SNX

Introducción básica a la programación en

C/C++
Con
SDL

El presente documento pretende servir de iniciación básica a las tecnologías anteriormente mencionadas. En la red existen muchísimos documentos avanzados que tratan el tema mas a fondo y con mayor rigurosidad, no soy ni me auto proclamo “gurú” de estas tecnologías, mi conocimiento es bastante básico pero lo he deseado compartir con quienes se interesen, mi correo electrónico esta disponible para recibir sugerencias, reclamos, etc. De la misma manera no me ago responsable de posibles problemas que este documento pueda provocar, ni tampoco de su mala utilización.

Osman Romero Alanis
Hoofmen@hotmail.com

25 may. 2005 Antofagasta – Chile

1. Mi historia en los juegos

Durante el tiempo que llevo como programador, siempre me llamo la atención el como era que se podían construir los grandes juegos que hoy vemos a diario, juegos como Quake o Unreal Tournament que en aquel momento era lo máximo en tecnología de motores tridimensionales, estos eran la meta a alcanzar. Pero antes de lanzarme en la programación 3D, sabia que era necesario conocer como funcionaba el desarrollo de juegos como el viejo Mario Bross, y toda la saga de juegos de Nintendo, pero también sabía que tenia que ir a un nivel más básico, como los juegos del viejo **Atari**.

Fue así como encontré en la red unos turoriales de **López** que estaban escritos en archivos de texto plano, venían con sources y recursos dentro de un archivo Zip, eran tutoriales para el desarrollo de juegos en **Turbo Pascal 7**, con los cuales aprendí mucho acerca de la programación grafica básica-forzada y los juegos 2D, digo grafica-forzada ya que turbo pascal 7 no presenta las suficientes herramientas para el buen desarrollo de aplicaciones graficas.

Cuando supe que los juegos grandes se programaban en C/C++, me vi obligado a aprender este lenguaje complicado pero potente y encontré los tutoriales de **Daniel Acuña**, y su comunidad **www.artebinario.cjb.net** que programaban juegos a un nivel superior, utilizando la librería **Allegro** y aplicando ingeniería de software en el desarrollo de sus proyectos. Allegro presentaba una API bastante buena para la creación de juegos.

Después de bastante tiempo programando en C y Allegro, tuve la suerte de conocer a 2 programadores aficionados de juegos, **Herman Polloni** y **Vicente Conejeros**, quienes me hablaron de **SDL** y me incluyeron en su proyecto del desarrollo de **SNX**, un motor 3D escrito en C, SDL, **OpenGL** entonces tuve que empezar a aprender OpenGL, pero debido al avanzado conocimiento de estas persona sobre la materia, yo en vez de aportar, estaba atrasando el proyecto y fue cuando me encomendaron la tarea de aprender **OpenAL**, librería que nadie había usado anteriormente por que me hizo ganar un lugar firme dentro del proyecto.

Mi motivación para escribir este documento es el saber que deben haber muchas personas de habla hispana que deseen recopilar información para el desarrollo de video juegos, y este documento pretende ser una guía de iniciación bastante básica sin términos tan técnicos pero con códigos fuentes completos que sean 100% compilables y sin sorpresas como suele suceder con los muchos códigos fuentes escritos en otros compiladores de C/C++.

Mis agradecimientos van a las personas anteriormente mencionadas que sin interés lucrativo han compartido sus conocimientos con los demás, al grupo **Xsoft**, a los lectores de este documento, a mi familia y amigos especialmente a mi novia Gisela.

2. La librería

- ✓ **SDL** creado por **Sam Lantinga** es una de las librerías más utilizadas para el desarrollo de software multimedia, especialmente video juegos. Además esta librería esta bajo la licencia LGPL lo más importante aún, es multiplataforma.

2.1 Herramientas y clasificación

Utilizaremos la IDE **Dev-C++** versión 4.9.80 con compilador incluido.

C/C++ este será el lenguaje a utilizar para la **lógica** de programación, que se asume como sabido.

SDL Librería para permitir crear las **Ventanas**, obtener información del hardware, cargar imágenes y permitir la interacción del usuario con el **Mouse** y **Teclado**.

2.2 Instalando las herramientas

El entorno de Dev-C++ se puede descargar gratuitamente desde **www.bloodshed.net**, que trae las herramientas necesarias para programar en C/C++, con un editor y además trae incluidas las librerías de OpenGL.

Antes de instalar las siguientes librerías será necesario entender el siguiente esquema

Fig. 1
Tipos de archivos necesarios en una librería.

A: Es el archivo **SDL.h** conocido como archivo cabecera para programar SDL en C/C++, desde el cual se pueden ver los métodos a utilizar (API) y es necesaria para compilar los programas.
Ubicación dentro del SO = **"lugar donde se instalo Dev-C++"\include\SDL**

B: Es el archivo **libSDL.a** o **libSDL.lib** que es necesario para poder compilar la aplicación SDL y crear un ejecutable.
Ubicación dentro del SO = **"lugar donde instalo Dev-C++"\lib**

C: Es el archivo **SDL.dll** que es necesario para poder ejecutar las aplicaciones creadas con SDL
Ubicación dentro del SO = **C:\WINDOWS\system32**

Este es solo un esquema y no representa por completo la librería Standard SDL, que contiene aproximadamente 29 archivos cabecera (.h), 4 archivos librería (.lib o .a) y 2 archivos de sistema (.dll) todos descargables desde **www.libsdl.org**

La manera **no recomendada** de instalar SDL para su uso desde Dev-C++ es de crear la carpeta **SDL** dentro de la carpeta **include** y colocar todos los archivos **.h** de SDL allí y luego copiar todos los archivos **.a** o **.lib** de SDL en la carpeta **lib**, sin olvidarse de los archivos DLL necesarios para SDL (**SDL.dll** y **SDL_image.dll**) que van en Windows\System32.

La manera **recomendada** es de descargar el **Dev-C++ package file** de SDL, que es un archivo de extensión **.DevPak** (SDL-dev-1.2.4.DevPak). Este archivo se instala con un doble clic el que automáticamente se ejecutara el programa **PackMan.exe** que es el encargado de instalar paquetes de librerías o utilidades para Dev-C++.

En caso de que Windows no encuentre PackMan.exe se puede encontrar en la carpeta donde se instalo Dev-C++.

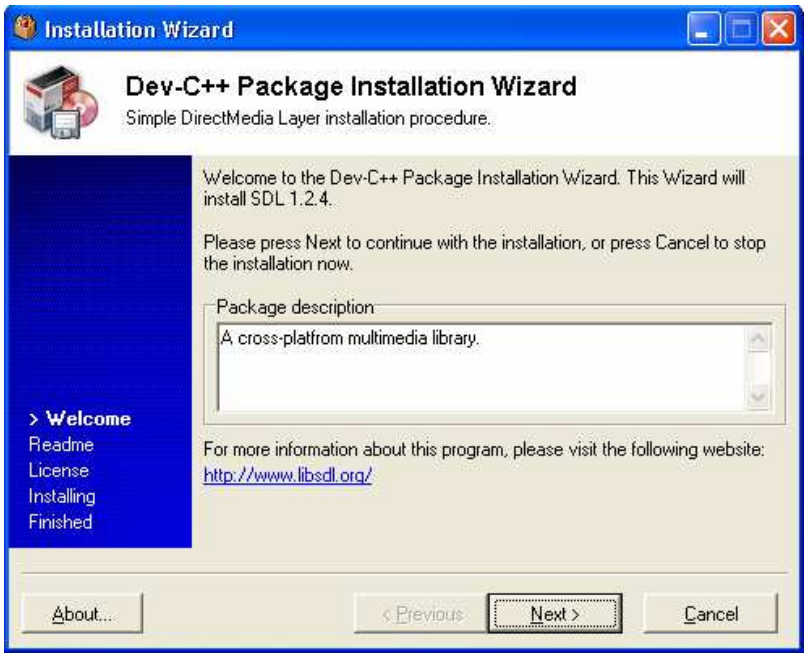

Fig. 2
Vista del programa PackMan.exe Instalando **SDL-dev-1.2.4.DevPak.**

En mi computador la instalación de las librerías luce así:

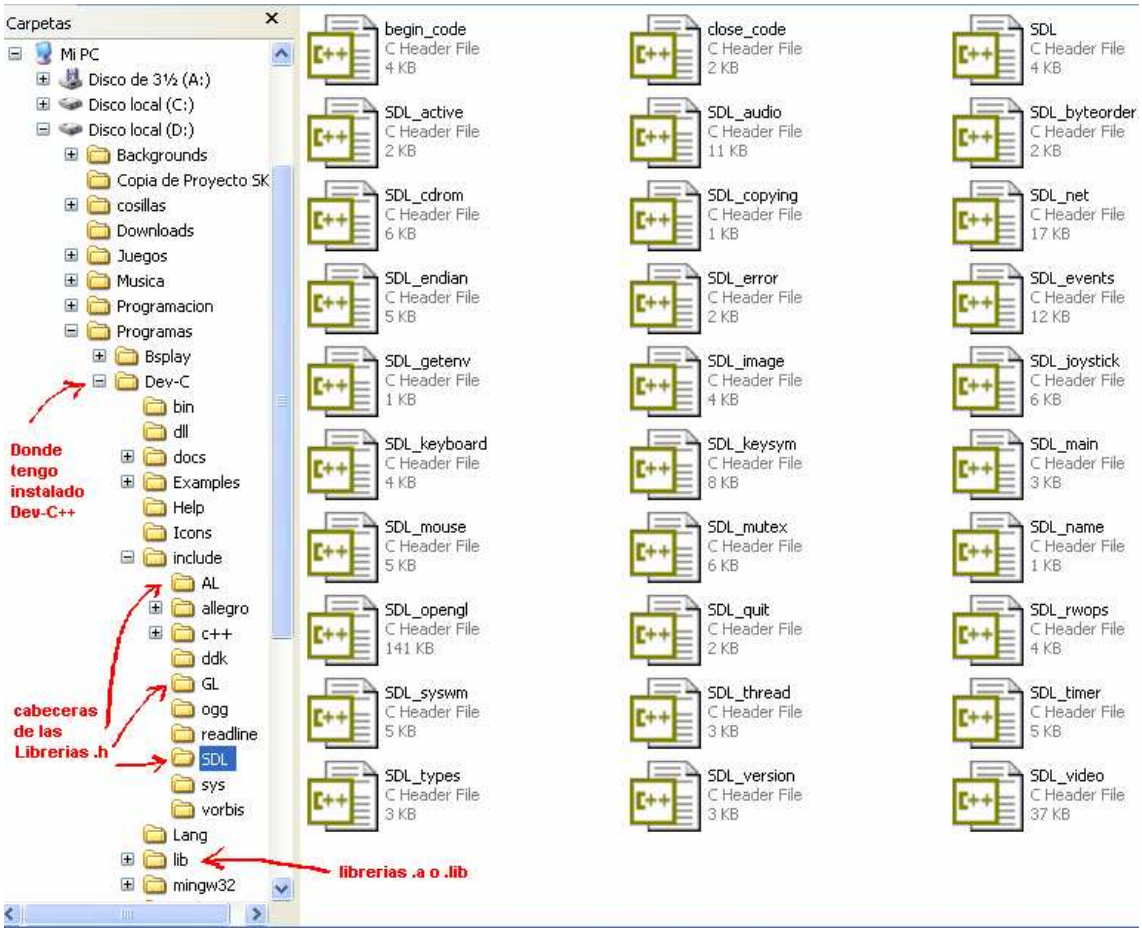

Fig.3 Vista de la jerarquía de directorios y archivos del programa Dev-C++.

2.3 Crear un proyecto SDL

Para crear un proyecto en Dev-C++ sigan las siguientes instrucciones visuales:

Fig.4 Crear un proyecto nuevo.

Seleccionar tipo de proyecto y guardarlo:

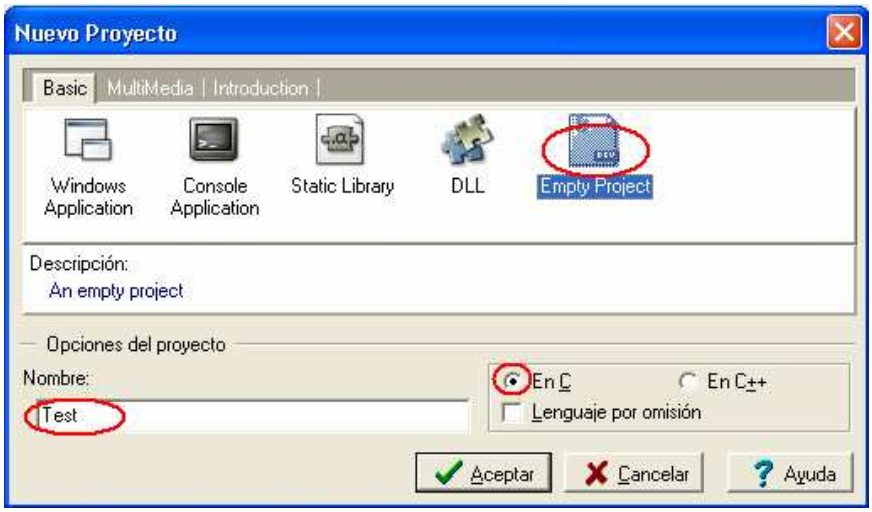

Fig.5 Selección de las características del proyecto.

Agregar un archivo fuente al proyecto:

Fig.6 Agregar un archivo de código fuente al proyecto.

2.4 Código Fuente “Mi primer programa SDL”

```
    /*includes standard para C*/
#include <stdio.h>
#include <stdlib.h>
    /*include para usar SDL*/
#include <SDL/SDL.h>

    /*programa principal*/
int main(int argc, char *argv[])
{
        /*declaro a screen como un puntero a superficie SDL*/
    SDL_Surface* screen;

        /*inicializo SDL, si este metodo me retorna un valor menor a 0 entonces Error*/
    if (SDL_Init(SDL_INIT_VIDEO)<0)
    {
            /*imprimo el error SDL_GetError() tambien lanza informacion de DEBUG*/
        printf("Error al iniciar SDL:: %s",SDL_GetError());
                /*
                aca no importa que valor retorne, lo que si este return terminaria
                con la ejecucion del programa
                */
        return -1;
    }

    /*
    inicializo el contexto SDL en una ventana REDIMENSIONABLE (resizable) con
    superficies generadas por hardware (hwsurface, o swsurface), de tamaño 640*480 y de
    16 bits en profundidad de color, ver SDL_video.h para mas modos
        este metodo me retorna una superficie de tipo SDL_Surface generada con las
        caracteristicas mencionadas
    y se abrira una ventana si no hay error
    */
    screen = SDL_SetVideoMode(640,480,16,SDL_HWSURFACE|SDL_RESIZABLE);

        /* si screen aun es null, entonces no se pudo inicializar el contexto SDL*/
    if (!screen)
    {
        printf("Error al iniciar el contexto SDL:: %s",SDL_GetError());
        return -1;
    }

        /*este metodo sirve para escribir un titulo en el caption de la ventana */
    SDL_WM_SetCaption(“:: Mi primera ventana SDL ::”,NULL);
        /* este metodo proboca un delay de 1000 nano-segs en la ejecucion del programa*/
    SDL_Delay(1000);

        /*return 1 diciendo que todo salio ok y se termina el programa*/
    return 1;
}
```

2.5 Agregar las librerías para compilar

Para poder hacer enlace a las librerías y archivos de cabecera de SDL se deben agregar unas líneas a las opciones del proyecto.

