

Creación de un juego

Programación con SDL

En este artículo, el primero de una serie, Steven Goodwin da un vistazo a SDL. Qué es, cómo funciona y, lo que es mucho más importante, cómo usarlo para escribir un flamante juego, *Explorer Dug*.

POR STEVEN GOODWIN

SDL significa Simple DirectMedia Layer y es una API multi-plataforma para programar aplicaciones multimedia, como los juegos. Aporta una base estable sobre la que los programadores pueden trabajar, sin preocuparse de cómo se encargará el hardware de renderizarlo o incluso *qué* hardware lo ejecutará.

SDL está disponible bajo la licencia LGPL (Licencia Pública General Menor), y como tal no requiere que el código fuente de la aplicación que se base en ella sea publicado.

Actualmente existen buenas implementaciones de SDL para Linux (i386, PPC, y PS2), BSD, Windows, Macintosh OS 9/X, Solaris, Irix y BeOS.

Acceso versus Emulación

La API SDL está compuesta de cinco subsistemas distintos: vídeo, audio, CDROM, entrada de joystick y temporizadores. Cada uno apunta a un área de desarrollo diferente y puede ser inicializada y usada independientemente de los demás. Además, SDL fomenta el uso de librerías que puedan suministrar nuevas funcionalidades a la API básica.

Las dos librerías más comunes son *SDL_mixer* (que proporciona un mejor manejo de audio) y *SDL_Image*, que proporciona soporte para una amplia selección de formatos de ficheros gráficos, incluyendo, GIF, JPG, PNG y TGA. El sitio principal de SDL [1] lista 109 librerías diferentes, actualmente en producción, suministrando un amplio



Figura 1: La pantalla de bienvenida

rango de conjuntos de características desde primitivas gráficas básicas, hasta redes y soporte para tipografías True Type (TTF). La mayoría de estas librerías han sido enviadas por usuarios finales y no forman parte del paquete principal de SDL. Esto no es un problema en sí mismo, pero si el propósito principal es usar SDL para trabajar desarrollando para varias plataformas, el desarrollador habrá de asegurarse de que las librerías que pretende utilizar se encuentran disponibles para las plataformas con las que vaya a trabajar, ya que no todas están completamente soportadas.

La herencia de Windows en SDL es muy evidente dentro de la API, tanto es así que mucha de la terminología (y muchas de las funciones) son muy parecidas a sus equivalente en DirectX. A pesar de ello, no se emula Windows de ninguna manera. En cambio, cada llamada a la API gráfica de SDL, por ejemplo, utiliza directamente el controlador desde el sistema operativo anfitrión, bastante distinto a lo que sería un sistema emulado.

En Windows, esto implicaría el uso de DirectX. Bajo Linux, se utiliza una de las librerías del dispositivo de gráficos, como X11 o DGA. También está soportado el control MTR (Memory Type Range Register, [2]) aplicaciones aceleradas a pantalla completa (véase el Cuadro 1: Dispositivos).

La API de audio puede reclutar los servicios de OSS, ESD o aRts para suministrar música y efectos de sonido.

Gracias al hecho de que se utiliza el acceso directo (en vez de la emulación) a los dispositivos del sistema, se puede lograr un gran nivel de rendimiento en todas las plataformas.

¡Excavad, Excavad, Malditos!

Hay cientos de juegos SDL disponibles en Internet y muchos de ellos están disponibles con todo su código fuente. Contribuiremos a esta colección con un pequeño juego de plataformas llamado *Explorer Dug*. Lo programaremos en C (ya que probablemente sea la forma en que se entenderá mejor), aunque se puede escribir aplicaciones SDL en muchos otros lenguajes, como PHP, Ruby, Perl, Java y LISP. Principalmente nos concentraremos en las características de SDL y cómo usarlas para producir un juego. También destacaremos las áreas del desarrollo del juego que requiere más trabajo de nosotros los programadores ¿De acuerdo? ¡Bien! Comencemos...

