SDL: Segundos pasos

Programación de Vídeo Juegos

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

javi.pacheco@terra.es

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

Tabla de contenidos

Capítulo 1. Eventos

Lo primero que vamos a estudiar en este artículo es el manejo de eventos con SDL. Con los eventos vamos a poder capturar las acciones que mande el usuario desde el teclado, ratón o joystick.

1.1. ¿Cómo trabajar con eventos?

Como hemos dicho los eventos nos ayudarán a capturar las intenciones del usuario en nuestro juego. Para ello SDL nos provee de una estructura llamada SDL_Event, la cual se encontrará escuchando hasta que se produzca algún evento en el juego. La estructura es la siguiente:

```
typedef union{
  Uint8 type;
  SDL_ActiveEvent active;
  SDL_KeyboardEvent key;
  SDL_MouseMotionEvent motion;
  SDL_MouseButtonEvent button;
  SDL_JoyAxisEvent jaxis;
  SDL_JoyBallEvent jball;
  SDL_JoyHatEvent jhat;
  SDL_JoyButtonEvent jbutton;
  SDL_ResizeEvent resize;
  SDL_ExposeEvent expose;
  SDL_QuitEvent quit;
  SDL_UserEvent user;
  SDL_SywWMEvent syswm;
} SDL_Event;
```

Una vez que tenemos nuestra estructura sólo tenemos que utilizar varias funciones para poder manejarlos. La primera función que vamos a ver es "SDL_PollEvent". A esta función se le pasa una referencia al evento que tengamos creado y devuelve 1 en el caso que exista algún evento pendiente y 0 en el caso que no. Una vez que sepamos que se ha producido un evento, lo primero es saber de que tipo es, para poder actuar en consecuencia. Para ello miramos la variable type de la estructura del evento. Esta puede contener cualquier de estos valores:

- SDL_KEYDOWN: se produce cuando se presiona una tecla (key)
- SDL_KEYUP: se produce cuando se suelta la tecla (key)
- SDL_MOUSEMOTION: se produce cuando se mueve el ratón (motion)
- SDL_MOUSEBUTTONDOWN: se produce cuando se presiona un botón del ratón (button)
- SDL_MOUSEBUTTONUP: se produce cuando se suelta un botón del ratón (button)
- SDL_JOYAXISMOTION: se produce cuando se mueven los ejes del joystick (jaxis)
- SDL_JOYBUTTONDOWN: se produce cuando se presiona un botón del joystick (jbutton)
- SDL_JOYBUTTONUP: se produce cuando se suelta un botón del ratón (jbutton)
- SDL_VIDEORESIZE: se produce cuando se redimensiona la pantalla (resize)
- SDL_QUIT: se produce al cerrar la ventana de la aplicación (quit)

Existen más tipos de eventos, y hasta pueden ser creados por el usuario, pero no es objetivo de este artículo. Cada uno de estos eventos tienen una varaible asociada en la estructura con la información necesaria para poder manejarlo. Esa variable es la que se encuentra entre paréntesis. Durante los artículos veremos como manejar algunos eventos, pero no todos. Se puede encontrar toda la información necesaria en la documentación de SDL.

A continuación vamos a poner un ejemplo de lo que podrían ser los eventos de un juego.

```
while (SDL_PollEvent(&event)) {
  // Cerrar la ventana
  if (event.type == SDL_QUIT) {
   // código que se ejecuta al cerrar la ventana
  }
  // Pulsando una tecla
  if (event.type == SDL_KEYDOWN) {
// código que se ejecuta al pulsar una tecla
// Comprobamos la tecla pulsada
if (event.key.keysym.sym==SDLK_P) {
// código que se ejecuta al pulsar la tecla P
}
if (event.key.keysym.sym==SDLK_ESCAPE) {
// código que se ejecuta al pulsar la tecla Escape
}
  }
}
```

Por último vamos a ver una función muy interesante y muy útil. Esta es SDL_GetKeyState(NULL); que nos devuelve un Uint8 con el código de la tecla pulsada. En el ejemplo de la sección siguiente podemos ver como trabajar con esta función.

Cáda tecla tiene referenciado un código. En la siguiente lista podemos ver todo el teclado.

