ayudaran a la hora de programar un juego, estas clases serán el corazón de nuestro diminuto motor de desarrollo de videojuegos.

Al terminar el capitulo 4 y ejecutar el ejemplo número 2, vimos un poco el potencial de lo que hemos desarrollado hasta el momento, pero no fue agradable cuando movimos nuestro personaje y nos fue dejando la sombra, detrás de el, una solución seria antes de volcar las superficies a ***screen***, debemos hacerle una limpieza para que borre lo viejo, y nos muestre solamente lo nuevo, esto lo hacemos por medio de esta función ***int SDL_FillRect(SDL_Surface *dst, SDL_Rect *dstrect, Uint32 color)***, lo que hace esta función es, pintar un rectángulo de un color especifico, por ejemplo para borrar el contenido de ***screen***, pintando toda la superficie de negro así:

```
SDL_FillRect(screen, 0, SDL_MapRGB(screen->format, 0, 0, 0));
```

En el Ejemplo2, si colocaramos esta línea de código, antes de llamar a la función ***blitting_surface()***, nos dejaria de mostrar la sombra que va dejando el personaje mientras camina.

```
...

switch(eventGame.key.keysym.sym){
      case SDLK_RIGHT:
            posXFill++;
            SDL_FillRect(screen, 0, SDL_MapRGB(screen->format, 0, 0, 0));
            blitting_surface(screen,fill_d[cd], posXFill, posYFill);
            cd++;

            if(cd >= 4)
                  cd = 0;
```

```
}

. . .
```

Código 11. Cambio en Ejemplo2.cpp

Pero también dejaría de mostrarnos otra cosa importante, nuestro fondo.

Para mejorar un poco esto pensemos en el ciclo de un juego, primero lee las entradas, luego procesa las entradas y por ultimo despliega en pantalla, los cambios que han surgido en todos los gráficos, o mejor en todos los sprites (yo los prefiero llamar actores) que se van a visualizar en ese momento, esto nos da una idea, un juego contiene una colección de sprites, si en nuestro caso, nuestro fondo y el personaje fueran sprites, al momento de pintar en pantalla, pintáramos de nuevo el fondo y al personaje, en total para formar nuestra escena, se nos elimina el problema de la sombra que va dejando el personaje cuando se mueva y nuestro fondo seguiría apareciendo, hemos llegado a una d las soluciones, tal vez sea la menos optima pero funciona.

Empecemos por organizar nuestra ideas, primero que todo ya sabemos que nuestro juego se va a componer de un conjunto de sprites o actors, y que además un juego casi siempre se compone de un ciclo en donde se ejecuta los actores y procesos lógicos del juego, además podemos añadirle, que por medio de la ejecución de un evento, los datos de los actors se actualizan, si pensamos en programación orientada a objetos podemos ver el concepto del juego cómo una clase, la cual tiene como atributos, una colección de actors y un evento, además tiene unas funciones miembro como por ejemplo, la ejecución del ciclo del juego.

Ahora el concepto de sprite o actor, también lo podemos ver, cómo una clase, el cual va a tener como atributos, las posiciones x e y de la pantalla en donde se va a pintar o en donde esta en algún momento, podemos añadirle un identificador (por ejemplo un nombre), una variable que nos guarde el valor de una tecla, y su representación grafica, esta representación grafica también la podemos "encapsular" en el concepto de clase, y podemos colocarle como atributo una superficie y algunas funciones miembro que le indique a la superficie si tiene que pintarse con alguna transparencia o no.

Hemos organizado un poco los conceptos, pues es el momento de empezar a crear las clases, que se han descrito, empecemos por crear un nuevo proyecto llamado *Ejemplo3*, de nuevo crea las carpetas (**lib, res y log**) físicas dentro del directorio del proyecto, ahora copia los archivos RDO_initSDL.h, RDO_initSDL.cpp, RDO_UtilSDL.h, RDO_UtilSDL.cpp, dentro de la carpeta ***lib***, ahora crea la carpeta lógica (filtro) ***lib*** en el proyecto dentro de *Archivos de código fuente* y *Archivos de encabezado* por medio del explorador de soluciones, la figura 9 muestra como que la configuración en el disco (físico) y figura 10 muestra como queda en el proyecto.

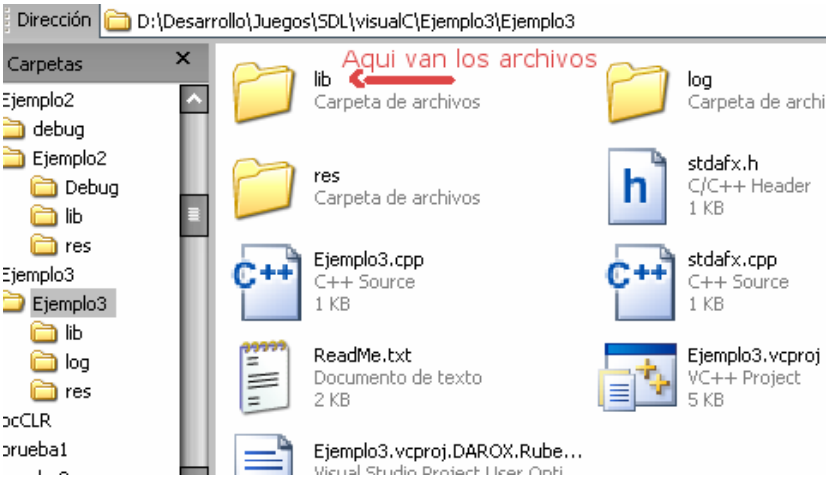

Figura 9. Carpetas físicas.

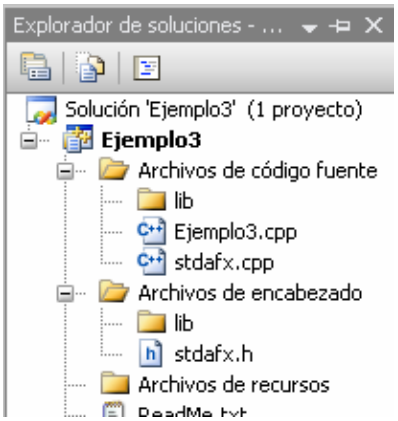

Figura 10. En el proyecto

Ahora agreguemos la librería y la clase que creamos en el proyecto anterior (*Ejemplo2*), esto lo hacemos por medio del explorador de soluciones, damos clic derecho en *Archivos de Código Fuente/lib* y seleccionamos ***Agregar – Elemento Existente…***, en el cuadro de dialogo abrimos la carpeta ***lib*** y seleccionamos los dos archivos con extensión .cpp.

Realizamos el mismo procedimiento para agregar los archivos .h, pero esta vez en *Archivos de encabezado/lib,* la figura 11 muestra el explorador de soluciones y los archivos agregados.

Realicemos el mismo procedimiento que hicimos cuando adicionamos los archivos *RDO_UtilSDL.h* y *RDO_UtilSDL.h* (capitulo 4), pero creamos y agregamos los archivos *RDO_ActorGraphic.h* y *RDO_ActorGraphic.cpp*.

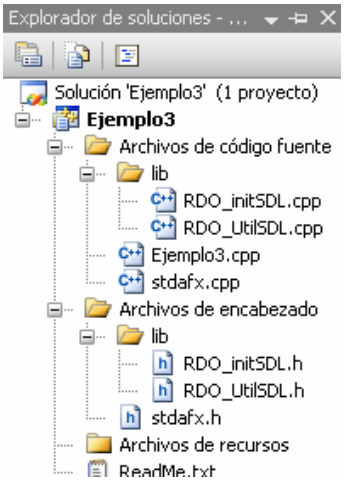

Figura 11. Añadiendo los archivos.

