SDL: Primeros pasos

Programación de Vídeo Juegos

Fco. Javier Pérez Pacheco
Javielinux (http://www.javielinux.tk)

javi.pacheco@terra.es

SDL: Primeros pasos: Programación de Vídeo Juegos
por Fco. Javier Pérez Pacheco

Tabla de contenidos

Capítulo 1. Introducción

1.1. Qué encontraremos en este artículo

Este es el primero de una serie de artículos que pretendo escribir de creación de Video Juegos en SDL. Pretendo que al final de los artículos el usuario sea capaz de tener una base para poder empezar a desarrollar su propio juego, aunque sólo una base, lo demás corre de su cuenta.

El lector debería de tener una base de C y C++, y conocimientos de Programación Orientada a Objetos para leer este documento, ya que los ejemplos serán escritos en este lenguaje utilizando POO.

Me gustaría decir que no soy un gran programador y que llevo muy poco tiempo trabajando con estas librerías y lo único que pretendo es explicar un poco por encima de una manera fácil y sencilla a base de ejemplo como trabajar con ellas. Con esto lo que quiero decir que tiene que ser el lector el primero que debe de buscar información por su cuenta, aquí sólo expondré los pequeños avances que he conseguido para que sirvan como base a todo el que quiera leerlo.

Al final de este capítulo seremos capaces de compilar un programa usando las librerías SDL y podremos trabajar básicamente con imágenes.

1.2. ¿Qué es SDL?

SDL son unas librerías creadas por Locky Games que facilitan el desarrollo de Vídeo Juegos. Podemos utilizarlas con muchos lenguajes como Perl, C, C++, PHP y muchos más, aunque en este documento vamos a trabajar con C y C++. Con estas librerías podremos controlar los gráficos, sonidos, thread, red y todo lo necesario para construir nuestro juego.

¿Por qué SDL?. Muy fácil, porque son las únicas librerías que conozco por el momento para crear juegos. Además son multi-plataforma, por lo que podremos trabajar en el sistema operativo que queramos, el mío por ejemplo es Linux, y luego muy fácilmente podremos compilar nuestro juego en cualquier otra plataforma.

Toda la información sobre estas librerías se pueden encontrar en su sitio Web http://www.libsdl.org. En ellas podréis encontrar una descripción muy completa de cada una de sus funciones, juegos realizados en SDL, librerías extras, y mucho más.

1.3. Y ¿Cómo vamos a aprender SDL?

La mejor para aprender es haciendo ejemplos. Para ello doy por hecho que el usuario tiene conocimientos de C y C++ y durante todos los artículos vamos a trabajar en el desarrollo de un juego de aviones. Es la forma más fácil de aprender, creo, así que todos los ejemplos que veamos durante los artículos estarán orientados a este tipo de juego y al final tendremos un juego sencillo que podremos enseñarle a nuestros amigos. Será un juego muy sencillo, así que tampoco llaméis ahora a todos vuestros colegas diciendole que estáis haciendo un gran juego, porque luego los váis a decepcionar ;-).

Capítulo 2. Instalación de SDL

2.1. Instalación en Linux

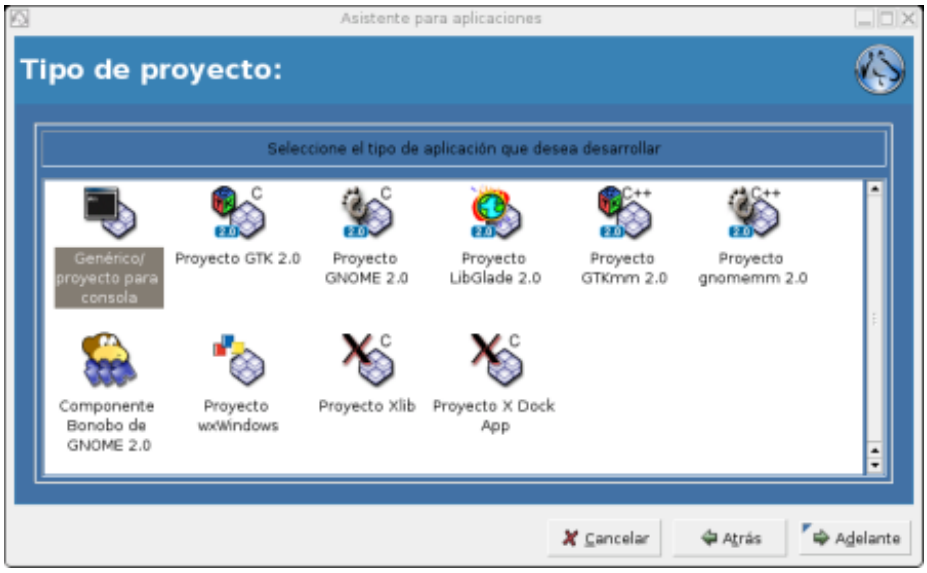
La instalación en Linux es demasiado fácil. Para poder compilar un juego realizado en SDL debemos de tener instalado gcc, que es el compilador que vamos a utilizar para poder compilar programas en C y luego instalar las librerias SDL que podemos bajarnos de su página Web, o instalar en el caso de debian desde "apt-get". Debemos de instalar los siguiente paquetes:

- libsdl1.2debian: paquete de librerías SDL
- libsdl1.2-dev: paquete de librerías SDL para desarrollo. Importante para poder compilar nuestros programas
- libsdl-image1.2: paquete para poder trabajar con diferentes tipos de imágenes
- libsdl-image1.2-dev: paquetes para desarrollo
- libsdl-mixer1.2: paquete para trabajar con diferentes formatos de sonido. SDL trae sus propias funciones para trabajar con sonidos, pero con libsdl-mixer podremos trabajar mejor
- libsdl-mixer1.2-dev: paquetes para desarrollo
- libsdl-ttf1.2: paquete para trabajar con fuentes ttf
- libsdl-ttf1.2-dev: paquetes para desarrollo

Por supuesto existen muchas más para muchas otras cosas, pero estas serán las librerías que vamos a utilizar durantes estos artículos.