- SDLK_BACKSPACE: retroceso
- SDLK_TAB: tabulador
- SDLK_CLEAR: clear
- SDLK_RETURN: return
- SDLK_PAUSE: pausa
- SDLK_ESCAPE: escape
- SDLK_SPACE: espacio
- SDLK_EXCLAIM: ’!’ exclamación
- SDLK_QUOTEDBL: ’"’ dobles comillas
- SDLK_HASH: ’#’ hash
- SDLK_DOLLAR: ’$’ dollar
- SDLK_AMPERSAND: ’&’ ampersand
- SDLK_QUOTE: ”’ comillas simples
- SDLK_LEFTPAREN: ’(’ paréntesis izquierdo
- SDLK_RIGHTPAREN: ’)’ paréntesis derecho
- SDLK_ASTERISK: ’*’ asterisco
- SDLK_PLUS: ’+’ signo de suma
- SDLK_COMMA: ’,’ coma
- SDLK_MINUS: ’-’ signo menos
- SDLK_PERIOD: ’.’ periodo

- SDLK_SLASH: ’/ barra derecha
- SDLK_0: ’0’ número 0. Todos los números hasta el 9 llevan la misma forma.
- SDLK_COLON: ’:’ dos puntos
- SDLK_SEMICOLON: ’;’ punto y coma
- SDLK_LESS: ’<’ signo menor que
- SDLK_EQUALS: ’=’ signo igual
- SDLK_GREATER: ’>’ signo mayor que
- SDLK_QUESTION: ’?’ signo de interrogación
- SDLK_AT: ’@’ arroba
- SDLK_LEFTBRACKET: ’[’ corchete izquierdo
- SDLK_BACKSLASH: ’\’ barra izquierda
- SDLK_RIGHTBRACKET: ’]’ corchete derecho
- SDLK_CARET: ’^’ caret
- SDLK_UNDERSCORE: ’_’ guión bajo
- SDLK_BACKQUOTE: ’‘’ grave
- SDLK_a: ’a’ a. Todas las letras llevan la misma forma.
- SDLK_DELETE: ’^?’ delete
- SDLK_KP0: keypad 0. Todos los números del keypad hasta el 9 llevan la misma forma.
- SDLK_KP_PERIOD: ’.’ periodo del keypad
- SDLK_KP_DIVIDE: ’/’ división del keypad
- SDLK_KP_MULTIPLY: ’*’ multiplicación del keypad
- SDLK_KP_MINUS: ’-’ signo menos del keypad
- SDLK_KP_PLUS: ’+’ signo más del keypad
- SDLK_KP_ENTER: ’\r’ enter del keypad
- SDLK_KP_EQUALS: ’=’ signo igual del keypad
- SDLK_UP: cursor arriba
- SDLK_DOWN: cursor abajo
- SDLK_RIGHT: cursor derecha
- SDLK_LEFT: cursor izquierda
- SDLK_INSERT: tecla insert
- SDLK_HOME: tecla inicio
- SDLK_END: tecla fin
- SDLK_PAGEUP: tecla avanzar página
- SDLK_PAGEDOWN: tecla retroceder página
- SDLK_F1: F1. Todas las funciones "F" llevan la misma forma.
- SDLK_NUMLOCK: bloqueo numérico
- SDLK_CAPSLOCK: bloqueo mayúsculas

- SDLK_SCROLLOCK: scrollock
- SDLK_RSHIFT: shift derecho
- SDLK_LSHIFT: shift izquierdo
- SDLK_RCTRL: ctrl derecho
- SDLK_LCTRL: ctrl izquierdo
- SDLK_RALT: alt derecho
- SDLK_LALT: alt izquierdo
- SDLK_RMETA: meta derecho
- SDLK_RMETA: meta derecho
- SDLK_LMETA: meta izquierdo
- SDLK_LSUPER: tecla windows izquierda
- SDLK_RSUPER: tecla windows derecha
- SDLK_MODE: mode shift
- SDLK_HELP: ayuda
- SDLK_PRINT: imprimir
- SDLK_SYSREQ: SysRq
- SDLK_BREAK: break
- SDLK_MENU: menu
- SDLK_POWER: power
- SDLK_EURO: euro

No es objetivo de este artículo ver todo lo referente a eventos sino dar unas nociones básicas de trabajo con ellos y poder usar el teclado en nuestros juegos.