Fig.7
Acceder a las opciones del proyecto o Alt+P.

En Opciones del Proyecto se debe cambiar el **tipo** por Win32 GUI grafico y en **Argum…**

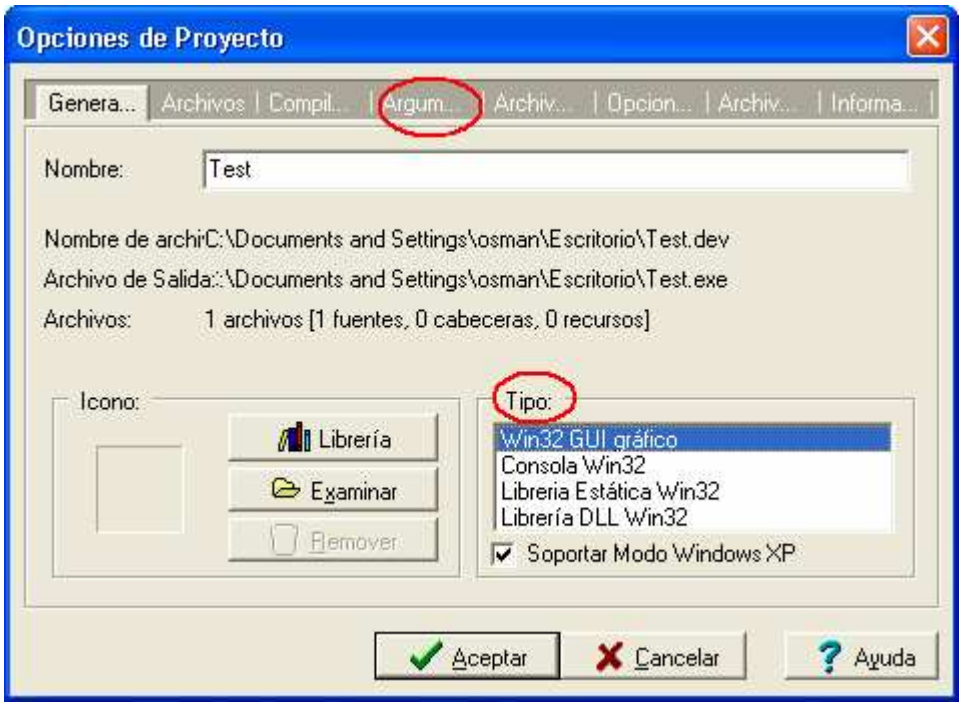

Fig.7.1 Opciones del proyecto.

Se debe especificar donde están las cabeceras de SDL en la sección **Compilador**:

```
-I"<INCLUDE>\SDL" -Dmain=SDL_main
```

Y tambien especificar las librerías en **Enlazador(linker)**:

```
-lmingw32 -lSDLmain -lSDL
```

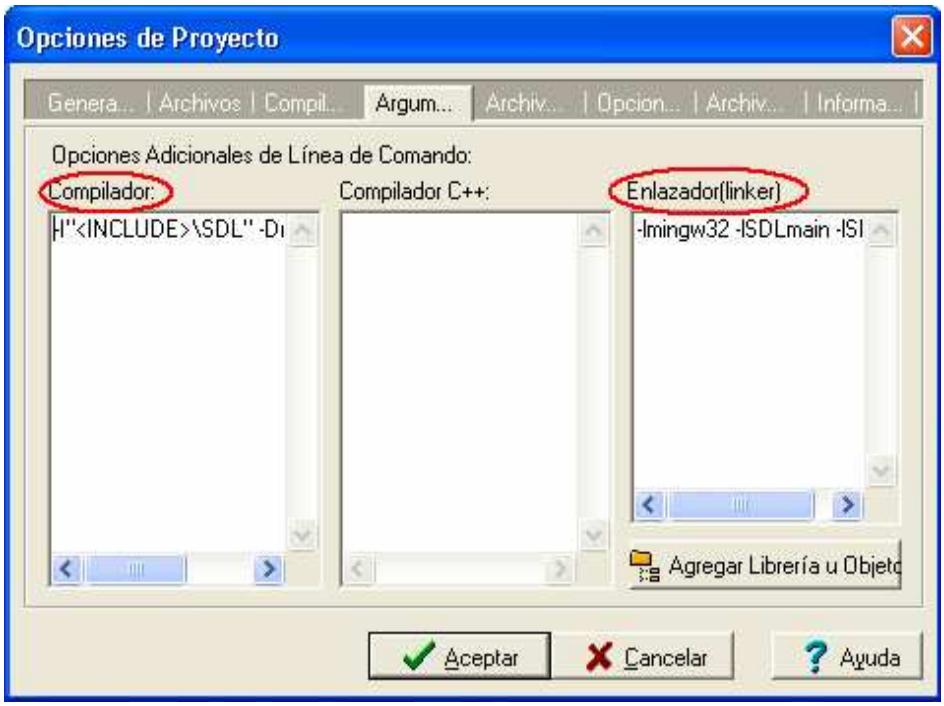

Fig.7.2 Opciones adicionales del proyecto.

2.6 Compilando el proyecto

Una vez configuradas las opciones del proyecto, se puede proceder a compilar y todo resulta bien habrá creado nuestra primera aplicación SDL.

Fig.8
Proceso de compilación del proyecto o Ctrl+F9.