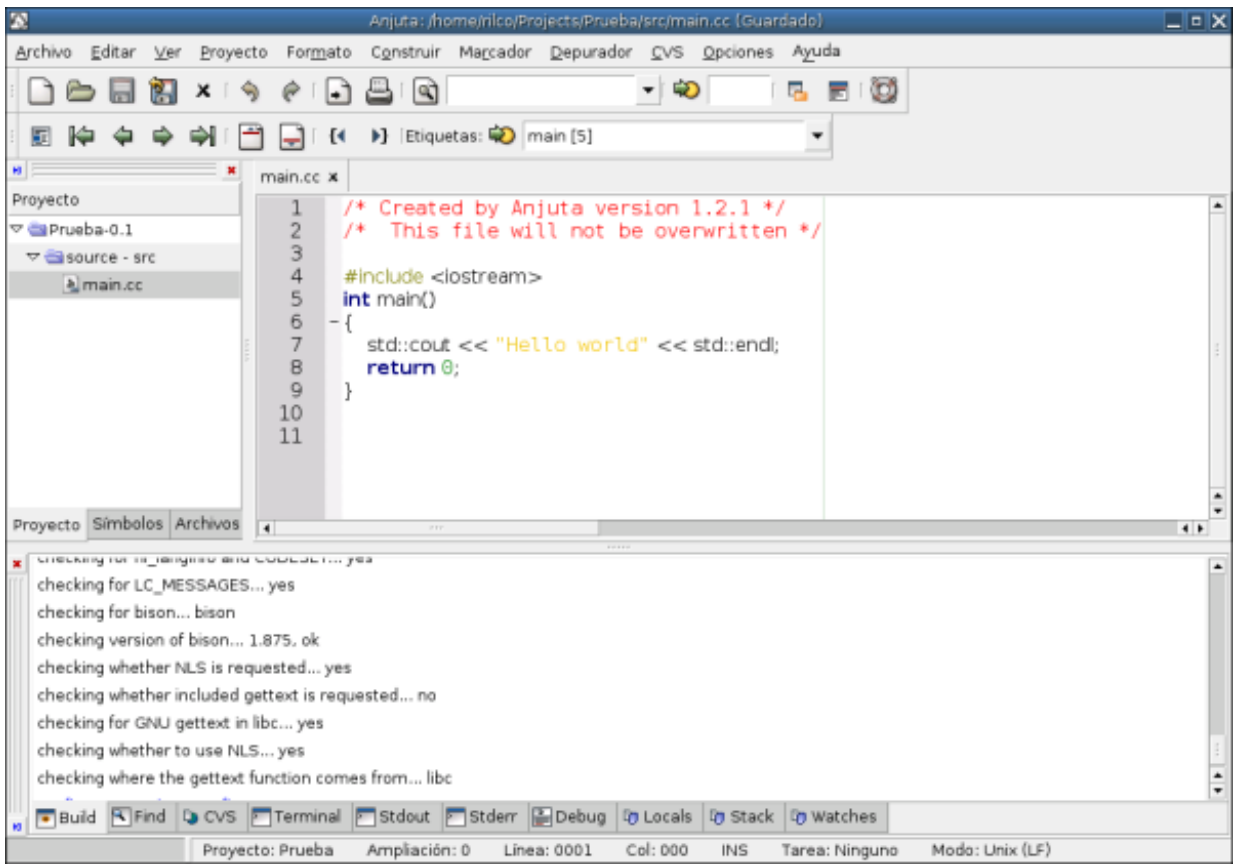
Después lo que tenemos es que configurar el programa con el que vayamos a compilar nuestros proyectos. Para ello utilizaremos Anjuta (http://www.anjuta.org/). Lo primero es crear un "Proyecto". Para ello pulsamos en "Archivo -> Nuevo Proyecto".

Seleccionamos un nuevo proyecto de consola.

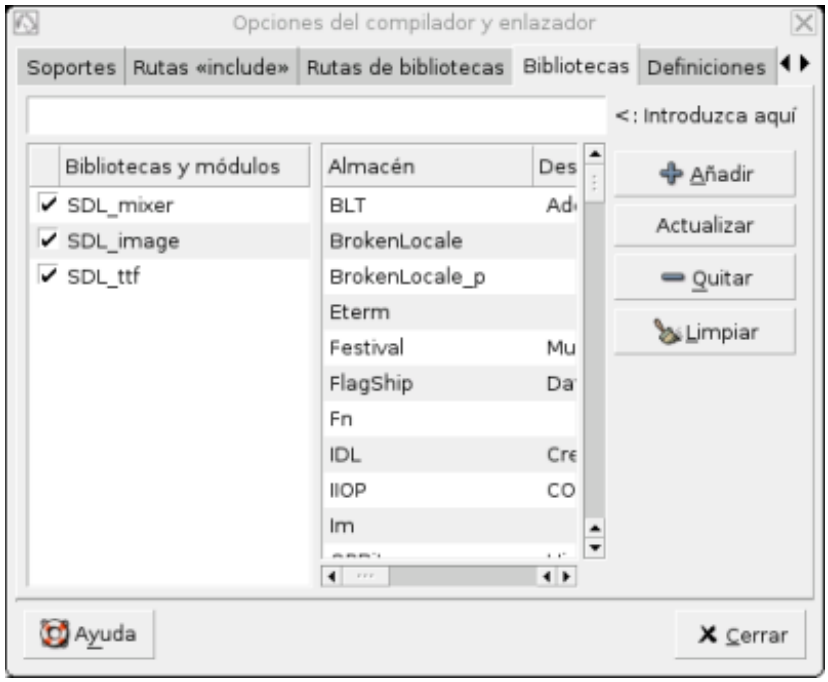


Luego colocamos el nombre al proyecto y seleccionamos "C y C++" como lenguajes. Recuerda que esto no lo podrás modificar luego.