Gran Bola de Fuego


La primera tarea es descargar e instalar SDL. La versión actual estable es la 1.2.7, y está disponible desde [3] como un archivo tar comprimido con gzip. Además de distribuirse el código fuente, están disponibles dos versiones binarias

Cuadro 1: Drivers

SDL no está limitado a X11 como dispositivo de salida. También puede usar dga, fbcon ¡y hasta aalib! Para hacer uso de estas características, debemos asegurarnos de que el dispositivo apropiado se compila dentro de SDL (lo cual requerirá otro *./configure, make, make install* del paquete). Entonces podrá usar la variable de entorno *SDL_VIDEO_DRIVER* para indicar que dispositivo quiere usar. Por ejemplo:

```
export SDL_VIDEO_DRIVER=aalib
./explore
```

Puede encontrar una lista de los dispositivos suministrados con SDL tecleando:

```
./configure -help | grep 
enable-video
```

diferentes: Ejecución, runtime, y desarrollo (development). Naturalmente, la última proporciona mejores facilidades de depuración, lo cual es más útil para nosotros, como desarrolladores. De todas formas recomiendo trabajar con las fuentes ya que esto nos permite un mejor ajuste y el uso completo de un depurador para introducirnos en el mismísimo código de SDL. Esto no solo es útil en el seguimiento de errores, también puede ser muy iluminador.

El archivo tar.gz se desempaqueta de la manera habitual:

```
tar xfz SDL-1.2.7.tar.gz
```

Y se instala con el proceso igualmente familiar:

```
./configure
make
make install
```

Por defecto, esto colocará las librerías necesarias y los archivos incluidos en `/usr/local/lib` y `/usr/local/include/SDL` respectivamente, aunque esto puede cambiarse en el proceso de inicio con:

```
./configure -prefix=/poner/sdl/
algunsitio/encualquierotraparte
```

Además de suministrar dispositivos adicionales, debería tener pocos motivos para repetir el proceso `./configure` ya que SDL es fácil de configurar y muy estable, e invariablemente funciona a la primera.

El paquete SDL también contiene un extenso juego de páginas de manual (sección 3) detallando los parámetros e indicadores de cada función.

Arriba con ese muro

El paquete SDL viene con un pequeño conjunto de pruebas que permiten comprobar la integridad de cada subsistema. Lo encontraremos en el directorio `test`, y funcionará nada más terminar la instalación, siempre que haya salido todo bien. En el caso de que surjan problemas, buscaremos los mensajes de error que cada prueba envía a la línea de comandos, ya que pueden indicar si son problemas con el hardware o la configuración. Si la máquina es capaz de ejecutar un escritorio, debería estar a la altura de cualquier tarea con SDL

Todos los programas SDL pueden comenzar con una de estas líneas:

```
#include "SDL.h"
```

o:

```
#include "SDL/SDL.h"
```

La primera es preferible por motivos de portabilidad en varias plataformas, aunque esto requerirá que se añada el directorio SDL a la lista de rutas incluidas que `gcc` buscará. Todas las demás cabeceras de archivos SDL están incorporadas dentro de ésta para así limitar el mantenimiento. También necesitamos enlazar la librería principal de SDL (*libSDL*) dentro de nuestra aplicación. Al principio solo usaremos las funcionalidades de esta librería. Según pase el tiempo y nuestros requisitos crezcan, añadiremos otras librerías. Hasta entonces, los dos requisitos de más arriba pueden satisfacerse en la línea de comandos con:

```
gcc -I/usr/include/SDL -lSDL
test/testsprite.c
```

Si cambia las rutas de librerías e incluidos, se puede usar el programa `sdl-config` para obtener esta información (véase Cuadro SDLCONFIG).

El Mundo en Nuestras Manos

Nuestra primera incursión en la programación con SDL será la pantalla de "bienvenida" del juego. Con esto abarcaremos un número de conceptos fundamentales, tal como superficies, blitting (de blit: Block Image Transfer) y actualización de pantalla.