Esta será la clase que representara gráficamente a un actor, veamos el código.

```
//RDO_ActorGraphic.h clase para representación grafica
//de un actor
//(c) Rubén Dario Orozco 2008
#pragma once

#ifndef RDO_ACTORGRAPHIC_H
#define RDO_ACTORGRAPHIC_H

#include "RDO_initSDL.h"

class RDO_ActorGraphic
{
private:
	SDL_Surface *image; //grafico del actor
public:
	RDO_ActorGraphic(void);
	//constructor, se pasa el nombre de un archivo de imagen
	RDO_ActorGraphic(const char *file){image = load_img(file);}
	//constructor, sobrecargado
	//const char *: nombre de archivo de una imagen
	//int: color rojo, int: color verde; int: color azul
	RDO_ActorGraphic(const char *file, Uint8 r, Uint8 g, Uint8 b);
	~RDO_ActorGraphic(void);

	void set_image(SDL_Surface *img){image = img;}
	SDL_Surface *get_image(void){return image;}

	//libera memoria asignada a image
	void freeImage(void){SDL_FreeSurface(image);}

	//volca image, en una superficie
	//SDL_Surface *, en dondee se copiara image
	//int, posicion x; int posicion y, coordenadas
	//donded se copiara image
	void draw(SDL_Surface *des, int x, int y){blitting_surface(des, image, x,
y);}

	//volca, image en la variable global screen
```

```
      //int, posicion x; int y posicion y, coordenadas
      //donde se pegara image
      void draw(int x, int y){blitting_surface(image, x, y);}
};

#endif
```

Código 12. RDO_ActorGraphic.h

```
//RDO_ActorGraphic.cpp implementación de la
//(c) Rubén Dario Orozco 2008

#include "stdafx.h"
#include "RDO_ActorGraphic.h"

RDO_ActorGraphic::RDO_ActorGraphic(void){
}

RDO_ActorGraphic::RDO_ActorGraphic(const char *file, Uint8 r, Uint8 g, Uint8 b){
      image = load_imgT(file, r, g, b);
}

RDO_ActorGraphic::~RDO_ActorGraphic(void){
      printf("destructor de RDO_ActorGraphic\n");
}
```

Código 13. RDO_ActorGraphic.cpp

No hay mucho que explicar en este código, la clase tiene solo un atributo, este representa la superficie de un actor, mejor dicho su representación grafica, para este atributo se le han creado sus métodos *get* y *set*, se ha sobre cargado el constructor, uno que no asigna nada, el segundo se le pasa como parámetro una cadena que representa el nombre de un archivo de imagen, el cual será cargado en ***image***, el tercero, recibe cuatro parámetros el primero es el nombre del archivo de imagen a cargar, el segundo un valor entre 0 y 255 que representa la intensidad del color rojo, el tercero, intensidad del color verde, y el cuarto es la intensidad del color azul.

Lo que si hay que aclarar es que, se adiciono una función a ***RDO_UtilSDL***, esta función es una sobrecarga a la función ***blitting_surface***, esta hace lo mismo, el único cambio es que vuelca una superficie en la superficie global ***screen***, además se han adicionado nuevas variables globales a esta librería, se han adicionado el valor del ancho y alto de pantalla, para acceder a ellos rápidamente, puedes ver los cambios de estos archivos en el código fuente, que vienen adjuntos al libro, los cambios son pocos.

Un cambio importante, es la adición de una variable global que almacena el valor de una tecla presionada del teclado, esta variable es ***SDLKey key***; se utilizara para guardar la tecla que presina el usuario.

Ahora creemos nuestra clase ***Actor***, la cual nos representara a un actor o sprite d una escena de nuestro juego, para crear esta clase realizamos el mismo procedimiento que RDO_ActorGraphic, pero cambiamos el nombre a RDO_Actor, a otra cosa, acuérdate que para compilar debes realizar la adición de las librerías de SDL al vinculador del proyecto (archivos SDLmain.lib, SDL.lib, etc)

```
//RDO_Actor.h clase para un Actor del juego
//(c) Rubén Dario Orozco 2008
#pragma once

#ifndef RDO_ACTOR_H
#define RDO_ACTOR_H

#include "RDO_ActorGraphic.h"
#include <string>

class RDO_Actor
{
protected:
      int posX,         //posicion X del actor
            posY;         //posicion Y del actor

      string idUnique, //identificador string unico
               idGroup;  //identificador de grupo

      RDO_ActorGraphic *actorGraphic;     //representación grafica del actor
public:
      RDO_Actor(void);
      //sobrecarga, se manda como parametro el nombre de una
      //archivo de imagen
      RDO_Actor(const char *file){actorGraphic = new RDO_ActorGraphic(file);}
      //sobrecarga, se manda como parametro el nombre de una
      //archivo de imagen, con un color de transparencia
      RDO_Actor(const char *file, int r, int g, int b){actorGraphic = new
RDO_ActorGraphic(file, r,g,b);}
      ~RDO_Actor(void);

      void set_posX(int x){posX = x;}
      int get_posX(void){return posX;}
      void set_posY(int y){posY = y;}
      int get_posY(void){return posY;}
      void set_idUnique(string s){idUnique.assign(s);}
      string get_idUnique(void){return idUnique;}
      void set_idGroup(string s){idGroup.assign(s);}
      string get_idGroup(void){return idGroup;}

      void set_actorGraphic(RDO_ActorGraphic *s){actorGraphic = s;}
      RDO_ActorGraphic *get_actorGraphic(void){return actorGraphic;}

      //llama a draw de la representación grafica del actor para que se pinte
      //en screen
      void draw(void);

      //llama a draw de la representación grafica del actor para que se pinte
      //en la superficie des
      void draw(SDL_Surface *des);

      //libera memoria de algunos recurso utilizados por la clase
      void freeActor(void);

      //actualización de la parte logica y grafica del actor
      //si se realizaa volcado se hace en screen
      virtual void update();
      //actualización de la parte logica y grafica del actor
```

```
      //si se realizaa volcado se hace en des
      virtual void update(SDL_Surface *des);
};

#endif
```

Código 14. RDO_Actor.h

```
//RDO_Actor.cpp implementación de la clase
//(c) Rubén Dario Orozco 2008

#include "stdafx.h"
#include "RDO_Actor.h"

RDO_Actor::RDO_Actor(void){
}

RDO_Actor::~RDO_Actor(void){
      printf("Destructor de Actor: %s\n", idUnique.c_str());
}

void RDO_Actor::draw(void){
      actorGraphic->draw(posX, posY);
}

void RDO_Actor::draw(SDL_Surface *des){
      actorGraphic->draw(des, posX, posY);
}

void RDO_Actor::freeActor(void){
      actorGraphic->freeImage();
      delete actorGraphic;

      this->~RDO_Actor();
}

void RDO_Actor::update(void){
      //llamamos a draw para pintar la representación grafica
      draw();
}

void RDO_Actor::update(SDL_Surface *des){
}
```

Código 15. RDO_Actor.cpp

Lo que hay que explicar en este código, es que se a añadido un atributo del tipo ***RDO_ActorGraphic***, l cual se encargara de la representación grafica del actor, de pronto te preguntaras por que los atributos los declare bajo la palabra ***protected:***, realice esto para ver la real utilidad en el capitulo 6, por el momento vasta decir que los miembros *protected*, es como una especi de nivel intermedio entre los miembros *private* y *public*, te adelanto algo, se utiliza básicamente cuando, se necesita que una clase derivada pueda acceder a algunos de los atributos de una clase base o padre, estamos hablando de herencia, en el capitulo 6, veremos como utilizamos la herencia y otra característica de la programación el polimorfismo, como se dijo en la introducción no se explicara a profundidad los conceptos de programación como estos, pero si hará corta explicación, sobre estos conceptos.

Si te has dado cuenta hemos definido la función ***update*** como ***vitual***, una función virtual, es una función que nos permite que el compilador determine dinámicamente que función de una clase hija o padre debe ser invocada, esto nos ayudara a utilizar polimorfismo en el capitulo 6.