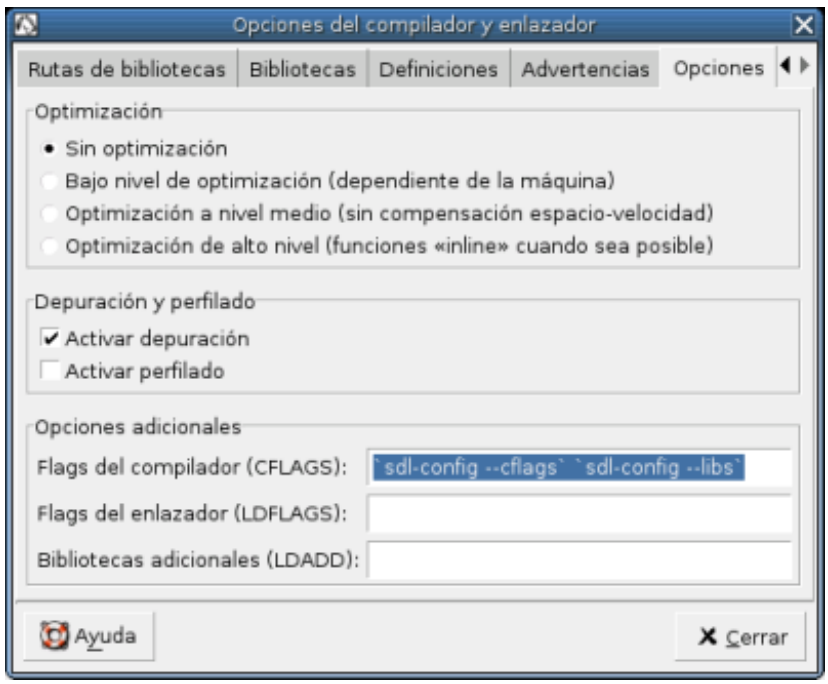
Colocamos la descripción de nuestro proyecto y terminamos. Anjuta nos debe de quedar algo tal que así.



Luego lo que tenemos que hacer el configurarlo para poder trabajar con SDL. Para ello nos vamos a "Opciones -> Opciones de compilador y enlazador". Allí nos vamos a la pestaña biblioteca y añadirmos las librerías SDL_mixer, SDL_image y SDL_ttf.



Por último nos vamos a la pestaña "Opciones" y colocamos los FLAGS. Nos posionamos en el cuadro Flags de compilador (CFLAGS): y podemos: "‘sdl-config --cflags‘ ‘sdl-config --libs‘".



Con esto ya tenemos Anjuta configurado para poder compilar. Para probarlo pondremos este código en el archivo "main.c" de nuestro proyecto y compilaremos el código pulsando "F11". Una vez compilado pulsaremos "F3" y si todo ha salido bien saldrá una pantalla en negro de la cual podremos salir pulsando una tecla. No te preocupes por no entender el código, dentro de poco lo tendrás todo claro.

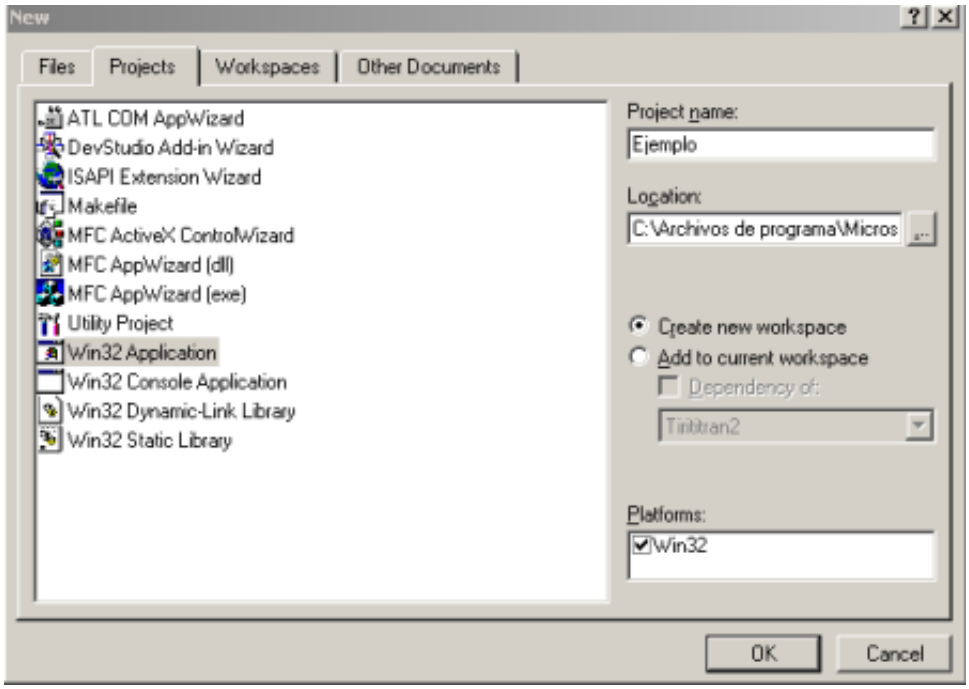
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL.h>
```