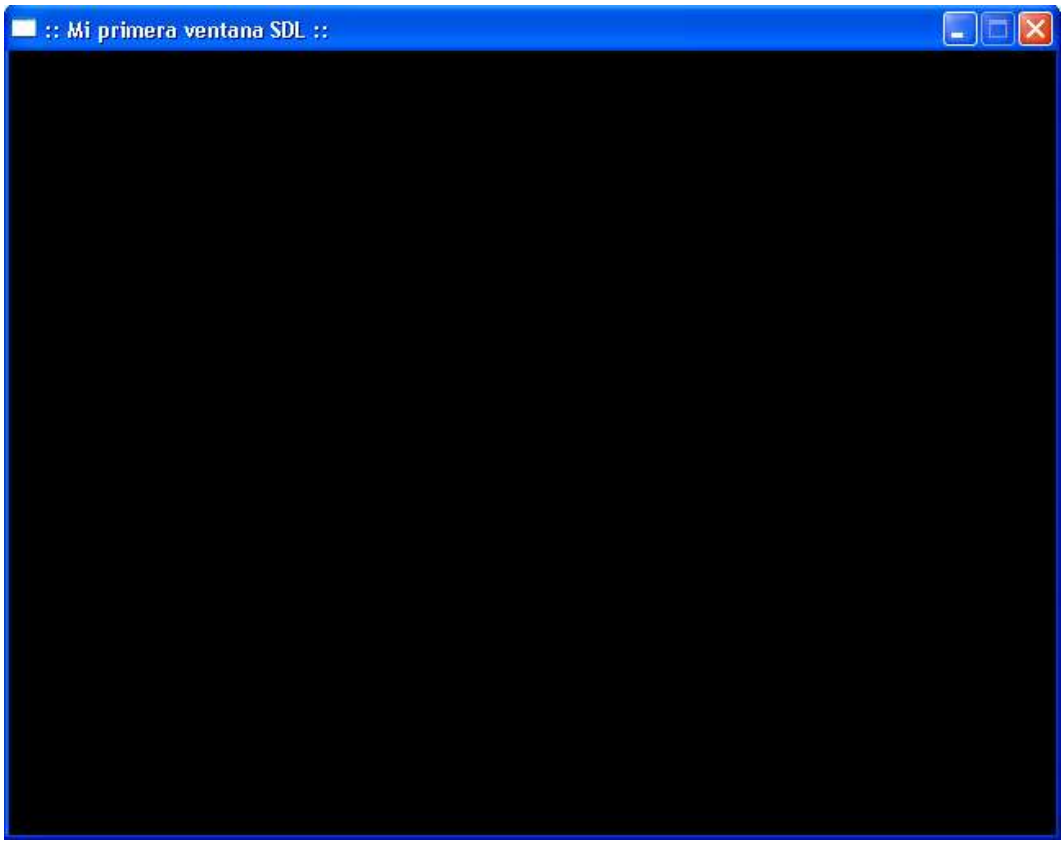

Fig.9 Ejecución de la ventana 640*480 en SDL.

3 Carga de imágenes en SDL

La carga de imágenes en SDL y su manipulación es bastante simple ya que cualquier imagen cargada con queda almacenada en memoria como una variable de tipo SDL_Surface*, existen distintos métodos para cargar formatos de imágenes específicos, pero sin duda el mas simple es IMG_Load(char* fileName), que se abstrae del formato de la imagen (JPG, GIF, BMP, etc.) y la carga retornándola como SDL_Surface.

3.1 Código Fuente “Carga de imagen con SDL”

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
    /* se incluye SDL_image.h para los metodos de
       carga de imagenes */
#include <SDL/SDL_image.h>

    /* metodo que permite cargar imagenes y las devuelve
       como SDL_Surface */
SDL_Surface* cargarImagen(char *nombreArchivo)
{
        /* SDL tiene metodos para cargar cada tipo de imagen
           BMP,PCX,JPG,GIF,etc. pero este metodo (IMG_Load())
           automaticamente carga imagenes sin preguntar el tipo*/
    SDL_Surface *image = IMG_Load(nombreArchivo);
    return image;
}

    /* metodo que permite dibujar una *imagen previamente cargada
       (con cargarImagen()), y dibujarla en una superficie dada
       (en este caso *screen), en la posicion (x,y)*/
void dibujaImagen(SDL_Surface *screen, SDL_Surface *imagen, int x, int y)
{
        /* se crea un rectangulo */
            SDL_Rect rect;
               /* se le asigna donde empieza en x el rectangulo */
            rect.x=x;
               /* se le asigna donde empieza en y el rectangulo */
            rect.y=y;
               /* se pega la imagen sobre screen de acuerdo a
                  la posicion del rectangulo */
            SDL_BlitSurface(imagen, 0, screen, &rect);
}

int main(int argc, char *argv[])
{
    SDL_Surface* screen;
    if (SDL_Init(SDL_INIT_VIDEO)<0)
    {
        printf("Error al iniciar SDL:: %s",SDL_GetError());
        return -1;
    }

    screen = SDL_SetVideoMode(640,480,16,SDL_HWSURFACE|SDL_RESIZABLE);

    if (!screen)
    {
        printf("Error al iniciar el contexto SDL:: %s",SDL_GetError());
        return -1;
    }

    SDL_WM_SetCaption(":: Carga de imagen con SDL ::",NULL);
        /* llamamos al metodo para que cargue la imagen "foto.jpg" */
    SDL_Surface* img = cargarImagen("foto.jpg");
        /* preguntamos si realmente se cargo la imagen */
    if (!img)
    {
            /* si nose cargo, salimos del programa */
        printf("No se pudo cargar la imagen: %s",SDL_GetError());
        return -1;
    }
        /* llamamos al otro metodo para que dibuje la imagen (*img)
           sobre nuestra superficie buffer (*screen), en la posicion
           100,100
        */
    dibujaImagen(screen, img, 100, 100);
        /* Este metodo tomara todos los cambios que se han hecho sobre
           la superficie buffer(*screen), y las desplegara por pantalla
        */
    SDL_Flip(screen);
    SDL_Delay(5000);
    return 1;
}
```

3.2 Agregar las librerías para compilar

El ejemplo anterior utiliza para cargar las imágenes una cabecera extra, que es SDL_image.h es por esto que deberá agregar una librería mas en **Enlazador(linker)** como lo muestran las figuras anteriores **7**, **7.1** y **7.2**.

-lmingw32 -lSDLmain -lSDL –lSDL_image

Si se habrán dado cuenta, el metodo SDL_Delay(5000) consume mucho recurso ya que deja esperando en ejecución al programa y además es bastante malo por lo que les recomiendo que cambien ese metodo por todo este código:

```
    /*
     Se declara una variable de tipo evento
    (en este caso es para eventos de la ventana)
    */
SDL_Event event;
    /* todo esta ok?, asi que ok es true (1)*/
int ok = 1;
    /* mientras todo ok */
while (ok)
{
    /* verificar si hay eventos en la ventana
       (minizar, maximizar, cerrar) */
    if(SDL_PollEvent(&event))
    {
                /* preguntar si el tipo de evento es
                   QUIT (cerrar ventana)*/
            if (event.type==SDL_QUIT)
            {
                 /* todo deja de estar ok (0)*/
                 ok = 0;
            }
    }
}
```

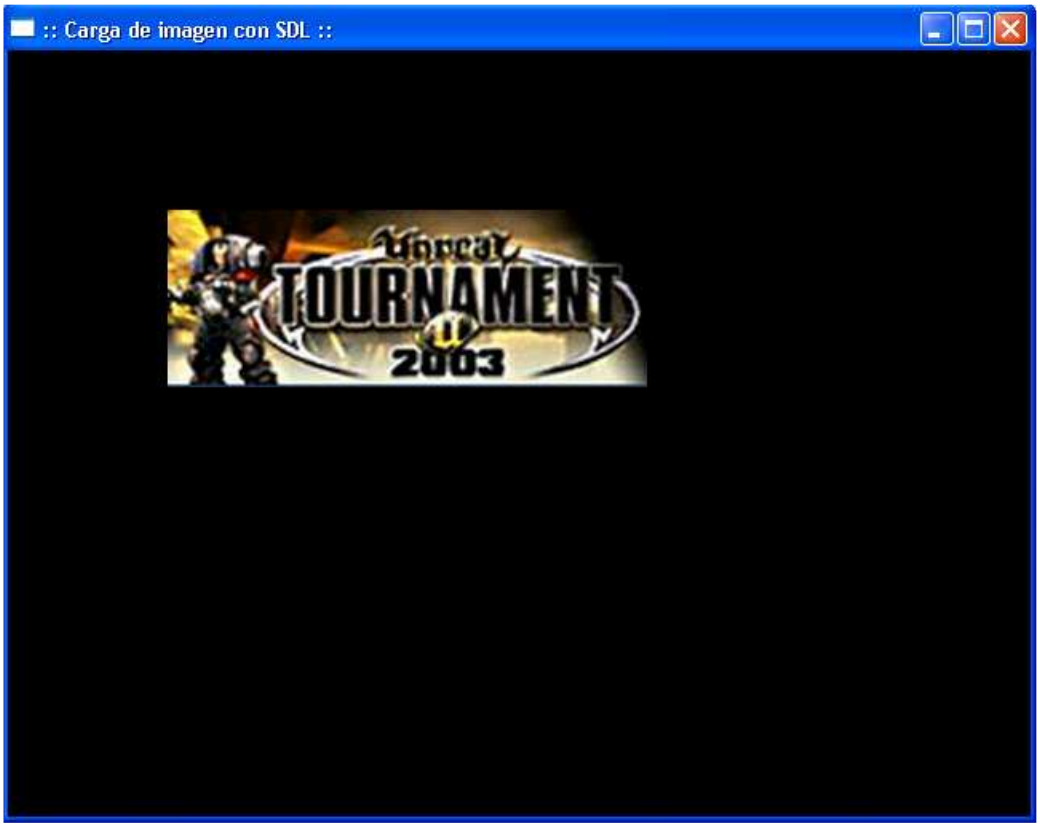

Fig.10 Ejecución del programa de carga de imagen con SDL.

3.3 Entendiendo SDL_Flip(SDL_Surface*)

La manera en que los videojuegos 2D hacen sus animaciones es casi de la misma manera en que una dibujante hace sus animaciones en papel, esto es, dibujando cada uno de los movimientos del personaje a animar y mostrándolos en el orden que corresponden. Entonces lo que SDL_Flip hace, es tomar la superficie que se le envía por parámetros y la copia a la pantalla, que es cuando nosotros la podemos ver.

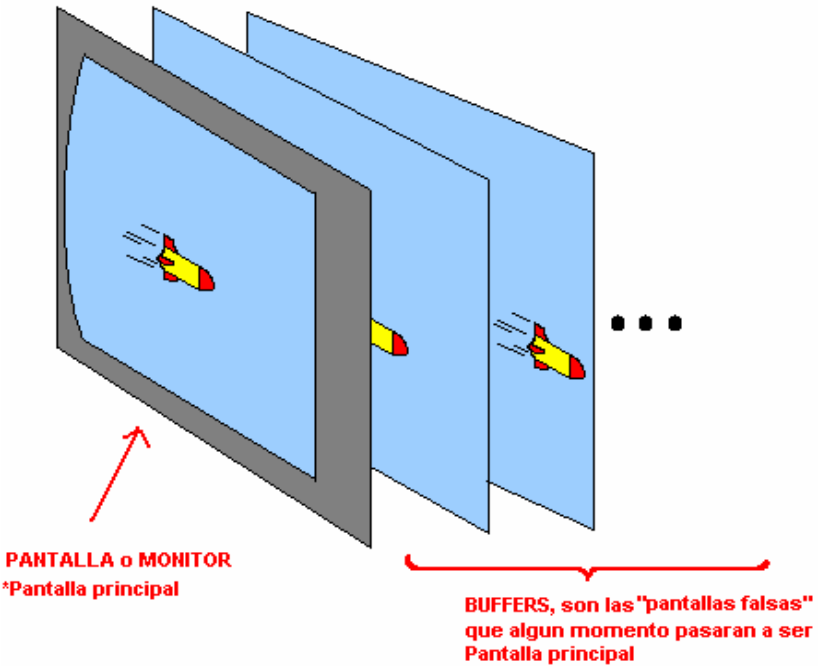

Fig.11 Forma en la que se anima un objeto en los juegos 2D. En la programación de juegos esos objetos se denominan "Sprites".

Ahora que se entiende la manera en la que se animan los objetos, se estarán preguntando como hacer para animar movimientos propios de los personajes, como en el caso de un juego de pelea, donde los personajes pueden lanzar patadas y puñetazos.

3.4 Animación de Sprites

Para hacer esto, la manera mas simple es que en vez de declarar la imagen como SDL_Surface*, se debe crear **un arreglo de tipo SDL_Surface*** es decir, SDL_Surface *kick[5], y en cada posición del arreglo cargar la parte correspondiente a la animación.

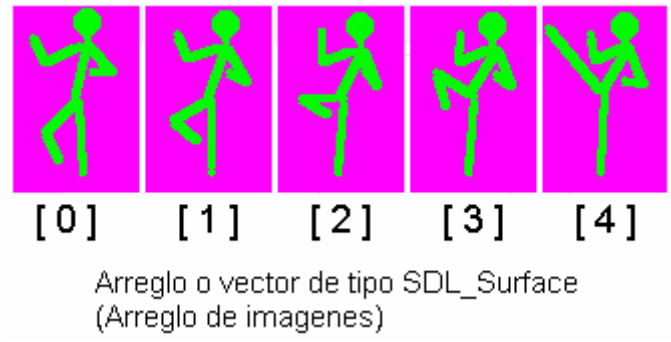

Fig.12 Animación de una patada.

Una vez cargado el arreglo con las imágenes, se puede completar, o ejecutar la animación con un simple **for** o **do-while** de 0 a 4 y para devolver la patada será de 4 a 0.

Trozo de código para una animación

```
   SDL_Surface *kick[5];
       //carga de las imagenes para la patada
   kick[0]= cargarImagen("images/kick0.jpg");
   kick[1]= cargarImagen("images/kick1.jpg");
   kick[2]= cargarImagen("images/kick2.jpg");
   kick[3]= cargarImagen("images/kick3.jpg");
   kick[4]= cargarImagen("images/kick4.jpg");
       //ciclo de animacion
   int i=0;
   int di=1;
   do
   {
        //se manda a dibujar el frame o cuadro de la patada
      dibujaImagen(screen,kick[i],100,100);
       //se muestra el dibujo
      SDL_Flip(screen);
       /** Este metodo se presentara mas adelante…
       se limpia el buffer donde se esta dibujando para volverlo a utilizar*/
      clear(screen);
       //se da un pequeño retardo para que no sea muy rapida la animacion
      SDL_Delay(70);
       //incremento i en uno o decremento i en uno segun el if de mas abajo
      i=i+di;
       //este if permite animar la patada en reversa
      if (i==4)
        di=di*(-1);
    }while(i!=-1);
```

3.5 Configuración de colores transparentes en SDL

Cuando cargamos una imagen que será un actor de nuestro juego, el fondo de la imagen del actor estorba cuando se le quiere dibujar sobre un escenario cualquiera, para esto es necesario definir colores de fondo que nosotros queramos que el programa los considere transparentes, un buen color transparente puede ser el rosado (con valor RGB 255,0,255) ya que no es muy común en las imágenes. Para definir el color clave rosado como transparente en SDL, se hace uso del método SDL_SetColorKey() que le dirá a SDL que no dibuje o no pinte los píxeles sean del color clave, en este caso de color rosado.

Entonces modificaremos el método que utilizábamos para cargar las imágenes, agregando la posibilidad de definir transparencia.

Trozo de código para definir color transparente