Cada estructura y función SDL comienza con el prefijo `SDL_`. Además envolveremos estas funciones dentro de las nuestras (todas precedidas con 'ex') para encapsular más funcionalidad y suministrar un lugar común donde reunir el código del juego y el de SDL. Esto ayudará en la depuración.

La primera etapa es inicializar SDL. Hay dos funciones para hacer esto, `SDL_Init` y `SDL_InitSubSystem`. `SDL_Init` puede inicializar uno (o más) subsistemas y debería ser la primera función SDL en un programa. Aunque podemos inicializar subsistemas adicionales más

tarde, deberíamos comenzar siempre con un `SDL_Init`, porque incluye la inicialización de otros componentes que `SDL_InitSubSystem` no tiene. Muestras de estos componentes podrían ser: Las hebras son inicializadas, el último código de error es borrado y el paracaídas es (opcionalmente) desplegado (véase Cuadro Paracaídas).

```
SDL_Init(SDL_INIT_VIDEO |
SDL_INIT_AUDIO);
```

Es exactamente igual a:

```
SDL_Init(SDL_INIT_VIDEO);
SDL_InitSubSystem
(SDL_INIT_AUDIO);
```

Por ahora, solo necesitamos el subsistema de vídeo, por tanto lo inicializaremos y comprobaremos si hay errores. SDL adopta el estándar de usar valores negativos para indicar un código de error. El valor de este número indica el error concreto.

SDL también suministra la función `SDL_GetError` que devuelve el nombre textual del último error. Pero debe ser llamado inmediatamente, porque si la subsiguiente función SDL falla, el anterior error se perderá.

Cerrar el sistema es tan simple como llamar a la función `SDL_Quit`.

```
void exRelease(void)
{
    SDL_Quit();
}
```

Cuadro 2: SDLCONFIG

`sdl-config` es un pequeño programa que viene empaquetado con SDL y su función es básicamente informar o prefijar, el lugar de la instalación de SDL. También se puede usar para dar los indicadores correctos tanto al compilador como al enlazador.

```
$ sdl-config
Usage: sdl-config [--
prefix[=DIR]] [--exec-
prefix[=Dir]] [--version]
[--cflags] [--libs][--static-
libs]
$ sdl-config -libs
-L/usr/local/lib -Wl, -
rpath, /usr/local/lib -lSDL -
lpthread
```

De igual modo que se puede inicializar un subsistema individualmente, también puede cerrarlo de una manera similar:

```
SDL_QuitSubSystem(
  ( SDL_INIT_VIDEO );
```

La función final siempre debe ser *SDL_Quit*, ya que esto retirará lo manejadores de señales del paracaídas y terminará cualquier hebra que quede activa.

La primera vez que vi tu cara

En SDL todos los gráficos se almacenan en 'superficies'. Es un termino que se ha tomado prestado de DirectX y que significa 'memoria gráfica'. La pantalla es una superficie. La imagen del fondo es una superficie. Los personajes son superficies, etcétera. Las superficies pueden ser creadas y destruidas varias veces a lo largo de la vida del juego. Sin embargo, sólo puede haber una superficie de pantalla. Es esta superficie de pantalla la que aparece en el monitor y, por tanto, cualquier cosa que quiera que sea visible debe ser dibujada encima de esta superficie.

Las superficies pueden ser de un tamaño arbitrario y son retenidas en dos lugares: memoria de vídeo o memoria de sistema. La memoria de vídeo es más rápida y es conocida como "superficie de hardware". La memoria de sistema (o memoria software) es parte de la RAM normal y marcada internamente por SDL como 'datos gráficos retenidos'.

La pantalla se inicializa con el comando especial *SetVideoMode*, aunque la superficie que devuelve no es diferente de cualquier otra.

```
SDL_Surface *pScreen;
pScreen = SDL_SetVideoMode(
  (640, 480, 16, SDL_HWSURFACE);
```

Aquí tenemos una configuración de superficie de vídeo a 640x480. El 16 indica la profundidad de bit, el cual refleja el número total de colores disponibles. En este caso, 65.536 (2^{16}), lo cual nos da un buen equilibrio entre calidad visual y velocidad. Las opciones habituales son 8, 16 y 24 aunque SDL puede soportar 12 y 15 (véase el cuadro Carga de Profundidad).