```
int main (int argc, char *argv[]) {
SDL_Event event;
SDL_Surface *screen;
int done = 0;

screen = SDL_SetVideoMode(640,480,16, SDL_SWSURFACE | SDL_DOUBLEBUF );
if(!screen){
printf("No se pudo iniciar la pantalla: %s\n", SDL_GetError());
SDL_Quit();
exit(-1);
}

while (done == 0) {
SDL_Flip (screen);
// Comprobando teclas para opciones
while (SDL_PollEvent(&event)) {
// Cerrar la ventana
if (event.type == SDL_QUIT) { done = 1; }
// Pulsando una tecla
if (event.type == SDL_KEYDOWN) {
done = 1;
}
}
}

SDL_FreeSurface(screen);

SDL_Quit();

    printf("\nTodo ha salido bien.\n");
    return 0;
}
```

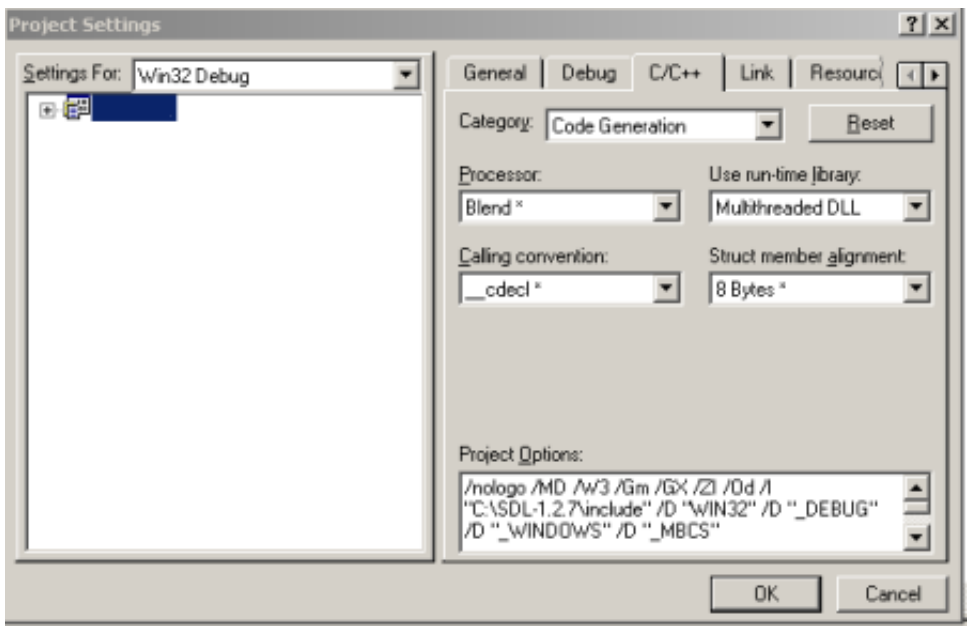
2.2. Instalación en Windows

La instalación en Windows tampoco es muy complicada. Explicaré como compilar en Microsoft Visual C++ 6, para otros programas debe de ser algo parecido. Primero debemos de bajarnos de la web de SDL el paquete "SDL-devel-1.2.7-VC6.zip". Estos están preparados para trabajar con Visual C++ 6. Una vez descargados los descomprimimos en C:\SDL-1.2.7. Esto dependerá de la versión, si hay una versión superior no dudes en barjarla.

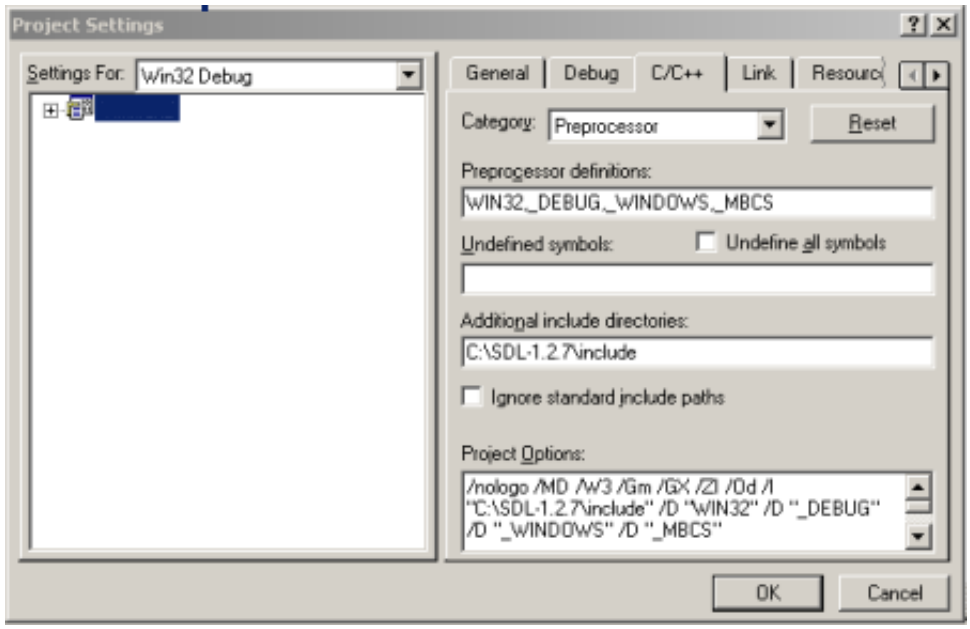
Ahora nos toca configurar Visual C++ 6 para poder trabajar con SDL. Para ello abrimos el programa y creamos un nuevo proyecto. Pulsamos en "Win32 Application" y le ponemos un nombre.



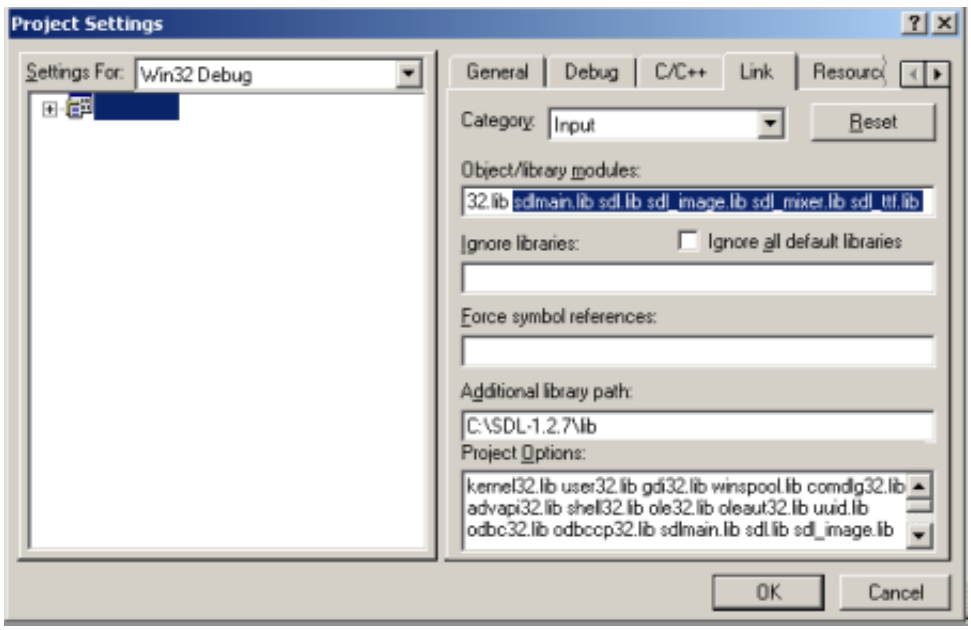
Una vez creado nos vamos a las opciones del proyecto en el menú. Nos vamos a la pestaña C/C++ y en el combo "Category" seleccionamos "Code Generation", luego en el combo "Use run-time librery" seleccionamos "Multithreaded DLL".



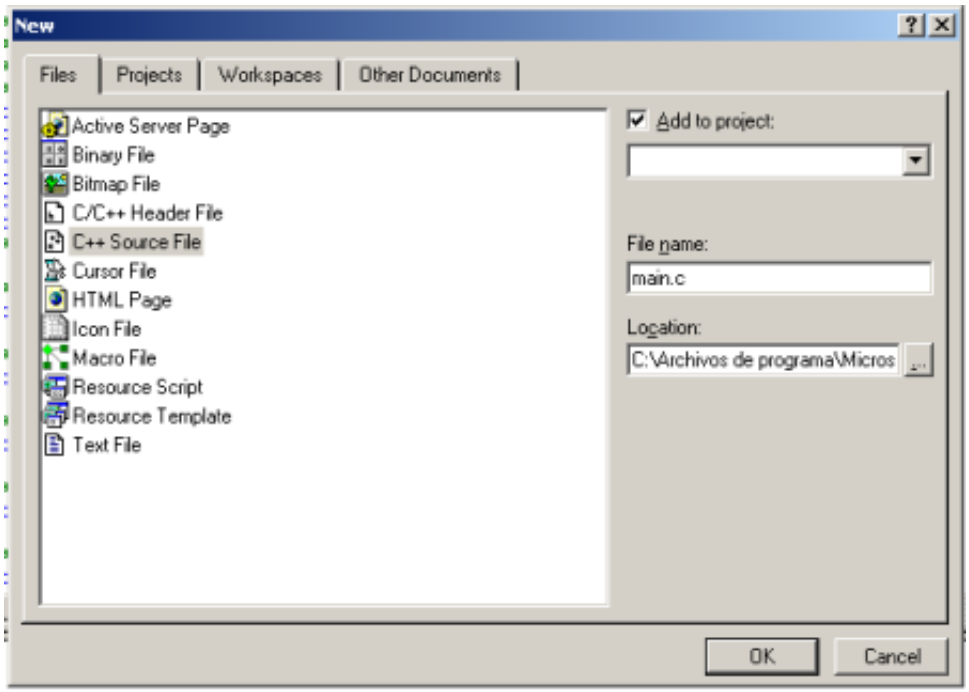
Luego seleccionamos en el combo "Category" "Preprocessor" y en la caja "Additional include directory" colocamos el directorio include donde hayamos incluidos las librerías.



Para finalizar las opciones le damos a la pestaña "Link" y en el combo "Input" dentro de la caja "Object librery modules", al final colocamos todas las librerías (archivos .lib) que vayamos a necesitar. Aquí incluimos tanto las SDL como las extensiones de SDL_Mixer, SDL_Image y SDL_TTF. Por último en la caja "Additional library path" colocamos el directorio de las librerías.