```
SDL_Surface* cargarImagen(char *nombreArchivo)
{
            /*declaro 2 imagenes, una que cargara la imagen con el fondo rosado y la
            otra sera la imagen resultante sin el fondo*/
       SDL_Surface *image, *bmp;

       image = IMG_Load(nombreArchivo);
       if(!image) return 0;
              /* declaro el color clave según la paleta de colores de la imagen cargada
              y definiendo al color rosado (255,0,255) como transparente */
       Uint32 color_key = SDL_MapRGB(image->format, 255, 0, 255);
       /* defino que cualquier pixel de color rosado en la imagen es transparente */
       SDL_SetColorKey(image, SDL_SRCCOLORKEY|SDL_RLEACCEL, color_key);
       /* obtengo el la imagen libre del color rosado de fondo */
       bmp=SDL_DisplayFormat(image);
              /* libero la memoria utilizada para la primera imagen */
       SDL_FreeSurface(image);
       if(!bmp) return 0;

       return bmp;
}
```

3.6 Código Fuente “Animación de Sprites en SDL”

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>

       /* el nuevo metodo para cargar imágenes */
SDL_Surface* cargarImagen(char *nombreArchivo)
{
    SDL_Surface *image, *bmp;

    image = IMG_Load(nombreArchivo);

    if(!image) return 0;

    Uint32 color_key =SDL_MapRGB(image->format, 255, 0, 255);
    SDL_SetColorKey(image, SDL_SRCCOLORKEY|SDL_RLEACCEL, color_key);
    bmp=SDL_DisplayFormat(image);
    SDL_FreeSurface(image);
    if(!bmp) return 0;

    return bmp;
}

void dibujaImagen(SDL_Surface *screen, SDL_Surface *imagen, int x, int y)
{
       SDL_Rect rect;
       rect.x=x;
       rect.y=y;
       SDL_BlitSurface(imagen, 0, screen, &rect);
}

void clear(SDL_Surface *screen)
{
  Uint32 col=SDL_MapRGB(screen->format, 0, 0, 0);
  SDL_FillRect(screen, 0, col);
}


int main(int argc, char *argv[])
{
    SDL_Surface* screen;
    if (SDL_Init(SDL_INIT_VIDEO)<0)
    {
        printf("Error al iniciar SDL:: %s",SDL_GetError());
        return -1;
    }

    screen = SDL_SetVideoMode(320,200,16,SDL_HWSURFACE|SDL_RESIZABLE);

    if (!screen)
    {
        printf("Error al iniciar el contexto SDL:: %s",SDL_GetError());
        return -1;
    }

    SDL_WM_SetCaption(":: Animación de Sprites en SDL ::",NULL);

    /* imagenes para la animacion */
    SDL_Surface *gun[3];
    gun[0] = cargarImagen("imagenes/RR.bmp");
    gun[1] = cargarImagen("imagenes/RR2.bmp");
    gun[2] = cargarImagen("imagenes/RR1.bmp");
        /* si no se cargo alguna imagen se sale del programa */
    if (!gun[0] || !gun[1] || !gun[2])
    {
        printf("No se pudo cargar la imagen: %s",SDL_GetError());
        return -1;
    }

    SDL_Event event;
    int ok = 1;
```

```
    while (ok)
    {
            /* codigo para animar el sprite */
        int i=0;
        int di=1;
        do
        {
           dibujaImagen(screen,gun[i],120,120);
           SDL_Flip(screen);
           clear(screen);
           SDL_Delay(70);
           i=i+di;
           if (i==2)
               di=di*(-1);
        }while(i!=-1);
                /* que espere hasta hacer el proximo disparo */
        SDL_Delay(500);
        if(SDL_PollEvent(&event))
        {
                if (event.type==SDL_QUIT)
                {
                     ok = 0;
                }
        }
    }
    return 1;
}
```

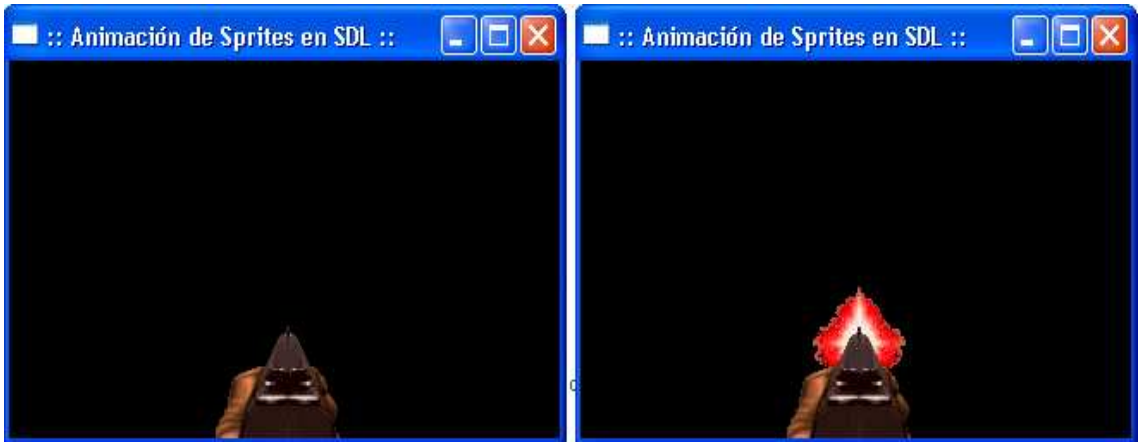

Fig.13 animación de sprites, junto con carga de fondo transparente.

El resultado de este código es un programa que carga 3 imágenes y las anima de la imagen 0 a la 3 y de la imagen 3 a la imagen 0 una y otra vez en una ventana de 320x200, en mi caso la animación es del arma de Doom II disparando, también las imágenes cargadas tienen fondo rosado, que fue eliminado por el algoritmo que anteriormente revisamos.

Fig.14 Muestra del proceso de animación del programa "Animación de Sprites en SDL"

4 Manejo de eventos con SDL

4.1 Teclado con SDL

Para el manejo de eventos de teclado. SDL nos entrega un arreglo de tipo **Uint8** con acceso a 322 constantes para cada tecla y su identificador. Todos los nombres que SDL le da a sus teclas se pueden ver en el archivo **SDL_keysym.h**

Código:

```
//declaracion de la variable key
Uint8 *key;
       //se obtiene el arreglo de teclas
key = SDL_GetKeyState(0);
       //algunas posibles teclas a presionar
if (key[SDLK_UP])
   printf("Se ah presionado la flecha ARRIBA");
if (key[SDLK_DOWN])
   printf("Se ah presionado la flecha ABAJO");
if (key[SDLK_LEFT])
   printf("Se ah presionado la flecha IZQUIERDA");
if (key[SDLK_RIGHT])
   printf("Se ah presionado la flecha DERECHA");
if (key[SDLK_ESCAPE])
   printf("Se ah presionado la tecla ESCAPE");
```

El manejo de eventos de teclado es muy sensitivo en SDL, es decir, si mantienes presionada una o mas teclas, SDL atenderá de manera concurrente la instrucción asociada a cada tecla, y dado que las teclas son tan sensibles en SDL el haber presionado una tecla una sola vez, puede ser considerado como que se mantuvo presionado por una cierta cantidad de microsegundos. Ahora bien todo esto se puede arreglar con un poco de código.