Normalmente, en el desarrollo de software el programador está limitado por el hardware con el que trabaja y obligado a doblegarse a sus exigencias sin rechistar. Los juegos, afortunadamente, son un mundo aparte. Hemos elegido el tamaño de pantalla y la profundidad de bit para hacer alarde de nuestros gráficos en el mejor entorno posible. Si no utilizamos suficientes colores, por ejemplo, gráficos esenciales como la *llave* o la *puerta de salida* pueden ser difíciles (o imposibles) de ver. Esto es injusto para el jugador y por eso es admisible salir del juego si esta resolución no se puede conseguir.

Por otra parte, si se está usando SDL para aplicaciones que no son juegos, o no importa la degradación de la imagen, se puede usar cualquier modo de vídeo que el usuario pueda tener ajustado en su escritorio poniendo la profundidad de bit a cero:

```
pScreen = SDL_SetVideoMode(
  (640, 480, 0, SDL_HWSURFACE);
printf("The bit depth is set to
%d\n", pScreen->format->
BitsPerPixel);
```

Cuadro 3: Paracaídas

El paracaídas es una manera de capturar señales (tales como fallos de segmentación, errores de bus, cauces rotos y excepciones de punto flotante) lo cual da a SDL una oportunidad para llamar a *SDL_Quit* y liberar sus recursos. El paracaídas se instala automáticamente en la inicialización, si no queremos utilizarlo, debemos empezar la aplicación con:

```
SDL_Init ( SDL_INIT_VIDEO |
  SDL_INIT_NOPARACHUTE );
```

La aplicación debe suministrar sus propios manejadores de señales si es necesario, si bien conviene recordar que, en aras de la portabilidad, no todas las plataformas soportan todas las señales y algunas plataformas no soportan señales en absoluto.

Véase el cuadro Funciones de vídeo.

El parámetro final es un juego de indicadores a nivel de bit. Estos especifican atributos para la superficie y la ventana en la que es mostrado. El indicador *SDL_HWSURFACE* requiere que la superficie de la pantalla, si es posible, deba ser creada en memoria de vídeo. Si nos encontramos que el dispositivo no soporta superficies de hardware (como en el caso de X11), o que la memoria hardware está llena, la superficie puede ser creada, pero software.

Si es apropiado crear una superficie hardware para un búfer en particular (como por ejemplo para la pantalla principal), seguimos teniendo que invocar *SDL_HWSURFACE*. Puede que nosotros estemos utilizando X11, pero otro usuario podría no estar haciéndolo y tenemos que darles el mejor juego que podamos.

Listing 1: Initialise

```
01 BOOL exInitSDL(void)
02 {
03   if (SDL_Init(SDL_INIT_VIDEO)
    < 0)
04   {
05     fprintf(stderr, "No se
    pudo inicializar video SDL:
    %s\n", SDL_GetError());
06     return FALSE;
07   }
08
09 return TRUE;
10 }
```

Listing 2: First screen

```
01 SDL_Surface *pImg;
02
03 if ((pImg = SDL_LoadBMP("welcome.bmp")))
04 {
05   /* Hemos cargado con éxito la imagen */
06
07   SDL_BlitSurface(pImg, NULL, pScreen, NULL); /* transferir todo */
08   SDL_UpdateRect(pScreen, 0,0,0,0); /* toda la pantalla */
09   SDL_Delay(2*1000); /* 2 segundos */
10
11   /* A continuación liberamos la superficie */
12   SDL_FreeSurface(pImg);
13 }
```

Además está el parámetro final que permite redimensionar la ventana (*SDL_RESIZABLE*), quitarle el marco (*SDL_NOFRAME*) o soportar renderizado OpenGL (*SDL_OPENGL*). A menudo, el primer impulso con un juego es hacerlo a pantalla completa (*SDL_FULLSCREEN*).