Una vez colocadas todas las opciones insertamos un nuevo archivo a nuestro proyecto seleccionando "C++ source file" y le colocamos el nombre.



Ya nada mas que tenemos que insertar el código del programa de prueba que está en la sección de "Instalación de Linux" para probar que todo funciona sin problemas.

Capítulo 3. Caráterísticas de un Vídeo Juego

Lo primero es saber como funciona un vídeo juego. Ahora mismo yo estoy trabajando en uno de ellos (mi primer vídeo juego) y después del desarrollo que llevo, una de las cosas que tengo claras es que crear un vídeo juego se traduce en comprobar. Durante todo el tiempo tenemos que comprobar cual es la posición de los enemigos, la del jugador, si explota o disparo un enemigo, si colisiona con otro, si ha muerto y cuantas vidas tiene, y todo lo necesario para que el juego se desarrolle con normalidad y el usuario no vea nada raro en él.

La creación de un juego no es simplemente programarlo, también tenemos que realizar buenos gráficos para él, y tener una buena música. Todo esto es necesario para que el juego tenga calidad y poder introducir al usuario dentro de nuestro mundo, un mundo que hemos creado y que queremos que el usuario no se salga de él mientras esté delante de la pantalla. Así que si no sabes dibujar o hacer una buena música, lo primero es buscar ayudantes que trabajen contigo durante el desarrollo. Por supuesto también tenemos que buscarle una buena historia, que enganche al usuario y un juego que no sea demasiado complicado ni tampoco demasiado fácil, tenemos que buscar un punto medio.

3.1. Programando un Vídeo Juego

Como he dicho antes un video juego, entre otras cosas se basa en comprobar, por lo que en cada instante del juego debemos de estar comprobando todos y cada uno de los movimientos de cada uno de los elementos que tenemos en la pantalla. Para ello debemos de tener un bucle que se está desarrollando en cada instante del juego y que hara las comprobaciones necesarias para que todo funcione correctamente.

Este bucle se realizará en el programa principal y llamará a todo lo que sea necesario para su trabajo. Puede ser parecido a esto:

```
done = 0;
while (done == 0) {
  COMPROBAR TECLAS;
  MOVER JUGADOR;
  MOVER ENEMIGOS;
  DISPAROS ENEMIGOS;
  DISPAROS JUGADOR;
  DIBUJAR EXPLOSIONES;
  COMPROBAR COLISIONES;
  DIBUJAR PANTALLA;
  SI ( SALIR DE JUEGO) {
    done = 1;
  }
}
```

Esto sería sólo el principio, luego tenemos que implementar cada una de las partes, pero esto nos puede hacer una idea de como trabaja un juego a grandes rasgos.

3.2. Utilizando Programación Orientada a Objetos

No es un objetivo de este artículo hablar sobre la POO, de esto se puede encontrar mucha información en internet y muy buena. Sólo voy a explicar un poco como lo vamos a utilizar en nuestros ejemplos.

Utilizaremos la POO para poder trabajar con objetos. Cada elemento que exista dentro de nuestros ejemplos será un objeto de una clase en concreto que definiremos en próximos capítulos. Así cada uno tendrá sus propios métodos y propiedades que le darán los comportamientos necesarios.

Después de conocer como trabaja SDL mi intención en trabajar en un juego, así que el que no tenga conocimientos de POO y quiera seguir con los siguientes artículos mejor que empiece a estudiar un poco esto porque le será necesario.

Capítulo 4. Conceptos básicos de SDL

A continuación voy a explicar los conceptos más básicos de SDL. Estos son desde como arrancar SDL en nuestro programas hasta el trabajo con superficies en pantalla. Con ello podremos hacer poca cosa, pero por algo tenemos que empezar.

4.1. Sistemas de SDL

SDL trabaja con varios subsistemas que podremos ir cargando en nuestro programa como veremos mas adelante. Cada uno de estos sistemas de encargan de algo en concreto, como del audio, vídeo, joystick, etc... A continuación vemos cada uno de ellos:

- SDL_INIT_VIDEO: inicializa el subsistema de vídeo
- SDL_INIT_AUDIO: inicializa el subsistema de audio
- SDL_INIT_CDROM: inicializa el subsistema para trabajo con el CDROM
- SDL_INIT_JOYSTICK: inicializa el subsistema de joystick
- SDL_INIT_EVERYTHING: inicializa todos los subsistemas

Cada uno de estos subsistemas contienen sus propias funciones para poder trabajar con la parte que tiene encargada dentro del programa. Por supuesto para poder trabajar con estas funciones tenemos que cargar el subsistema como veremos posteriormente.

4.2. Arrancando y cerrando SDL

Lo primero que tenemos que hacer cuando empecemos a programar un juego con estas librerías es incluirlas dentro de nuestro programa. Para ello ponemos:

```
#include <SDL.h>
```

Una vez tengamos incluidas las librerías tenemos que iniciar SDL. Para ello ponemos lo siguientes:

```
  if (SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO) == -1) {
    printf("No se pudo iniciar SDL: %s\n", SDL_GetError());
    SDL_Quit();
    exit(-1);
  }
```

Como podemos ver en el ejemplo, SDL_init nos inicializa SDL y en el caso que exista un error nos devuelve -1. También podemos ver la función SDL_GetError(), que contiene un cadena con el último error que se ha producido, que en el caso que no se hubiese podido cargar, lo escribiría por consola. Por último decir que a la función SDL_init le podemos pasar los sistemas que queramos cargar separados por "|".

En el caso que mas tarde queramos iniciar un sistema que previamente no hayamos cargado podemos hacerlo con SDL_InitSubSystem() pasándole como parámetro el sistema que queremos cargar y si queremos cerrar un sistema cargado podemos hacerlo con SDL_QuitSubSystem().

Siempre que abrimos algo, lo tenemos que cerrar y para ello tenemos esta función:

```
SDL_Quit();
```

que nos cierra SDL. Siempre lo colocaremos al final del programa, o en el caso de tener un error como en el caso anterior.

4.3. Trabajo con superficies

Una superficie contiene una imagen o una imagen formada por varias superficies. Para ello SDL tiene un tipo de dato llamado SDL_Surface, en el cual podemos cargar cualquier imagen y a la cual le podemos ir "pegando" otras imagenes cargadas en otros SDL_Surface. La definición de SDL_Surface es la siguiente:

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

En el caso de los video juegos es importante que no se puedan dibujar las superficies directamente en el pantalla. Si fuese así el usuario vería como se van colocando cada una de las imágenes en la pantalla en cada instante. Para solucionar esto SDL sólo nos deja mostrar en pantalla una sólo superficie, a la que con anterioridad le hemos podido "pegar" otras superficies.

Para no liarnos vamos a empezar a ver código. Lo primero será cargar la variable SDL_Surface que nos va a servir como pantalla y la cual vamos a ir colocando cada una de las imágenes de nuestro juego. Para ello tenemos la función SDL_SetVideoMode a la cual le podemos pasar varios parámetros.