4.2 Mouse con SDL

Para poder utilizar el Mouse con SDL se utiliza la variable que anteriormente declaramos como **SDL_Event event**, donde se puede preguntar por el estado del mouse, si este se esta moviendo o si se ha hecho un clic y con que botón. Mas información sobre los métodos que tiene el mouse ver **SDL_mouse.h**

Código:

```
if(SDL_PollEvent(&event))
{
         /* evento en el que cierran la ventana */
     if (event.type==SDL_QUIT)
        ok = 0;
         /* evento en el que se mueve el mouse */
     if (event.type = SDL_MOUSEMOTION)
     {
         int x,y;
         /* obtengo la posicion del mouse */
        SDL_GetMouseState(&x,&y);
        printf("El mouse esta en la posicion %i,%i",x,y);
     }

     if (event.type == SDL_MOUSEBUTTONDOWN)
     {
         /* vuelvo a mostrar el cursor del mouse, activo el teclado
            y desactivo el mouse */
         printf("Se ha hecho clic con el mouse");
     }
}
```

4.3 Código Fuente “Mouse y Teclado con SDL”

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>

SDL_Surface* cargarImagen(char *nombreArchivo)
{
    SDL_Surface *image, *bmp;
    image = IMG_Load(nombreArchivo);
    if(!image) return 0;
    Uint32 color_key =SDL_MapRGB(image->format, 255, 0, 255);
    SDL_SetColorKey(image, SDL_SRCCOLORKEY|SDL_RLEACCEL, color_key);
    bmp=SDL_DisplayFormat(image);
    SDL_FreeSurface(image);
    if(!bmp) return 0;

    return bmp;
}

void dibujaImagen(SDL_Surface *screen, SDL_Surface *imagen, int x, int y)
{
            SDL_Rect rect;
            rect.x=x;
            rect.y=y;
            SDL_BlitSurface(imagen, 0, screen, &rect);
}

    /*este metodo limpia la superficie que llega por parametros*/
void clear(SDL_Surface *screen)
{
    /* se crea el color negro(0,0,0)*/
  Uint32 col=SDL_MapRGB(screen->format, 0, 0, 0);
    /* se rellena la superficie a limpiar con el color creado (negro)*/
  SDL_FillRect(screen, 0, col);
}
            /* metodo de programa principal */
int main(int argc, char *argv[])
{
    SDL_Surface* screen;
    if (SDL_Init(SDL_INIT_VIDEO)<0)
    {
        printf("Error al iniciar SDL:: %s",SDL_GetError());
        return -1;
    }

    screen = SDL_SetVideoMode(640,480,16,SDL_HWSURFACE|SDL_RESIZABLE);

    if (!screen)
    {
        printf("Error al iniciar el contexto SDL:: %s",SDL_GetError());
        return -1;
    }

    SDL_WM_SetCaption(":: Mouse y Teclado con SDL ::",NULL);

    SDL_Surface* img = cargarImagen("imagenes/mira.bmp");
    if (!img)
    {
        printf("No se pudo cargar la imagen: %s",SDL_GetError());
        return -1;
    }

        /* estas seran las posiciones de la imagen en el programa */
    int x = 100;
    int y = 100;
    dibujaImagen(screen, img, x, y);
    SDL_Flip(screen);

    SDL_Event event;
        /* declaro key para poder usar el teclado */
    Uint8 *key;

    int ok = 1;
```

```
    /*
    estas variables son para los estados del mouse y teclado, en un principio
    el mouse esta desactivado y el teclado activado
    */
int mouse=0;
int teclado=1;


while (ok)
{
        /* atrapo la posible tecla presionada */
    key = SDL_GetKeyState(0);
           /* si la tecla es arriba y el teclado esta activado => arriba... */
    if (key[SDLK_UP] && teclado)
    {
         /* muevo la imagen en -4 unidades en el eje Y,
            esto es hacia arriba*/
            y -=4;
                /* limpio el buffer */
            clear(screen);
                /* dibujo la imagen en su nueva posicion sobre el buffer limpio */
            dibujaImagen(screen, img, x, y);
                /* muestro en pantalla el buffer actualizado */
            SDL_Flip(screen);
    }
           /* si la tecla es abajo y el teclado esta activado => abajo ... */
    if (key[SDLK_DOWN] && teclado)
    {
         /* muevo la imagen en +4 unidades en el eje Y */
            y +=4;
            clear(screen);
            dibujaImagen(screen, img, x, y);
            SDL_Flip(screen);
    }
          /* si la tecla es izquierda y el teclado esta activado => izquierda ... */
    if (key[SDLK_LEFT] && teclado)
    {
         /* muevo la imagen en -4 unidades en el eje X */
            x -=4;
            clear(screen);
            dibujaImagen(screen, img, x, y);
            SDL_Flip(screen);
    }
           /* si la tecla es derecha y el teclado esta activado => derecha ... */
    if (key[SDLK_RIGHT] && teclado)
    {
         /* muevo la imagen en +4 unidades en el eje X */
            x +=4;
            clear(screen);
            dibujaImagen(screen, img, x, y);
            SDL_Flip(screen);
    }
           /* si la tecla es M ... */
    if (key[SDLK_m])
    {
         /* si el mouse estaba desactivado, lo activo y desactivo el
            teclado */
            if (!mouse)
            {
                   /* al activar el mouse, escondo el cursor de la pantalla
                     (puntero del mouse)*/
                 SDL_ShowCursor(0);
                   /* lo que hace WarpMouse es colocar el puntero del mouse en la
                    posicion que se le da, en este caso la posicion en que
                    se encuentra la imagen*/
                 SDL_WarpMouse(x,y);
                 mouse = 1;
                 teclado = 0;
            }
    }
           /* si la tecla es ESCAPE el programa termina */
    if (key[SDLK_ESCAPE])
           ok = 0;

           /* estos son los eventos generados por el MOUSE */
    if(SDL_PollEvent(&event))
    {
```