Pero nos quedaremos con una versión en ventana ya que es más fácil trabajar con ella. Además, es posible hacer que la máquina se vuelva inutilizable si se está ejecutando el juego a pantalla completa, desde donde no es siempre posible conmutar a otro escritorio virtual y matar el programa. Esto también ocurre si se tropieza con un punto de ruptura en el depurador o se nos olvida aportar algún mecanismo para abandonar el juego. Recordemos que, al igual que un depurador, el paracaídas de SDL captura varias señales (como Ctrl + C) y, por tanto, éstas no están siempre disponibles.

Cada superficie que se crea (y eso incluye a la pantalla) debe ser liberada con la función *SDL_FreeSurface*, una vez que ha dejado de usarse:

```
SDL_FreeSurface(pScreen);
```

Mis Posesiones Terrenales

Además de la pantalla, necesitamos algunas superficies de nuestra propiedad, dentro de las cuales podremos dibujar nuestros gráficos. Hay dos formas principales de crear nuestras propias superficies. La primera es crear la superficie manualmente con una de las siguientes funciones:

```
SDL_Surface *SDL_CreateRGBSurface(Uint32 flags, int width,
```

Cuadro 4: Funciones de vídeo

Estas funciones pueden suministrar alguna idea útil sobre que modos de vídeo son posibles con SDL. Las páginas de manual tienen prototipos completos y una explicación de estas funciones.

<code>SDL_GetVideoInfo</code>	Recupera información acerca del hardware de vídeo.
<code>SDL_VideoDriverName</code>	Da el nombre del dispositivo de vídeo.
<code>SDL_ListModes</code>	Enumera todas las resoluciones de pantalla disponibles.
<code>SDL_VideoModeOK</code>	Determina si un modo específico de vídeo está disponible.

```
, int height, int depth, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask);
```

```
SDL_Surface *SDL_CreateRGBSurfaceFrom(void *pixels, int width, int height, int depth, int pitch, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask);
```

Esto se utiliza raramente por que tiene que escribir cada pixel de la imagen (en el formato correcto) directamente dentro de la superficie. Además, normalmente hay una gran cantidad de pixels.

Una forma mucho más fácil es dibujar las imágenes con GIMP y salvarlas como un BMP. Entonces se puede cargar la imagen, correctamente formateada y copiarla dentro de una flamante superficie. SDL permite esta generosa funcionalidad con una simple función llamada:

```
SDL_Surface *SDL_LoadBMP(const char *file);
```

SDL solo provee soporte a archivos BMP dentro de la librería estándar. Aunque *SDL_image* suministra soporte para muchos otros formatos, nuestro juego se limitará a BMPs y hará uso de varias

Cuadro 5: Carga de profundidad

Cuando la profundidad de bit es distinta de 8, los datos del pixel se almacenan en un "formato empaquetado". Esto se llama así porque el color se representa con tres números, uno para cada uno de los componentes rojo, verde y azul que se empaquetan juntos (en un simple *Uint* o *Uint32*) para mostrar el color. Con una profundidad de 24 bits (a veces llamado Color verdadero), cada uno de los componentes RGB ocupa 8 bits. Esta es la resolución más alta que probablemente necesitemos, pero normalmente es excesiva para la mayoría de los juegos. Desafortunadamente las cosas se vuelven más complicadas con el más común de los formatos empaquetados. Me refiero al color de 16 bits. En este caso los componentes RGB usan 5, 6, y 5 bits para cada color o ¡5, 5 y 6 o 6, 5 y 5! El formato exacto de la superficie puede variar dependiendo de donde esté almacenado y que tarjeta gráfica se esté usando. La orden también puede variar si se está usando un PowerPC, por ejemplo, debido a la cuestión de la orientación de los datos (endian). Sin embargo, esto no es algo de lo que debamos preocuparnos, ya que desde SDL se convierten los formatos automáticamente durante una transferencia de bloque o blit. Si se necesita comprender el formato interno (tal como veremos más tarde en esta serie) es agradable saber que SDL suministra la función *SDL_MapRGB* para ayudarnos. En verdad, este problema con el formato de los paquetes también se manifiesta con el modo de color de 24 bits, pero es menos pronunciado. Las reglas cambian cuando se especifica una profundidad de bit de 8. En vez de separar los 8 bits dentro del componente RGB, cada valor (desde 0 a 255) referencia a una tabla separada que indica el color actual. Esta