Ahora miremos la clase ***RDO_Game***, que es el corazón de nuestro juego.

```
//RDO_Game.h clase para administrar el juego
//(c) Rubén Dario Orozco 2008
#pragma once

#ifndef RDO_GAME_H
#define RDO_GAME_H

#include "RDO_initSDL.h"
#include "RDO_Actor.h"
using namespace std;
#include <string>
#include <vector>

class RDO_Game
{
private:
	string name;  //nombre del juego
	vector <RDO_Actor *> vecActor; //vector de actores
	SDL_Event eventGame;	//evento que ocurre segun los dispositivos de
E/S

public:
	RDO_Game(void);
	//sobrecarga, recibe una cadena de caractere
	//nombre del juego
	RDO_Game(char *n){name.assign(n);}
	~RDO_Game(void);

	//ciclo del juego
	virtual void main(void);

	//elimina de memorial, algunos de los atrtibutos, finaliza el juego
	void endGame(void);

	//retorna el atributo name
	string get_name(void){return name;}
	//retorna el atributo name
	void set_name(char *n){name.assign(n);}

	//adiciona un actor al vector de actores (puntero a clase Actor)
	void add_Actor(RDO_Actor *actor){vecActor.push_back(actor);}

	//actualiza la parte logica y grafica
	void update(void);
};

#endif
```

Código 16. RDO_Game.h

```
#include "StdAfx.h"
#include "RDO_Game.h"

RDO_Game::RDO_Game(void){
}

RDO_Game::~RDO_Game(void){
	printf("Destrtuctor de Game\n");
}

void RDO_Game::main(){
	bool exit = false;	//variable control para terminar el juego

	while(!exit){
		//actualizamos la parte logica y grafica
		//de la lista de actores
		update();

		while(SDL_PollEvent(&eventGame)){
			if(eventGame.type==SDL_QUIT)
				exit = true;
			if(eventGame.type == SDL_KEYDOWN){
				switch(eventGame.key.keysym.sym){
					case SDLK_ESCAPE:
						exit = true;
				}
			}
		}
	}
}

void RDO_Game::update(){
	vector <RDO_Actor *>::iterator tmp;

	if(!vecActor.empty())
		for(tmp = vecActor.begin(); tmp!=vecActor.end(); tmp++){
			(*tmp)->update(); //actualizamos el actor
		}

	SDL_Flip(screen);

	if(!vecActor.empty())
		if(vecActor.at(1)->get_posX() <= screen->w - 27)
			vecActor.at(1)->set_posX(vecActor.at(1)->get_posX()+1);
}

void RDO_Game::endGame(){
	vector <RDO_Actor *>::iterator tmp;

	if(!vecActor.empty())
		for(tmp = vecActor.begin(); tmp!=vecActor.end(); tmp++){
			delete (*tmp);
		}
}
```

Código 17. RDO_Game.cpp

Los atributos de la case ***RDO_Game***, primero tenemos un string que nos almacenara el nombre del juego, como segundo tenemos un vector de ***RDO_Actor***, en donde almacenaremos todos los actores del juego, y el tercer parámetro es un tipo SDL_Event, en el cual almacenaremos el evento más actual que ha sucedido en el juego.

Las funciones miembro ***main()***, es la que nos va a representar el ciclo del juego, aquí averiguamos si se desea salir del juego, actualizamos la parte grafica del vector de actores, y capturamos los eventos que pasan en la aplicación, la función ***add_Actor()***, adiciona un actor al final del vector de actores, la función ***update()***, lo que realiza es recorrer cada uno de los actores que hay en el vector de actores, y se llama a la función *update()*, del actor, luego copia el contenido de ***screen*** en pantalla, las sentencias que siguen a continuación actualizan la posición del actor que se encuentra en la posición 1 del vector de actores os actualiza, y la función ***endGame()***, lo que hace es liberar la memoria ocupada por el vector de actores.

Creo que es hora de probar lo que hemos hecho hasta el momento, copia este código en *Ejemplo3.cpp*, compila y ejecuta.

```
// Ejemplo3.cpp: define el punto de entrada de la aplicación de consola.
//

#include "stdafx.h"
#include "lib/RDO_Game.h"
#include "lib/RDO_initSDL.h"

int _tmain(int argc, _TCHAR* argv[])
{
	//iniciamos la libreria SDL
	RDO_initSDL *iniSdl = new RDO_initSDL(0);

	//creamos el objeto que nos manejara el juego
	RDO_Game game("el juego");

	//indicamos las dimenciones de la pantalla
	SCREEN_W = 640, SCREEN_H = 480;
	//indicamos los bit por pixel
	BPP = 16;

	//inicimos el modo de video, y lo asignamos a screen
	screen = iniSdl->initModGraphic(SCREEN_W, SCREEN_H, BPP);

	//ejecutamos el ciclo del juego
	game.main();

	printf("%s\n",game.get_name().c_str());
	game.endGame();
	iniSdl->~RDO_initSDL(); //esto debe ser lo ultimo que se llama
	printf("SDL, ha cargado correctamente\n");
	system("pause");

	return 0;
}
```

Código 18. Una prueba.

Todo ha salido bien, pero las mismas pantallas negras al inicio del texto, pues si todo ha salido bien, es hoja de ver un poco la capacidad de lo que se ha desarrollado hasta el momento.

Ahora copia unas imágenes a la carpeta ***res*** y copia este código en ***Ejemplo3.cpp***.

```
// Ejemplo3.cpp: define el punto de entrada de la aplicación de consola.
//

#include "stdafx.h"
#include "lib/RDO_Game.h"
#include "lib/RDO_initSDL.h"

int _tmain(int argc, _TCHAR* argv[])
{
	//iniciamos la libreria SDL
	RDO_initSDL *iniSdl = new RDO_initSDL(0);

	//creamos el objeto que nos manejara el juego
	RDO_Game game("el juego");

	//indicamos las dimenciones de la pantalla
	SCREEN_W = 640, SCREEN_H = 480;
	//indicamos los bit por pixel
	BPP = 16;

	//creamos los actores, y le adicionamos una imagen
	RDO_Actor *actor0 = new RDO_Actor("res/img.png");
	RDO_Actor *actor1 = new RDO_Actor("res/fill_d1.png");
	RDO_Actor *actor2 = new RDO_Actor("res/fill_d1.png");

	//les establecemos un identificador
	actor0->set_idGroup("fondo");
	actor1->set_idGroup("heroe1");
	actor2->set_idGroup("heroe1");

	//le indicamos una posicion
	actor0->set_posX(0), actor0->set_posY(0);
	actor1->set_posX(100), actor1->set_posY(100);
	actor2->set_posX(100), actor2->set_posY(140);

	//inicimos el modo de video, y lo asignamos a screen
	screen = iniSdl->initModGraphic(SCREEN_W, SCREEN_H, BPP);

	//adicionamos los actores al vector de actores de game
	game.add_Actor(actor0);
	game.add_Actor(actor1);
	game.add_Actor(actor2);

	//ejecutamos el ciclo del juego
	game.main();

	printf("%s\n",game.get_name().c_str());
	game.endGame();
	iniSdl->~RDO_initSDL(); //esto debe ser lo ultimo que se llama
	printf("SDL, ha cargado correctamente\n");
	system("pause");
```

```
	return 0;
}
```

Código 19. Ejemplo3.cpp

Que hay para destacar de este código, primero hemos creado a tres actores, casa uno con una imagen, después les establecemos ciertos valores como un nombre y unas coordenadas iniciales, luego estos tres actores se los adicionamos al vector de actores de la clase ***RDO_Game***, y llamamos al ciclo del juego, pues es hora de compilar y ejecutar a ver que pasa.

Pues aburrido otra vez, la misma imagen de fondo, y el mismo muñequito, de los anteriores ejemplos, pero, uno de los muñequitos se mueve y no esta dejando la sombra detrás de el, hemos resuelto un problema, pero te preguntaras, yo ¿quiero mover al personaje?, lo mas lógico sería que en la clase Actor le definiéramos algún tipo de comportamiento al personaje, pero si lo hacemos, todos los actores que creemos quedaran con el mismo comportamiento.

Una solución seria, crear el comportamiento de cada actor dentro de la función ***update()***, de la clase ***RDO_Game***, tal como se hizo para el ***actor1***, pero no es optimo, pues una solución un poco más optima será la que vamos a ver en el capitulo 6, por el momento, nos merecemos otro descanso, pero te invito a que cacharres un poco con el código que se ha desarrollado, un ejemplo bueno seria que algún actor respondiera, al evento de alguna tecla.