```
                /* evento en el que cierran la ventana */
            if (event.type==SDL_QUIT)
                ok = 0;
                 /* evento en el que se mueve el mouse y el mouse esta activado */
            if (event.type == SDL_MOUSEMOTION && mouse)
            {
                  /* obtengo la posicion del mouse */
                SDL_GetMouseState(&x,&y);
                clear(screen);
                dibujaImagen(screen, img, x, y);
                SDL_Flip(screen);
            }
                  /* si el evento es que se presiono un boton del mouse */
            if (event.type == SDL_MOUSEBUTTONDOWN && mouse)
            {
                  /* vuelvo a mostrar el cursor del mouse, activo el teclado
                     y desactivo el mouse */
                SDL_ShowCursor(1);
                mouse = 0;
                teclado = 1;
            }
        }
    }
    return 1;
}
```

Fig.15 Ejecución del programa "Mouse y Teclado con SDL".

El programa carga una imagen que ustedes deseen, la que se podrá mover con las flechas del teclado, y se si presiona "m" la imagen se podrá mover con el Mouse hasta que se haga un clic de Mouse donde nuevamente el control queda desde teclado.

Si el código les parece muy poco ocurrente, entonces prepárense, por que las problemáticas que aparecen en la programación de juegos son aún peores, ya que tendrán que lidiar con algoritmos de **detección de colisiones** que son los choques entre figuras, personajes y escenario. Y la **física de juegos** como por ejemplo tratar de hacer saltar o rebotar una pelota hasta que se detenga por completo. Todo lo anteriormente mencionado tiene una complejidad extra la que se incrementara cuando se trate de hacer esto mismo en un ambiente de tres dimensiones.

5 Detección de Colisiones

Este es un campo muy interesante para investigación, ya que las técnicas que existen son por lo general muy personales de cada programador y dada su complejidad, a mi personalmente me aburre el tratar de hacerlas mas eficientes. Las siguientes

explicaciones vienen de experiencias personales y códigos bastante malos que hice hace ya bastante tiempo atrás.

Existen dos métodos muy utilizados para calcular las colisiones que son:

- ✓ Bounding Box (bordes de la caja o rectángulo)
- ✓ Píxel Collision (colisión al píxel)

5.1 Bounding Box

Quizás la técnica mas simple, rápida pero menos precisa para las colisiones 2D, y la manera en que funciona es que los personajes o actores que colisionan sean tomados como rectángulos y su uno rectángulo se solapa a otro hay colisión.

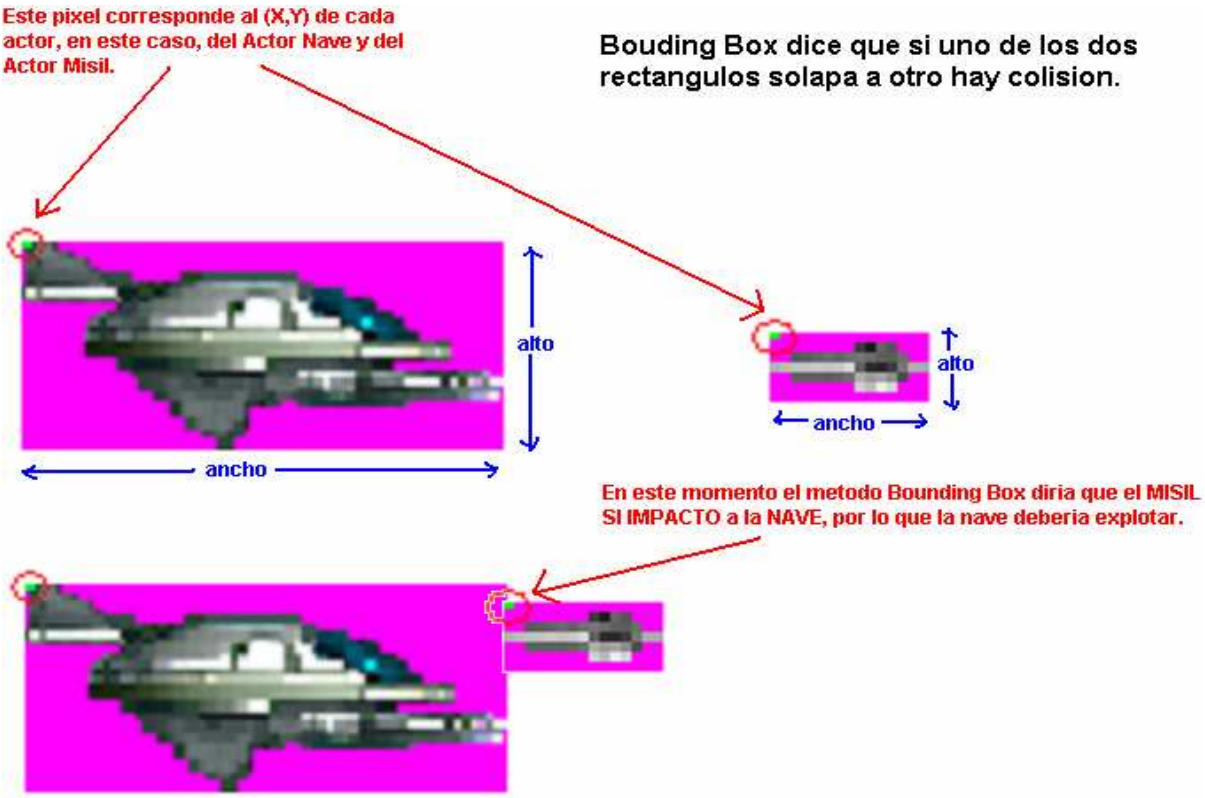

Fig.16 Método de Bounding Box

La manera de codificar este tipo de colisiones es bastante simple ya que deberíamos tener la posición X e Y de cada actor y además cuando cargamos una imagen en SDL_Surface* podemos obtener la información del ancho (w) y alto del actor (h), entonces:

```
        /* posición de la nave (100,100) */
int xNave,yNave = 100;
SDL_Surface* nave =cargarImagen("nave.bmp");
        /* posición del misil (300,100) */
int xMisil = 300;
int yMisil = 100;
SDL_Surface* misil =cargarImagen("misil.bmp");
...
        /* asi se saca el ancho y alto de una imagen en SDL */
printf("El area de la imagen MISIL es: %i", misil->w * misil->h);

      /* posible algoritmo bounding box de un misil que viene de derecha
          a izquierda. */
/* este if se puede explicar mas menos asi:
    SI la posision X del misil esta dentro del ancho del rectangulo de la nave
```

```
                        Y
     La posision Y del misil esta entre el rango del alto del rectangulo de la nave
     Entonces: COLISIONAN!!!  */
if ((xMisil <= (xNave + nave->w)) &&
   ((yMisil + misil->h) >= yNave) && (yMisil <= (yNave + nave->h)))

       printf("El misil colisiono con la nave en [%i,%i]",xMisil,yMisil);
```

5.2 Píxel Colision

Sobre este método de colisión, explicare a grandes rasgos de que se trata, ya que personalmente nunca lo implemente.

Bueno la idea es tomar solo la representación grafica del actor, de la nave en este caso, como una serie de arreglos de bits para que al ejecutar el algoritmo de colisiones se pueda saber si hay o no hay píxel perteneciente al actor chocando con los píxeles del otro actor, entonces se representa al actor con varios arreglos de bits donde un 0 significa que no hay píxel y un 1 que hay un píxel.

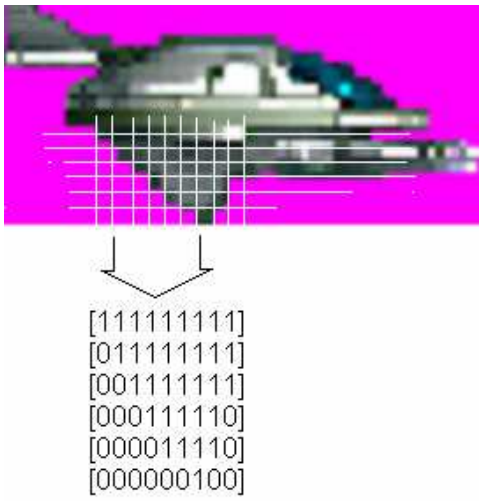

Fig.17 Interpretación del actor Nave por medio de arreglos de bits para el método **Píxel Colision.**

6 Obteniendo información del hardware

Esta parte es importante y bastante útil para cuando se desarrollen aplicaciones OpenGL con SDL, ya que podemos obtener información del computador en el que correrá la aplicación, información como soporte para hardware de video y aceleración grafica.

```
        /* queremos modo ventana */
int videoFlags |= SDL_RESIZABLE;
const SDL_VideoInfo *videoInfo = SDL_GetVideoInfo();
if(!videoInfo)
{
        printf("ERROR: No se pudo capturar la informacion de video\n");
        return -1;
}
 /*Si tenemos renderizado por hardware, la añadimos a las FLAGS*/
if(videoInfo->hw_available)
{
        videoFlags |= SDL_HWSURFACE;
        videoFlags |= SDL_HWPALETTE;
}else
        videoFlags |= SDL_SWSURFACE;
        /*acceleracion por Hardware permitido?*/
if (videoInfo->blit_hw)
        videoFlags |= SDL_HWACCEL;
        /*finalmente cuando recuperamos toda la informacion la agregamos a
        SDL_SetVideoMode()*/
screen = SDL_SetVideoMode(640,480,16,videoFlags);
```