```
SDL_Surface *screen;
screen = SDL_SetVideoMode(640,480,16, SDL_SWSURFACE | SDL_DOUBLEBUF | SDL_FULLSCREEN);
if(!screen){
  printf("No se pudo iniciar la pantalla: %s\n", SDL_GetError());
  SDL_Quit();
  exit(-1);
}
```

La variable screen contiene la única superficie que puede mostrarnos el juego. Los dos primeros parámetros son las resolución de pantalla, en este caso 640x480. El segundo la profundidad de color y como último parámetro le podemos pasar diferentes flags separados por "|". Los posibles flags son los siguientes:

- SDL_SWSURFACE: crea la superficie en memoria del sistema
- SDL_HWSURFACE: crea la superficie en memoria de vídeo
- SDL_DOUBLEBUF: activa el doble buffer. Lo que hace es que no se le muestra la pantalla directamente al usuario. Mientras el usuario ve la pantalla, el programa está trabajando en la siguiente pantalla a mostrar.
- SDL_FULLSCREEN: abre el programa en pantalla completa

Por este momento lo único que tenemos es la superficie que vamos a mostrar, pero por el momento es una pantalla en negro. Lo que haremos ahora es cargar una imagen en otra superficie que "pegaremos" en nuestra superficie principal (llamada "screen") la cual mostraremos. Para ello utilizaremos la siguiente imagen:

Lo primero es lo primero, y lo que tenemos que hacer es cargar la imagen. SDL por defecto sólo nos deja trabajar con archivos BMP utilizando la función SDL_LoadBMP(const char *file) que se le pasando como parámetro una cadena con la ruta del archivo BMP que queremos cargar. Después de probar diferentes formatos de imágenes yo me quedo con PNG, y no sólo porque sea un formato libre, sino porque es el mejor de todos y trabaja las transparencias como ningún otro. Pero el problema es que SDL no nos deja cargar PNG en nuestro programa. Para ello utilizamos SDL_image, que si habéis seguido el capítulo "Instalación de SDL" ya sabéis de lo que hablo. Lo primero es incluirlas en nuestro programa:

```
#include <SDL_image.h>
```

Luego en el programa cargamos la imagen:

```
  SDL_Surface *imagen;
  imagen = IMG_Load ("curso.png");
```

Ya tenemos cargada nuestra imagen, ahora toca colocarla dentro dentro de la superfecie "screen" para que puede ser visualizada. Para ello tenemos que definir un área rectangular que será donde será pegada nuestra imagen en el "screen". Para ello tenemos la estrucutura SDL_Rect que tiene la siguiente definición:

```
  typedef struct{
    Sint16 x, y;
    Uint16 w, h;
  } SDL_Rect;
```

La variable "x" nos da la posición X en el "screen", la "y" la posición Y, la "w" el ancho de la imagen y la "h" el alto.

Una vez creado nuestro SDL_Rect, utilizaremos la función SDL_BlitSurface() para colocar la imagen en el "screen". Esta función contiene 4 parámetro:

```
int SDL_BlitSurface(SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface *dst, SDL_Rect *dstrect);
```

La variable "src" es el "screen", "srcrect" normalmente tiene el valor NULL, "dst" es la superficie imagen y "dstrect" es el SDL_Rect que nos dará las posiciones donde queremos poner nuestra imagen en pantalla. El código completo quedaría algo tal que así:

```
  SDL_Surface *screen;
  SDL_Surface *imagen;
  SDL_Rect rect;
  screen = SDL_SetVideoMode(640,480,16, SDL_SWSURFACE | SDL_DOUBLEBUF | SDL_FULLSCREEN);
  if(!screen){
    printf("No se pudo iniciar la pantalla: %s\n", SDL_GetError());
    SDL_Quit();
    exit(-1);
```