El código fuente del proyecto lo encuentras en *Cap 5/Ejemplo3.rar*.

La Herencia de un Actor

"Empuja, sigue moviéndote"
Thomas Morton

En el anterior capitulo, se desarrollo el corazón del motor, las pocas clases que se crearon, son la estructura sobre la cual se basara el ejemplo de esta capitulo, pude haber sido poco pero en esta sección se mostrara como podemos extender el diminuto motor que has desarrollado.

Para comenzar, retomemos el problema que dejamos en el capitulo 5, el cual era el de crear actores con su propio comportamiento, se vio una posible solución la cual era colocar el código de cada actor dentro de la función ***update()*** de la clase RDO_Game, pero que pasa si tenemos unos 20 actores, para nuestro juego, pues esta función crecería en tamaño de líneas de código, y nos puede causar problemas de mantenimiento y lectura del código.

Para empezar a ver una solución, podemos empezar por comparar tres figuras un rectángulo, un cuadrado y un cuadrilátero, el rectángulo y el cuadrado tienen representaciones graficas diferente, y difiere la forma en que se calcula su área, pero ambos se consideran cuadriláteros, si tomamos el concepto de herencia de la programación orientada a objetos (POO), podemos crear una clase llamada *cuadrilatero*, que contenga como atributos el valor de sus cuatro lados, y una función miembro que calcule el área, podemos crear las clases cuadrado y rectangulo, estas dos clases heredaran de la clase cuadrilatero sus atributos, y la función miembro, pero cada clase implementara el calculo del área de una forma diferente.

Pasa lo mismo con dos actores de un juego, digamos un carro y un árbol, estos actores tienen un comportamiento diferente, el carro se puede mover con cierta libertad en el juego, pero un árbol no, lo que si tienen en común es que no dejan de ser actores en el juego, pues la solución para el problema presentado en el capitulo 5, es crear clases que hereden de la clase ***RDO_Actor***, pero que difieran en comportamiento (por ejemplo como se mueven, si lo podemos manejar o no, etc.).

Miremos esto no un ejemplo, en el se creara un actor que se moverá por la pantalla por medio de las teclas de cursor del teclado y otro que lo hará de manera aleatoria, para comenzar creemos un proyecto llamado *"Ejemplo4"* y realicemos los mismos pasos que se han hecho para crear las carpetas y añadir los archivos que se han desarrollado en los anteriores ejemplos (RDO_UtilSDL, RDO_initSDL, RDO_ActorGraphic, RDO_Actor, RDO_Game).

Empezaremos viendo el código de un actor que llamaremos Nave, crea una nueva clase por medio del explorador de soluciones, nómbrala ***Nave***.

```
//Nave.h representara a un actor del juego
//(c) Rubén Dario Orozco 2008
#pragma once

#ifndef NAVE_H
#define NAVE_H

#include "lib/RDO_Actor.h"

class Nave: public RDO_Actor
{
private:
	bool teletransportacion;
	//un contador para cambiar
	//el valor de teletransportacion
	int contT;

	SDL_Surface *img[3]; //cargara varias img para este actor
	int i;	//no hara cambiar el indice de img
public:
	Nave(void);
	Nave(const char *file, int r, int g, int b){actorGraphic = new
                                RDO_ActorGraphic(file, r,g,b);}
	~Nave(void);

	//se inicializara atributos de clace
	void initAC(void);

	//aqui se definira el comportamiento, propio para
	//este tipo de actor
	void update(void);
};

#endif
```

Código 20. Nave.h

```
#include "StdAfx.h"
#include "Nave.h"
#include <stdlib.h>

Nave::Nave(void){
}

Nave::~Nave(void){
	printf("Destructor de Nave\n");
	SDL_FreeSurface(img[0]);
```

```
	SDL_FreeSurface(img[1]);
	SDL_FreeSurface(img[2]);
}

void Nave::initAC(void){
	img[1] = load_imgT("res/naveL02.png", 255,255,255);
	img[2] = load_imgT("res/naveL03.png", 255,255,255);
	img[0] = actorGraphic->get_image();

	teletransportacion = false;
	contT = 0;
	i = 1;
}

void Nave::update(void){

	//if para mover el actor
	if(teletransportacion == false){
		set_posY(get_posY()+1);

		if((SCREEN_H - 30) <= get_posY()){
			set_posY(10);
		}

		contT++;

		if(contT > 90){
			teletransportacion = true;
			contT = 0;
		}
	}
	else{
		set_posX(30 + rand() % (SCREEN_W - 30));
		set_posY(30 + rand() % (SCREEN_W - 30));

		teletransportacion = false;
	}

	//cambio de imagen
	if(i <= 2){
		actorGraphic->set_image(img[i]);
		i++;
	}
	else{
		i = 0;
	}

	this->draw();
}
```

Código 21. Nave.cpp

La clase ***Nave***, heredara los miembros ***public*** y ***protected*** de la clase ***RDO_Actor***, una clase hija puede acceder directamente a lo miembros de una clase padre si estos se han declarado con ***public*** o ***protected***, en la clase hija se verán los miembros públicos de una clase padre como públicos, y los miembros protegidos se verán como privados, por esta razón declaramos algunos miembros de ***RDO_Actor*** como ***protected***, para poder acceder a ellos en la case ***Nave***.

Otro concepto importante es la declaración de una función ***virtual*** en una clase padre, lo que significa que es dicha función la podemos redefinir en la clase hija, pero esto se puede hacer sin necesidad de declarar una función como ***virtual*** en la clase padre, la diferencia radica en que al de clara la función como virtual en la clase padre y al redefinirla en una clase hija, y al llamar a esta función por medio de un apuntador de la clase padre, el compilador identifica dinámicamente en tiempo de ejecución que función debe ejecutarse si la de la clase padre o la clase hija, esta capacidad se da porque los objetos de una clase hija se puede considerar como objetos de una clase padre, pero no al contrario, esto se conoce como ***polimorfismo,*** si no ha que dado claro lo veremos cuando terminemos el ejemplo.

En la clase ***Nave***, hemos creado un vector de superficies (*SDL_Surfacec *img[3]*), en este se cargaran diferentes imágenes, que luego por medio del atributo *actorGraphic* iremos cambiando de imagen, esto se ve en la función ***update()***, de la clase Nave, esto se hace con el fin de representar una animación de propulsión en la nave.

Ahora creemos la clase ***Heroe*** que también heredara dee ***RDO_Actor.***

```
//Heroe.h representara a un actor del juego
//(c) Rubén Dario Orozco 2008
#pragma once

#ifndef HEROE_H
#define HEROE_H

#include "lib\RDO_Actor.h"

class Heroe : public RDO_Actor
{
private:
	int dir; //para identificar una dirección
			//0-abajo, 1-derecha, 2-izquierda, 3-abajo
	int inImg;//para identificar el img a mostrar
	SDL_Surface *img[16]; //cargara varias img para este actor

public:
	Heroe(void);
	Heroe(const char *file, int r, int g, int b){actorGraphic = new
RDO_ActorGraphic(file, r,g,b);}
	~Heroe(void);

	///se inicializara atributos de clace
	void initAC(void);

	//aqui se definira el comportamiento, propio para
	//este tipo de actor
	void update(void);
};

#endif
```