tabla se llama la paleta (la cual usa 24 bits completos para almacenar la información de color) y puede ser configurada con las funciones *SDL_SetColors* y *SDL_SetPalette*. Aunque hoy en día ha decaído en lo que se refiere a la programación de juegos, gracias a este formato, se puede mantener la producción de gráficos de alta calidad en hardware muy limitado. También se pueden producir muchos efectos interesantes (y muy rápidos) cambiando los colores en la paleta sin hacer cambios en cada pixel sobre la pantalla. El mayor problema con las superficies de 8 bits paletizadas es que solo se dispone de 256 colores *específicos* para toda la imagen. Si el fondo usa un juego de 256 colores y el personaje principal usa otro diferente, entonces SDL cambiará automáticamente algunos de los colores. Si bien esto no es algo demasiado malo, los resultados pueden ser impredecibles y por lo tanto hará que el trabajo artístico sea un poco menos impresionante. La otra cara de la moneda, sin embargo, es que mover datos de 8 bits es el doble de rápido que mover datos de 16 bits y, por tanto, es a menudo un truco empleado para dispositivos de mano (PDAs, móviles, etc.). Para asegurar la compatibilidad en varias plataformas, los tipos estándar de C como *short* e *int*, no se usan. En su lugar, SDL define los suyos propios, que son usados por toda la API. Estos tipos pueden cambiarse sobre nuevas plataformas para asegurar que un *Uint16*, por ejemplo, es siempre de 16 bits.

```
typedef unsigned char   Uint8;
typedef signed char     Sint8;
typedef unsigned short  Uint16;
typedef signed short    Sint16;
typedef unsigned int    Uint32;
typedef signed int      Sint32;
```

superficies cargadas de esta manera. Cada superficie retendrá un juego de gráficos específico: Uno para el telón de fondo, otro para el jugador, otro para el enemigo, etcétera.

La imagen final del juego, por tanto, se construirá a partir de muchas copias de superficies a superficie (recordemos que la pantalla es solo una superficie), aunque gobernadas por la lógica del juego. Este proceso de copiado se llama *blitting* y es la abreviatura de *B*lock *I*mage *T*ransfer.

El Baile del Blit

Podemos transferir bloques (blits) entre dos superficies cualquiera y eso incluye desde una superficie a sí misma. También podemos transferir bloques desde una porción de una superficie, a una porción diferente de otra superficie. La única limitación es que el tamaño de ambas porciones (la fuente y el destino) deben ser del mismo tamaño. La operación de transferencia de bloques se puede (si ambas superficies están en memoria de vídeo) ejecutar por hardware, lo cual es increíblemente rápido. No todos los dispositivos gráficos, no obstante, soportan la aceleración por hardware, así que para un breve recordatorio, véase el cuadro Dispositivos.

Hay una sola función para hacer transferencias de bloques. Es única así que es sencillo de recordar:

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

Desde esta función no se dispone de la capacidad de estirar o rotar la superficie; para esto deberemos recurrir a cualquiera de las librerías *SDL_gfx* [4] o *SGE* [5]. Como el estirar bitmaps consume mucho tiempo de proceso no lo usaremos en nuestro juego, ya que estamos aspirando a la máxima velocidad. Por tanto generaremos nuestros gráficos al tamaño exacto que necesitamos.