```
}
imagen = IMG_Load ("curso.png");
rect.x = 100;
rect.y = 100;
rect.w = imagen->w;
rect.h = imagen->h;
SDL_BlitSurface(imagen, NULL, screen, &rect);
SDL_Flip (screen);
```

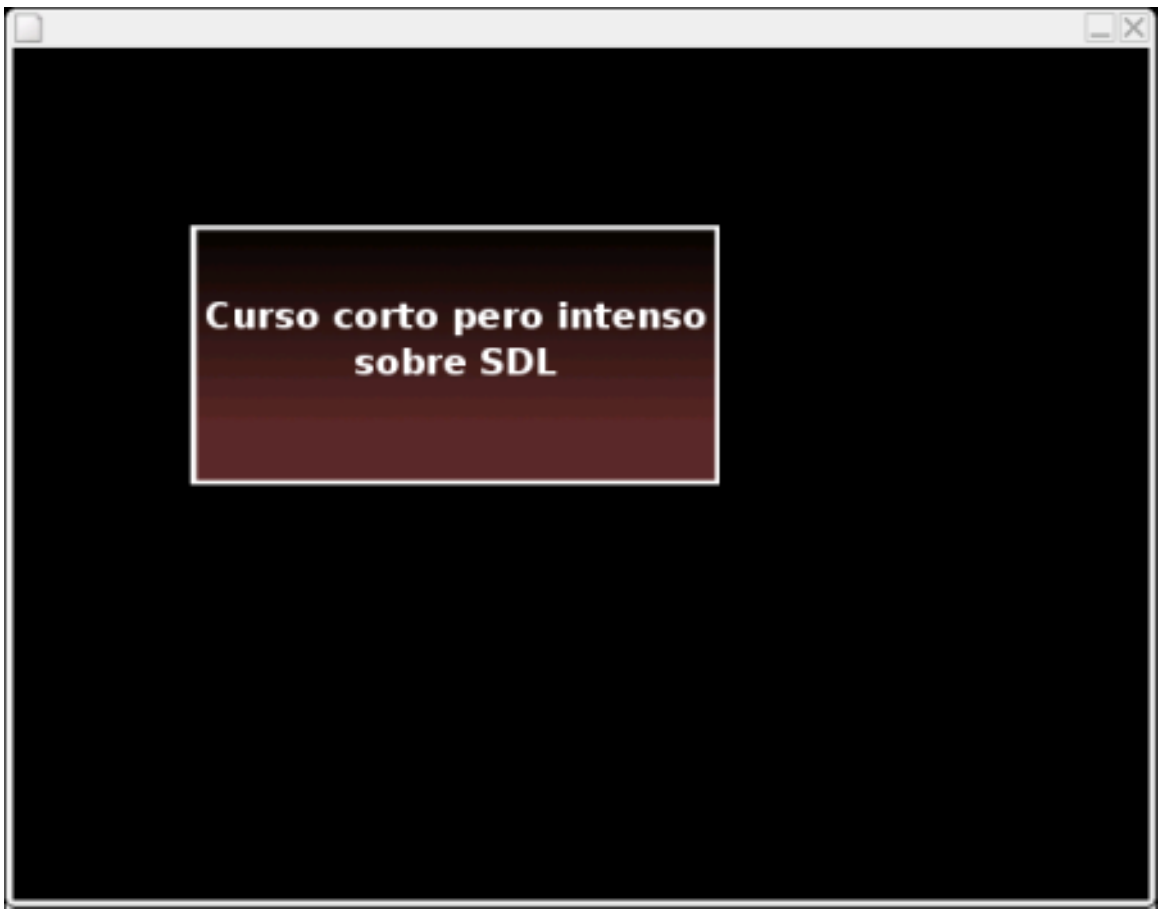
En este código podemos ver varias cosas. La primera como hemos cargado el archivo curso.png. Luego vemos como le damos los valores a la variable "rect". La posición x e y muestran que la imagen se cargar en la coordenada (100,100) de nuestra pantalla y a los valores w y h le damos la altura y anchura de la imagen PNG que hemos cargado. También vemos como colocamos la imagen en la pantalla con la función SDL_BlitSurface() y como refrescamos la pantalla con SDL_Flip (screen). La función SDL_Flip se tiene que ir repitiendo en cada bucle para que la pantalla muestre los nuevos datos.

Por último vamos a ver la función SDL_FreeSurface() a la cual le pasamos un SDL_Surface que como todos suponéis ya va a liberar de memoria.

Capítulo 5. Ejemplo completo

A continuación se muestra un ejemplo completo con todo lo que hemos aprendido en este artículo. El ejemplo carga una imagen en la pantalla hasta que el usuario pulsa una tecla para salir. Todavía no hemos visto nada sobre los eventos y las funciones para trabajar con teclado, que veremos en próximos artículos (o eso espero), así que lo que no entendáis no le deis importancia. Podemos ver en el ejemplo que el bucle de nuestro juego, como vimos en el capítulo "Programando un Vídeo Juego", es el "while" que se estará repitiendo hasta que se pulse una tecla. Por ahora es un bucle sencillo, pero es aquí donde tendremos que ir comprobando todos lo que hará nuestro juego: control de colsiones, disparos de enemigos, movimientos del jugador, etc...

En el caso que estés utilizando Anjuta en Linux, vas a tener que ejecutar el programa desde consola para que puede coger el archivo PNG. Ya veremos como solucionar esto en el futuro. El ejecutable lo tienes en el directorio "src/" del proyecto creado por Anjuta. Nuestro programa será algo tal que así:




```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <SDL.h>
#include <SDL_ttf.h>
#include <SDL_mixer.h>
#include <SDL_image.h>

int main (int argc, char *argv[]) {
  SDL_Event event;
  SDL_Surface *screen;
  SDL_Surface *imagen;
```

```
  SDL_Rect rect;
  int done = 0;

  screen = SDL_SetVideoMode(640,480,16, SDL_SWSURFACE | SDL_DOUBLEBUF );
  if(!screen){
    printf("No se pudo iniciar la pantalla: %s\n", SDL_GetError());
    SDL_Quit();
    exit(-1);
  }

  imagen = IMG_Load ("curso.png");
  rect.x = 100;
  rect.y = 100;
  rect.w = imagen->w;
  rect.h = imagen->h;
  SDL_BlitSurface(imagen, NULL, screen, &rect);

  while (done == 0) {
    SDL_Flip (screen);
    // Comprobando teclas para opciones
    while (SDL_PollEvent(&event)) {
      // Cerrar la ventana
      if (event.type == SDL_QUIT) { done = 1; }
  // Pulsando una tecla
      if (event.type == SDL_KEYDOWN) {
        done = 1;
      }
    }
  }

  SDL_FreeSurface(imagen);
  SDL_FreeSurface(screen);

  SDL_Quit();

  printf("\nTodo ha salido bien.\n");
  return 0;
}
```