Código 22. Heroe.h

```
#include "StdAfx.h"
```

```
#include "Heroe.h"

Heroe::Heroe(void){
}

Heroe::~Heroe(void){
	printf("destrructor de Heroe\n");

	for(int i=0; i<=15; i++)
		SDL_FreeSurface(img[i]);
}

void Heroe::initAC(void){

	//imagen de frente, abajo
	img[0] = actorGraphic->get_image();
	img[1] = load_imgT("res/fill_f2.png", 255,255,255);
	img[2] = load_imgT("res/fill_f3.png", 255,255,255);
	img[3] = load_imgT("res/fill_f4.png", 255,255,255);
	//imagen dir derecha
	img[4] = load_imgT("res/fill_d1.png", 255,255,255);
	img[5] = load_imgT("res/fill_d2.png", 255,255,255);
	img[6] = load_imgT("res/fill_d3.png", 255,255,255);
	img[7] = load_imgT("res/fill_d4.png", 255,255,255);
	//imagen dir izquierda
	img[8] = load_imgT("res/fill_i1.png", 255,255,255);
	img[9] = load_imgT("res/fill_i2.png", 255,255,255);
	img[10] = load_imgT("res/fill_i3.png", 255,255,255);
	img[11] = load_imgT("res/fill_i4.png", 255,255,255);
	//imagen de espalda, arriba
	img[12] = load_imgT("res/fill_e1.png", 255,255,255);
	img[13] = load_imgT("res/fill_e2.png", 255,255,255);
	img[14] = load_imgT("res/fill_e3.png", 255,255,255);
	img[15] = load_imgT("res/fill_e4.png", 255,255,255);

	dir = inImg = 0;
}

void Heroe::update(void){

	switch(key){
		case SDLK_DOWN:
			set_posY(get_posY()+1);

			inImg++; //incrmento para cambiar de imagen

			//si sale del rango de img abajo
			if(inImg > 3){
				inImg = 0;
			}

			//si viene de otra dirección
			if(dir != 0){
				inImg = 0;
				dir = 0;
			}

			//actualizao el grafico
```

```
                actorGraphic->set_image(img[inImg]);

                break;
            case SDLK_RIGHT:
                set_posX(get_posX()+1);

                inImg++; //incrmento para cambiar de imagen

                //si sale del rango de img abajo
                if(inImg > 7){
                    inImg = 4;
                }

                //si viene de otra dirección
                if(dir != 1){
                    inImg = 4;
                    dir = 1;
                }

                actorGraphic->set_image(img[inImg]);
                break;
            case SDLK_LEFT:
                set_posX(get_posX()-1);

                inImg++; //incrmento para cambiar de imagen

                //si sale del rango de img abajo
                if(inImg > 11){
                    inImg = 8;
                }

                //si viene de otra dirección
                if(dir != 2){
                    inImg = 8;
                    dir = 2;
                }

                actorGraphic->set_image(img[inImg]);
                break;
            case SDLK_UP:
                set_posY(get_posY()-1);

                inImg++; //incrmento para cambiar de imagen

                //si sale del rango de img abajo
                if(inImg >= 15){
                    inImg = 12;
                }

                //si viene de otra dirección
                if(dir != 3){
                    inImg = 12;
                    dir = 3;
                }

                actorGraphic->set_image(img[inImg]);
                break;
        }
```

```
	this->draw();
}
```

Código 23, Heroe.cpp

Esta clase es similar a ***Nave***, se diferencia en que el vector de superficies es mayor, y su función ***update()***, tiene otro comportamiento programado, este simula el movimiento de un personaje. El atributos ***dir***, indicara la dirección en la se mueve el actor (abajo, derecha, izquierda, arriba), el atributo ***inImg***, se utiliza para indicar el índice del vector ***img***, lógicamente es para sacar una superficie del vector, correspondiente a la dirección de movimiento. Para hacer uso de los eventos del teclado se consulta la variable global ***key***, definida en la librería ***RDO_UtilSDL***, para esto se realizo una modificación en la función ***main()*** de la clase ***RDO_Game***, en realidad se realizaron dos modificaciones, la primera fue declarar al destructor de clase como función virtual (*virtual ~RDO_Game();*), y la segunda se realizo en el archivo ***RDO_Game.cpp*** en la función ***main(),*** continuación se muestra esta modificación.

```
void RDO_Game::main(){
	bool exit = false;	//variable control para terminar el juego

	while(!exit){
		//actualizamos la parte logica y grafica
		//de la lista de actores
		update();

		while(SDL_PollEvent(&eventGame)){
			if(eventGame.type==SDL_QUIT)
				exit = true;
			if(eventGame.type == SDL_KEYDOWN){
				switch(eventGame.key.keysym.sym){
					case SDLK_ESCAPE:
						exit = true;
				}

				//este es el cambio
				key = eventGame.key.keysym.sym;
			}
		}
	}
}
```

Código 24. Cambio en RDO_Game

Ahora escribamos el código para *Ejemplo4.cpp*, el cual nos definirá el archivo de ejecución de nuestro programa.

```
// Ejemplo4.cpp: define el punto de entrada de la aplicación de consola.
//(c) Rubén Dario Orozco 2008

#include "stdafx.h"
#include "lib/RDO_Game.h"
#include "lib/RDO_initSDL.h"
#include "Nave.h"
#include "Heroe.h"
```

```
int _tmain(int argc, _TCHAR* argv[])
{
	//iniciamos la libreria SDL
	RDO_initSDL *iniSdl = new RDO_initSDL(0);
	//creamos el objeto que nos manejara el juego
	RDO_Game game("el juego");

	//indicamos las dimenciones de la pantalla
	SCREEN_W = 640, SCREEN_H = 480;
	//indicamos los bit por pixel
	BPP = 16;

	//inicimos el modo de video, y lo asignamos a screen
	screen = iniSdl->initModGraphic(SCREEN_W, SCREEN_H, BPP);

	//creamos los actores, y le adicionamos una imagen
	RDO_Actor *actor0 = new RDO_Actor("res/img.png");
	Nave *actNave = new Nave("res/naveL01.png", 255, 255, 255);
	Heroe *actHeroe = new Heroe("res/fill_f1.png", 255, 255, 255);

	//les establecemos un identificador
	actor0->set_idUnique("fondo");
	actNave->set_idUnique("Nave");
	actHeroe->set_idUnique("Heroe");

	//le indicamos una posicion
	actor0->set_posX(0), actor0->set_posY(0);

	actNave->set_posX(100), actNave->set_posY(140);
	actNave->initAC(); //inicializo al actor actNave

	actHeroe->set_posX(SCREEN_W/2), actHeroe->set_posY(SCREEN_H/2);
	actHeroe->initAC(); //inicializo al actor actHeroe

	//adicionamos los actores al vector de actores de game
	game.add_Actor(actor0);
	game.add_Actor(actNave);
	game.add_Actor(actHeroe);

	//ejecutamos el ciclo del juego
	game.main();

	printf("%s\n",game.get_name().c_str());
	game.endGame();
	iniSdl->~RDO_initSDL(); //esto debe ser lo ultimo que se llama
	printf("SDL, ha cargado correctamente\n");
	system("pause");

	return 0;
}
```

Código 25. Ejemplo4.cpp

Si tedas cuenta este código no se diferencia mucho del código de "*Ejemplo3.cpp*", la unica diferencia es que se ha creado dos objetos, uno de la clase ***Nave*** y el otro de la clase ***Heroe***.

Si te has dado cuenta, cuando vamos a adicionar los actores en el vector de referencias ***RDO_Actor*** adicionamos de la clase ***RDO_Game***, le estamos pasando objetos diferentes de la clase ***RDO_Actor*** o sea un objeto de ***Nave*** y otro de ***Heroe***, y no genera ningún error, esta es la capacidad que se mencionaba más arriba, un objeto de una clase hija se puede considerar como un objeto de la clase padre, pero nunca al revés.

Ahora como hace para ejecutar la función correcta ***update()*** de cada clase, pues como se menciono arriba por medio de polimorfismo, al ser el vector de la clase ***RDO_Game*** un arreglo de referencias del tipo ***RDO_Actor*** (clase padre), y al haber declaro la función ***update()*** como virtual en la clase padre, el programa es capas de identificar dinámicamente en tiempo de ejecución que función se de invocar.

Bueno creo que hemos llegado al final de este texto, al motor todavía le falta mucho, por ejemplo la forma en la que cargamos varias imágenes para un mismo actor no es la más óptima, en ves de crear un vector de superficies, se debería crear un vector de ***RDO_ActorGraphic***, otra cosa es que los actores no tienen la capacidad para detectar si ha colisionado con otro, bueno falta mucho, pero creo que el objetivo del texto era mostrarte un poco el desarrollo de videojuegos de una forma rápida y más o menos organizada, no puedo decir adiós puede ser que nos encontremos en otra edición de este texto, pero lo que espero es que dejes volar tu imaginación y practiques con el diminuto motor y que lo mejores.