SDL_Rect es una estructura simple para indicar el tamaño del área que queremos transferir, tomando como *x*, *y*, la anchura y la altura. *SDL* cortará automáticamente nuestras coordenadas internamente si excedemos los límites de

cualquiera de las dos superficies, fuente o destino. Esto es muy útil, ya que podemos dibujar gráficos en posiciones tales como (-4, -2) o (600, 450), lo cual permite dividir suavemente nuestros gráficos por el borde de la pantalla. *SDL* resuelve como representar la porción visible óptimamente.

```
SDL_Rect srcrect =
    {600, 450, 64, 64};
```

Solo transferiremos bloques desde 600, 450 a 639, 479 aunque la extensión sea 663, 513. Si se quiere transferir en bloque la superficie entera (como hicimos en la pantalla de bienvenida) entonces pasaremos *srcrect* como *NULL*. Solo usamos las coordenadas *x*, *y* de *dstrect* refiriéndonos a la esquina superior izquierda del área de destino, porque no somos capaces de estirar y transferir bloques. Pasando *NULL* como valor *dstrect* le diremos a *SDL* que comience la transferencia de bloque desde (0,0). Si la extensión del bloque transferido excede la superficie de destino la imagen se recortará normalmente. Esta área recortada puede ser limitada artificialmente con:

```
void SDL_SetClipRect(
    SDL_Surface *surface,
    SDL_Rect *rect);
```

Y cualquier futura transferencia de bloque a esa superficie solo ocurrirá dentro del área especificada en el rectángulo. Como dije más arriba, pasando *NULL* como puntero *SDL_Rect*, le estamos diciendo a *SDL* que use el área completa de la superficie, lo que efectivamente eliminara el área recortada. Esta característica nos permite mantener un área de la pantalla pura y sagrada, sin tener en cuenta que componentes del juego intenten transferir bloques allí, protegiendo información tal cómo la puntuación, o el número de vidas.

¡Bienvenidos!

Así que ahora podemos cargar una imagen en una superficie y transferir un bloque de esa superficie a otra superficie (tal como la pantalla). A fin de ver como queda nuestro trabajo, debemos dar a conocer una revelación más. La superfi-

cie pantalla que hemos configurado no es la imagen que se visualiza en la ventana. Esa imagen se controla por el dispositivo (tal como *X11*), no por *SDL*. A fin de ver la imagen, debemos decirle a *SDL* que actualice el dispositivo con el gráfico de nuestra superficie. Para hacer esto usaremos la función *SDL_UpdateRect*.

```
void SDL_UpdateRect(
    SDL_Surface *screen,
    Sint32 x,
    Sint32 y,
    Sint32 w,
    Sint32 h);
```

Sólo una pequeña porción rectangular de la pantalla es actualizada (lo cual es naturalmente más rápido que si actualizamos toda la superficie). Como alternativa, podemos elegir pasar ceros a cada uno de los parámetros y actualizar todo el área de golpe. Por ejemplo:

```
SDL_UpdateRect(
    pScreen, 0, 0, 0, 0);
```

Ahora hemos transferido algunos bloques a nuestra pantalla y provocado la actualización del monitor y así actualizarlo con nuestra propia imagen. De esta manera, si añadimos un pequeño retraso, completaremos nuestra primera pantalla.

Producir un nivel del juego o personajes animados, es simplemente una cuestión de más transferencias de bloques, en más lugares, desde más superficies. No implica nada más que lo que ya hemos visto. Desafortunadamente, hay buenas y malas maneras de hacer esto. El próximo mes le daremos un vistazo a las buenas maneras, mostraremos como las superficies pueden ser usadas para crear la pantalla del juego ¡y daremos vida a algunos de los malos!

INFO

- [1] *SDL*: <http://www.libsdl.org>
- [2] Memory Type Range Register: <http://www.linuxvideo.org/user/mtrr.txt>
- [3] Descargas de *SDL*: <http://www.libsdl.org/download-1.2.php>
- [4] Librería *SDL_gfx*: http://www.ferzkopp.net/~aschiffler/Software/SDL_gfx-2.0/
- [5] Librería *SGE*: <http://www.etek.chalmers.se/~e8cal1/sge/index.html>