Si de pronto quieres saber noticias de una nueva edición de este texto o de otros que tengan que ver sobre este tema mándame un correo a drincast@hotmail.com, y si en algún momento desarrollo algo nuevo referente al tema te avisare, ha también si tienes algún proyecto entre manos y no tienes con quien emprenderlo puedes avisarme, lo que si aclaro es que no soy un experto, tengo muy poca experiencia (casi un 5% aprox) desarrollando juegos, pero las ganas si que las tengo, bueno no me queda más que agradecerte, por dedicarle un rato de tu tiempo a leer este texto y de disfrutar de un buen videojuego.

Hasta la próxima…

Apéndice A

Instalación y configuración de SDL en DevC++ y Visual C++ Express 2005

Para comenzar, la versión que tengo de DevC++ es la 4.9.9.2 y la versión de visual c++ es la express 2005, estos dos entornos de desarrollo se pueden descargar, sin ningún costo. La instalación de DevC++ es sencilla, la del visual c++ es un poco más complicada, para su instalación necesitas .Net Framework 2.0 y Visual Web Developer (en si es el entorno grafico de desarrollo), estos también se pueden descargar de la pagina de Microsoft.

Ahora lo que necesitas, son las librerías de desarrollo de SDL, estas se pueden descargar de la pagina oficial de SDL http://www.libsdl.org en la sección de descargas, para el SO Windows son SDL-devel-1.2.13-mingw32.tar.gz para DevC++ y SDL-devel-1.2.13-VC8.zip para visual c++, para SO Linux es SDL-devel-1.2.13-1.i386.rpm.

Teniendo esto procedemos a descargar también los ejecutable o binarios de esta librería, para Windows es SDL-1.2.13-win32.zip y Linux SDL-1.2.13-1.i386.rpm.

A.1 Agregar SDL a DevC++

Comencemos por descomprimir el archivo del ejecutable que se encuentra en el comprimido SDL-1.2.13-win32.zip, hay dos archivos un txt y una librería dll (SDL.dll), este archivo dll lo vamos a copiar en la ruta C:\Windows\system32 esto es para Windows xp, para otra versión es en C:\Windows\system.

Ahora descomprimimos el contenido del archivo SDL-devel-1.2.13-mingw32.tar.gz, dentro de la carpeta SDL-1.2.13 hay muchos archivos pero los que nos interesan son dos carpetas, la carpeta lib y incluye.

Copia el contenido de la carpeta lib en donde tenemos instalado el DevC++, en mi caso e en C:\Dev-Cpp\lib, ahora revisa la carpeta include, mira que dentro de ella se encuentre solo una carpeta, llamada SDL, dentro de esta carpeta deben estar todos los archivos .h, si esos archivos se encuentran por fuera, crea una carpeta nombrada SDL y mueve todos los archivos a ella. Listo, copia la carpeta SDL en la ruta C:\Dev-Cpp\include, es muy importante que los archivos con extensión .h se encuentren dentro de la carpeta SDL anterior mente nombrada, si esto no es así, le aseguro que no compilara los programas.

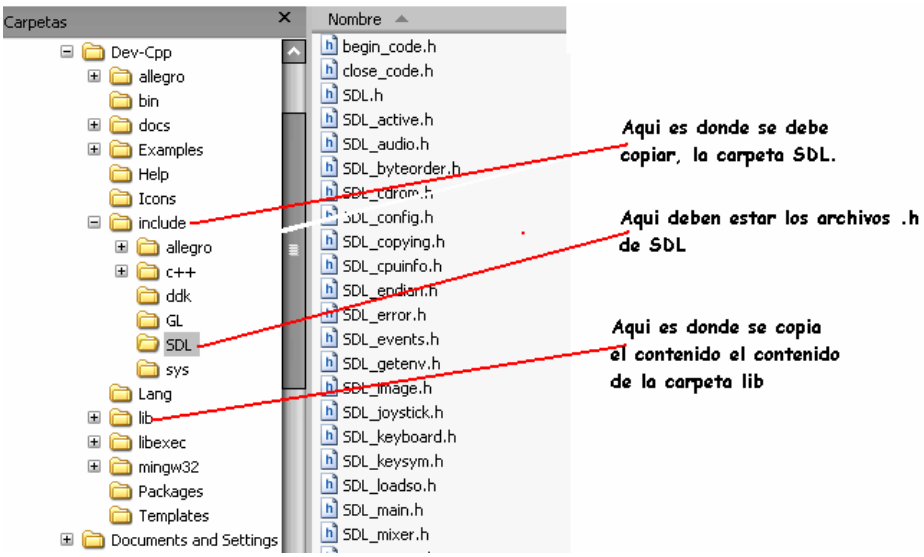

Figura A.1. Archivo SDL en DevC++

A.1.1 Configurando el Linker

Iniciamos DevC++ y creamos un proyecto nuevo del tipo ***Console Application***, y guardamos el archivo .cpp que se genera yo lo nombre main.cpp, luego debemos definir las opciones de compilación del proyecto, esto lo hacemos en ***Proyecto>>Opciones de Proyecto***, seleccionamos la pestaña ***Argum…***, en la sección ***Enlazador(Linker)*** escribimos ***-lmingw32 -lSDLmain –lSDL***, y presionamos el botón aceptar.

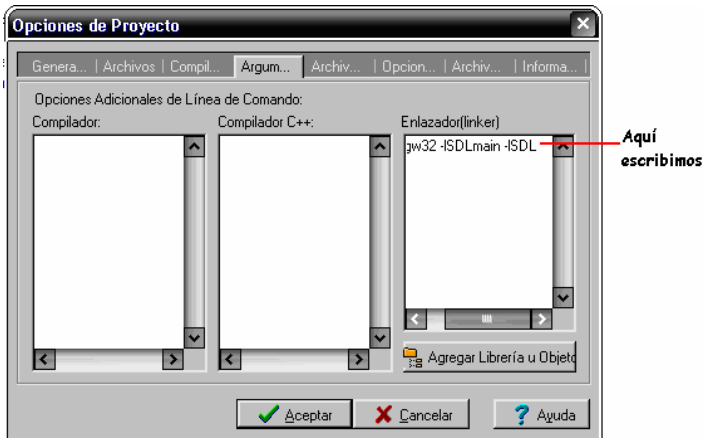

Figura A.2. Configurando el Enlazador

A.1.2 Añadiendo librerías auxiliares

Listo, ahora añadiremos las librerías auxiliares, existen muchas, para manejo de imágenes, manejo de sonidos, manejo de texto, ayuda con OpenGL, etc., yo he instalado 3, una para manejo de imágenes, otra para sonido y una para el manejo de texto (Fonts), la de imágenes se llama SDL_image, la de sonido SDL_mixer, la de texto SDL_ttf.

Estas librerías las encuentras en la pagina oficial, en la sección de librerías, y en el buscador escribes el nombre y vas al vinculo para descargar la librería, yo explicara la

inclusión de dos SDL_image y SDL_mixer, creo que para el resto de librerías es lo mismo, así sucede para la librería SDL_ttf.

Para SDL_image descarga el archivo SDL_image-devel-1.2.6-VC8.zip y para SDL_mixer SDL_mixer-devel-1.2.8-VC8.zip, bueno esto es para el sistema Windows, para Linux también vienen en formato rpm.

Primero comencemos por descomprimir SDL_image-devel-1.2.6-VC8.zip, dentro de la capeta SDL_image-1.2.6, hay dos carpetas, la carpeta lib y include, dentro de la carpeta lib seleccionamos todos los archivos con extensión dll y los copiamos en C:\windows\system32. Ahora dentro de la carpeta include se encuentra el archivo SDL.image.h, este lo copiamos a C:\Dev-Cpp\include\SDL.

El mismo procedimiento es para SDL_mixer, el contenido de lib (solamente archivos dll) al C:\windows\system32, y el contenido de include (SDL_mixer.h) a C:\Dev-Cpp\include\SDL

Otra vez debemos configurar las opciones de compilación, nos vamos a nuestro proyecto el que creamos anteriormente con DevC++, lo abrimos y en tramos a las ***opciones de proyecto***, en la misma pestaña ***Argum…***, y en la sección ***Enlazador(Linker)*** escribimos o mejor adicionamos esto:

-lSDL_image -lSDL_mixer.

O si lo prefieres remplaza el texto por esto:

-lmingw32 -lSDLmain –lSDL -lSDL_image -lSDL_mixer.

A.2 Visual C++ 2005 Express Edition

Comencemos por descomprimir el archivo del ejecutable que se encuentra en el comprimido SDL-1.2.13-win32.zip, hay dos archivos un txt y una librería dll (SDL.dll), este archivo dll lo vamos a copiar en la ruta C:\Windows\system32 esto es para Windows xp, para otra versión es en C:\Windows\system.Teniendo ya instalado expresare a explicar como incluir la librería SDL a DevC++.

Ahora descomprimimos el archivo SDL-devel-1.2.13-VC8.zip, esto nos descomprime la carpeta SDL-1.2.13, recomiendo colocar esta carpeta en C: con todo su contenido, dentro de esta carpeta hay dos carpetas, la carpeta lib y la carpeta include dentro de esta vamos a crear una nueva carpeta llamada **SDL** en mayúsculas, y metemos en ella todos los archivos que se encuentran en la carpeta incluye.

Listo el siguiente paso es abrir visual C++, creamos un proyecto nuevo tipo Aplicación de consola Win32, nos aparece un asistente y damos click en finalizar, ahora vallamos al menú ***Proyecto>>Preferencias…*** abrimos propiedades de ***configuración>>C/C++>>Generación de código*** y en la opción de Biblioteca en tiempo de ejecución seleccionamos DLL multiproceso (/MD), presionamos Aplicar y luego Aceptar (no es necesario este paso por defecto, viene la opción establecida en DLL de depuración multiproceso (/MDd)).

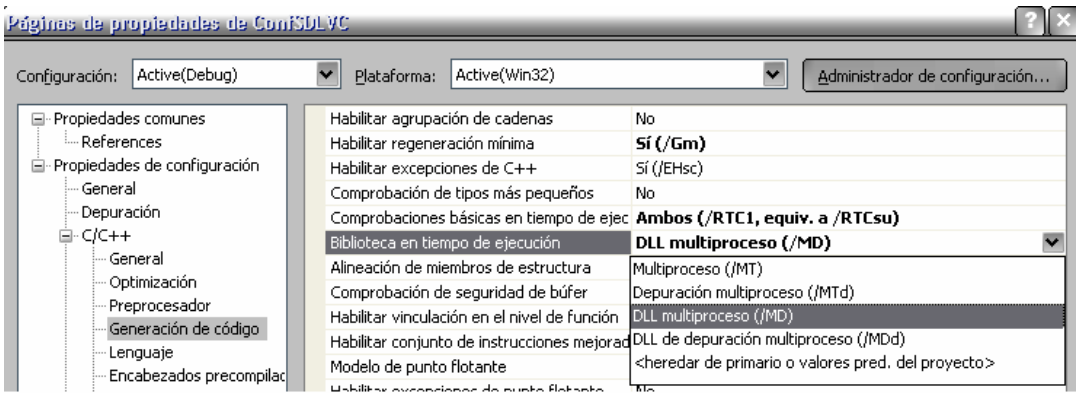

Figura A.3. Generación de código.

Ahora seleccionamos el menú ***Herramientas>>opciones*** nos paramos en ***Proyectos y Soluciones*** debajo de "*Mostrar directorios para:*" seleccionamos *Archivos de inclusión,* damos clic en el icono de carpeta y luego presionamos el botón que aparece a la derecha (Figura 7), y seleccionamos la carpeta include de SDL en mi caso en donde coloque esta carpeta fue en C:\SDL-1.2.13\include y presionamos el botón abrir.

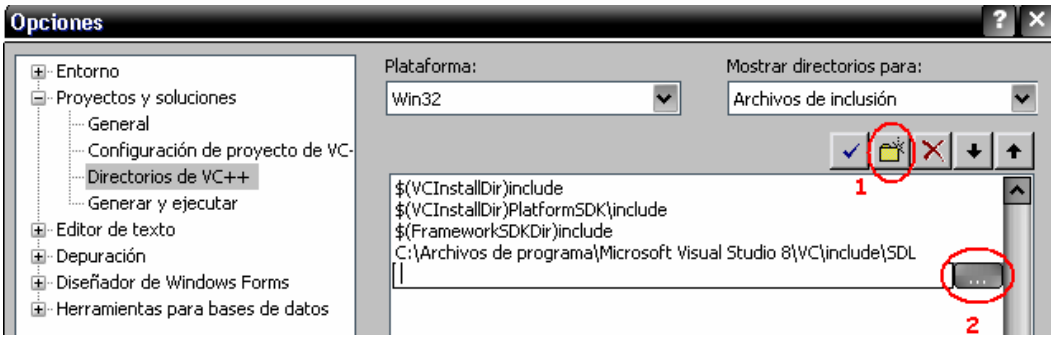

Figura A.4. Adicionado archivos incluye.

Ahora de nuevo en la opción "*Mostrar directorios para:*" seleccionamos *Archivos de biblioteca*, hacemos el mismo procedimiento pero esta vez añadimos la carpeta lib y presionamos Aceptar.

A 2.1 Configurando el Linker

De nuevo seleccionamos el menú ***Proyecto>>Preferencias…*** seleccionamos la opción de ***Vinculador>>Linea de comando*** y en abajo de opciones adicionales escribimos las librerías que necesitamos en esta caso SDL.lib y SDLmain.lib (Figura 8).

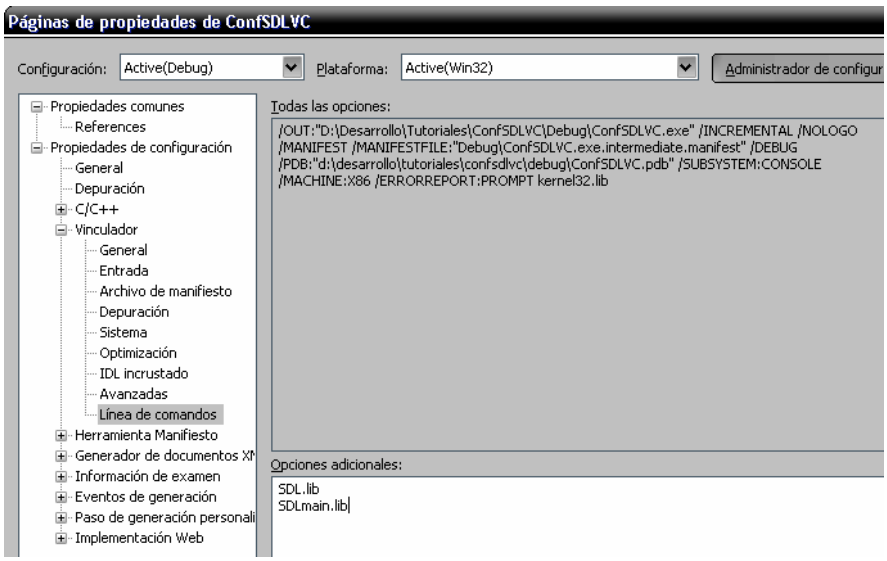

Figura 8. Línea de comandos.

A.2.2 Añadiendo librerías auxiliares

Ahora agreguemos las librerías auxiliares para SDL, como lo explique en la sección de DevC++, solo adicionare SDL_image y SDL_mixer.

Para SDL_image descarga el archivo SDL_image-devel-1.2.6-VC8.zip y para SDL_mixer SDL_mixer-devel-1.2.8-VC8.zip, bueno esto es para el sistema Windows, para Linux también vienen en formato rpm.

Primero comencemos por descomprimir SDL_image-devel-1.2.6-VC8.zip, dentro de la carpeta SDL_image-1.2.6, hay dos carpetas, la carpeta lib y include, dentro de la carpeta lib seleccionamos todos los archivos con extensión dll y los copiamos en C:\windows\system32, hay un archivo nombrado SDL_image.lib, este archivo lo debemos copiar en donde tenemos nuestra librería de SDL, en mi caso C:\SDL-1.2.13\lib. Ahora dentro de la carpeta include se encuentra el archivo SDL.image.h, este lo copiamos a C:\SDL-1.2.13\include\SDL.

El mismo procedimiento es para SDL_mixer, el contenido de lib (solamente archivos dll) al C:\windows\system32, y el archivo SDL_mixer.lib en C:\SDL-1.2.13\lib, el contenido de include (SDL_mixer.h) a C:\SDL-1.2.13\include\SDL.

Listo, ahora configuremos el Vinculador, abrimos el proyecto anteriormente creado, seleccionamos el menú ***Proyecto>>Propiedades…*** seleccionamos la opción de ***Vinculador>>Linea de comando*** y en abajo de opciones adicionales escribimos las librerías que necesitamos en esta caso SDL_image.lib y SDL_mixer.lib, ahora probemos utilizar estas librerias.

Apéndice B

Valores para SDLKey y mod

SDLKey	Tecla
SDLK_ESCAPE	Esc
SDLK_SPACE	Tecla espacio
SDLK_0, SDLK_1, …, SDLK_9'	Tecla 0, Tecla 1, .., Tecla 9
SDLK_a, SDLK_b, SDLK_c, …, SDLK_z	Tecla a, tecla b, tecla c, …, tecla z
SDLK_KP0, SDLK_KP1, …, SDLK_KP9	Teclado númerico, 0, 1, …, 9
SDLK_UP	Tecla cursor arriba
SDLK_DOWN	Tecla cursor abajo
SDLK_RIGHT	Tecla cursor derecha
SDLK_LEFT	Tecla cursor izquierda
SDLK_F1, SDLK_F2, SDLK_F3, …, SDLK_F12	Teclas de function: F1, F2, F3, …, F12
SDLK_RSHIFT, SDLK_LSHIFT	Tecla shift derecha, tecla shift izquierda
SDLK_RCTRL, SDLK_LCTRL	Tecla CRTL derecha, tecla CRTL izquierda
SDLK_RALT, SDLK_LALT	Tecla ALT derecha, tecla ALT izquierda

Tabla B.1. Códigos de SDLKey

KMOD_CAPS	Capslock está pulsado
KMOD_LCTRL	Control izquierdo está pulsado
KMOD_RCTRL	Control derecho está pulsado
KMOD_RSHIFT	Shift derecho está pulsado
KMOD_LSHIFT	Shift izquierdo está pulsado
KMOD_RALT	Alt derecho está pulsado

Tabla B.2. Valores que pueden tomar el parámetro ***mod***.

En estas tablas no estan presntados todos los valores que pueden tomar los parámetros ***sys*** y ***mod*** de la estructura ***SDL_keysym***.

Apéndice C

Conceptos de gráficos 2D

C.1 Sprites

Este concepto identifica a un objeto en nuestro juego, como una nave, un animal, etc., no es que solamente sea el grafico, un sprite representa más cosas, como la posición del objeto, vidas, algún comportamiento, etc., pero lo más básico es la representación grafica y unas coordenadas de posición.

C.2 Animación de Sprites, Frames y Frame Rate

Para animar un Sprite, este tiene que estar cambiando su representación grafica, por ejemplo, como se camina un personaje en pantalla, el objeto sprite para realizar una animación debe tener asociado una serie de imágenes, a esto se le conoce como *Frames*. Entonces para una animación de un sprite necesitamos un conjunto de Frames (conjunto de representaciones graficas), siendo su mínima unidad un *Frame* (un grafico), para representar una animación debemos dibujar un frame diferente cada cierto tiempo, a esto se le conoce como F*rame Rate*, que en si es el número de frames dibujado o visualizado en un periodo de tiempo.

Otro concepto importante es el sprite sheet, esto es una archivo de imagen en el cual guardamos todos los frames (gráficos) que van representar gráficamente a nuestro sprite en una animación determinada, un pequeño ejemplo es este.

Figura C.1. sprite sheet

C.3 Blitting

Se define como una operación de copiado de bloque de bytes, por ejemplo en SDL es el proceso que realizamos de copiar una superficie en otra superficie por medio de la función

SDL_BlitSurface(), en terminos gererales este es el proceso de renderizado o dibujado en un fondo.

C.4 Clipping

Es definir un area rectangular, para una superficie, el la cual solo se prodra realizar operaciones graficas como dibujar, si intentamos dibujar por fuera de esta area, ese dibujo no se tendra en cuenta.

Apéndice D

Algunas funciones de SDL

Este apéndice, se explican algunas de las funciones de SDL.

- ***int SDL_FillRect(SDL_Surface *dst, SDL_Rect *dstrect, Uint32 color),*** dibuja un rectángulo en la superficie *dst* , con un color determinado, el primer parámetro es la superficie en la cual vamos a pintar el rectángulo el segundo parámetro es el rectángulo (las dimensiones del rectángulo), y el tercer parámetro es el color con el cual vamos a rellenar el rectángulo (se pasa por medio de la función *SDL_MapRGB*).

- ***int SDL_Flip(SDL_Surface *screen)***, Se utiliza cuando esta activado el manejo de double buffer, sirve para mostrar la parte grafica de una superficie de video en la pantalla, recibe como parámetro una superficie de video, la que se desea desplegar en pantalla.

- ***void SDL_UpdateRect(SDL_Surface *screen, Sint32 x, Sint32 y, Sint32 w, Sint32 h)***, se utiliza para mostrar en pantalla la parte grafica de una superficie, es parecida a *SDL_Flip()*, pero esta se utiliza cuando no se activado el doble buffer, el primer parámetro es la superficie de video a mostrar en pantalla, el segundo y tercero son la posición x e y en donde vamos a colocar la superficie, y el cuarto y quinto parámetro es el ancho y alto.

- ***SDL_DisplayFormat(SDL_Surface *sur)***, se utiliza para transformar una superficie en el formato de pantalla, es útil para ayudar a agilizar el blitting, recibe como parámetro la superficie a la cual de vamos a cambiar el formato, y retorna una superficie con el formato convertido al formato de pantalla.

Apéndice E

Recursos.

IDE's de Desarrollo

Dev-C++: http://www.blodshed.net

Microsoft Visual C++ Express 2005: http://www.microsoft.com/express/download/

Graficos y Audio

Sobre este tema quisiera dar una opinión, no es mucha la información que tengo, aparte de algunos programas gráficos, seria bueno para el que este interesado, que nos crearemos algún foro en donde podamos, decir de donde bajar recursos gráficos y de audio, claro siempre y cuando estén bajo una licencia Creative Commons o GPL, lo que quiero decir con recursos son archivos bmp, png, etc y wav, mid, mp3, etc., en realidad para crear un buen videojuego, se necesitan unos buenos gráficos, seria interesante contar con gráficos ya elaboraros y archivos de audio que podamos utilizar en nuestros juegos sin tener algún problema legal por utilizarlos en nuestros programas, quien este interesado puede escribirme a mi corre electrónico, yo por mi parte dejo a disposición lo gráficos que he realizado para los ejemplos para que el que quiera utilizarlos en sus aplicaciones.

Blender, Dibujo 3D, http://www.blender.org

Inkscape: Dibujo vectorial, http:// www.injscape.org

GIMP: edición de gráficos, http://www.gimp.org

Bibliografía

H. M. Deitel, P. J. Deitel, C++ Cómo Programar, Segunda Edición, Pretice Hall, México, 1999.

Roberto Albornoz Figueroa, Desarrollo de Videojuegos ¿Como empezar en el Desarrollo de Videojuegos? 2006 – 2007.

Roberto Albornoz Figueroa, Desarrollo de Videojuegos Conceptos Básicos para Desarrollo de Videojuegos 2D, 2006 -2007.

Marius Andra, GFX with SDL Lesson 1: Getting started with SDL

Por Jorge García (Bardok), Curso de introducción a OpenGL (v1.0)

Referencias Web

http://www.libsdl.org, (Simple DirectMedia Layer library) sección de documentación.