1.2. Un ejemplo de manejo de eventos de teclado

Para nuestro ejemplo vamos a colocar una nave que podamos mover por la pantalla mediante los cursores. Podremos los eventos necesarios para ello y calcularemos la posición para que nos nos salga de la pantalla. Para ello vamos a crear dos clases llamadas: Frame y Elemento. La clase Elemento tendrá todo lo necesario para poder manejar a nuestra nave: posición en el eje "x", posicion en el eje "y" y todo lo necesario. La clase Frame no cargará las imágenes del elemento para poder dibujarlas en pantalla. En nuestro ejemplo vamos a utilizar la siguiente imagen:

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

```
#include <SDL_image.h>

//******************
//
//   CLASES
//
//******************

// la clase frame contiene la superficie con la imagen

class Frame {

public:
SDL_Surface *img;
void cargar(char archivo[50]);
};


class Elemento {

private:
int x, y; // posicion actual del elemento en la pantalla
Frame sprite; // imagen del elemento

public:
Elemento();
// las siguientes funciones actualizan y recogen los
// valores de posición de la pantalla
void setX(int valor_x) { x = valor_x; }
void setY(int valor_y) { y = valor_y; }
void addX(int valor_x) { x += valor_x; }
void addY(int valor_y) { y += valor_y; }
int getX() { return x; }
int getY() { return y; }
// esta función añade la imagen al elemento
void addFrame(char archivo[50]);
// esta función dibuja la imagen en la pantalla
void dibujar(SDL_Surface *screen);

};

//******************
//
//   CLASE FRAME
//
//******************

void Frame::cargar(char archivo[50]) {

img = IMG_Load (archivo);

}

//******************
//
//   CLASE ELEMENTO
//
```

```
//******************

Elemento::Elemento() {
x = 0;
y = 0;
}

void Elemento::addFrame(char archivo[50]) {
sprite.cargar(archivo);
}

void Elemento::dibujar(SDL_Surface *screen) {
SDL_Rect rect;
rect.x = x;
rect.y = y;
SDL_BlitSurface(sprite.img, NULL, screen, );

}

//******************
//
//   EMPEZAMOS
//
//******************

// esta función crea un fondo de color negro

SDL_Surface* crearFondo() {
/* Creamos las variables necesarias */
SDL_Surface *surface;
Uint32 rmask, gmask, bmask, amask;

/* damos valores a las variables */
rmask = 0xff000000;
gmask = 0x00ff0000;
bmask = 0x0000ff00;
amask = 0x00000000;

surface = SDL_CreateRGBSurface(SDL_SWSURFACE, 640, 480, 16, rmask, gmask, bmask, amask);
if(surface == NULL) {
fprintf(stderr, "CreateRGBSurface ha fallado: %s\n", SDL_GetError());
exit(1);
}
return surface;
}

// esta función dibuja el fondo en la pantalla

void dibujarFondo(SDL_Surface *screen, SDL_Surface *fondo) {
SDL_Rect rect;
rect.x = 0;
rect.y = 0;
SDL_BlitSurface(fondo, NULL, screen, );

}

int main (int argc, char *argv[]) {
```

```
SDL_Event event;
SDL_Surface *screen, *fondo;
Uint8 *keystate; // variable que recoge el valor de la tecla pulsada

int done = 0;

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

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

// creamos el fondo
fondo = crearFondo();

// creamos el elemento
Elemento *nave = new Elemento();
nave->addFrame("nave.png");
nave->setX(300);
nave->setY(300);

while (done == 0) {
dibujarFondo(screen, fondo);
nave->dibujar(screen);
SDL_Flip(screen);
keystate = SDL_GetKeyState(NULL);
if ( keystate[SDLK_UP] && nave->getY()>0 ) nave->addY(-5);
if ( keystate[SDLK_DOWN] && nave->getY() < 410 ) nave->addY(5);
if ( keystate[SDLK_LEFT] && nave->getX()>0 ) nave->addX(-5);
if ( keystate[SDLK_RIGHT] && nave->getX() < 555) nave->addX(5);
// 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) {
if (event.key.keysym.sym==SDLK_ESCAPE) {
done = 1;
}
}
}
}

SDL_Quit();

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

Capítulo 2. Algunas funciones de interés

2.1. Manejo de ventanas

En este capítulo vamos a aprender varios funciones para poder manejar información de las ventanas y poder modicar algunas de sus propiedades.

Con la función void SDL_WM_SetCaption(const char *title, const char *icon); podemos ponerle un título a la ventana y un icono. El icono puede ser NULL si no queremos que lo tenga.

Con void SDL_WM_GetCaption(char **title, char **icon); nos devuelve el título y el icono de la ventana.

Con int SDL_WM_ToggleFullScreen(SDL_Surface *surface); podemos intercambiar de pantalla completa a ventana. Retorna un 1 si ha salido bien y un 0 en el caso de fracasar.

2.2. Nuevas funciones para el manejo de superficies

En el artículo anterior vimos funciones básicas para el manejo de superficies, ahora vamos a ver algunas nuevas para ayudarnos entre otras cosas a que nuestro juego vaya más rápido.

2.2.1. Crear una nueva superficie

Para crear una nueva superficie utizamos la siguiente función:

```
SDL_Surface *SDL_CreateRGBSurface(Uint32 flags, int width, int height,
int depth, Uint32 Rmask, Uint32 Gmask, Uint32 Bmask, Uint32 Amask);
```

Con esta función podremos crear una nueva superficie cuadrada donde podemos agregar nuevos elementos. Estos son los parámetros:

- flags: son diferentes banderas que se le pueden pasar. Estas son SDL_SWSURFACE, SDL_HWSURFACE, SDL_SRCCOLORKEY y SDL_SRCALPHA.
- width: anchura de la nueva superficie.
- height: altura de la nueva superficie.
- depth: la profundidad.
- Rmask: color rojo.
- Gmask: color verde.
- Bmask: color azul.
- Amask: transparencia.

A continuación vemos un ejemplo de como aplicarlo para que quede más claro.

```
/* Creamos las variables necesarias */
SDL_Surface *surface;
Uint32 rmask, gmask, bmask, amask;

/* damos valores a las variables */
#if SDL_BYTEORDER == SDL_BIG_ENDIAN
rmask = 0xff000000;
```

```
gmask = 0x00ff0000;
bmask = 0x0000ff00;
amask = 0x000000ff;
#else
rmask = 0x000000ff;
gmask = 0x0000ff00;
bmask = 0x00ff0000;
amask = 0xff000000;
#endif

surface = SDL_CreateRGBSurface(SDL_SWSURFACE, 200, 200, 32, rmask, gmask, bmask, amask);
if(surface == NULL) {
fprintf(stderr, "CreateRGBSurface ha fallado: %s\n", SDL_GetError());
exit(1);
}
```

En el ejemplo anterior hemos creado una superficie de 200x200 pixeles.

2.2.2. Funciones para mejorar el rendimiento

Durante el desarrollo de un juego tenemos que ir eliminando todo aquello que gasta tiempo y que es prescindible. En esta sección vamos a ver varias funciones que nos ayudan a este trabajo y así mejorar la velocidad del programa.

Una parte importante para todo esto es el "clipping" que viene a ser recortar una parte de pantalla para que sea actualizada, mientras que la parte que quede fuera no se actualice en ese refresco. Con un ejemplo lo vemos mas claro. Imaginarse un juego de naves donde tenemos una parte de la pantalla donde se desarrolla el juego y otra donde colocamos las puntuaciones, vidas y más. Esta parte sólo se actualizará en el momento que subamos las puntuacion, perdamos o ganemos un vida o algo parecido, por lo que se esté actualizando en cada refresco es una pérdida de tiempo. Para ello utilizamos esta función:

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

Esta función debe aparecer antes del SDL_Flip y se le pasa como primer parámetro la pantalla y como segundo el SDL_Rect con el recorte correspondiente.

Otra parte importante es el formato de las imágenes que estemos utilizando. Supongamos que tenemos una pantalla con 24 bit, etc... Esta pantalla tendrá un formato determinado. Imaginemos ahora que queremos pegar (hacer blit) diferentes imágenes en la pantalla que posiblemente tengan otro formato diferente. A la hora de pegar cada una de las imágenes en cada frame, SDL tendrá que hacer la transformación al formato correspondiente. La función que vamos a ver ahora lo que hace es cambiar el formato de la imagen al principio, por lo que se ahorra el cambio de formato al pegar la imagen. La función es la siguiente:

```
SDL_Surface *SDL_ConvertSurface(SDL_Surface *src, SDL_PixelFormat *fmt, Uint32 flags);
```

El primer parámetro es la superficie que queremos convertir, el segundo el formato y el tercero las banderas. La función nos devuelve un puntero a una superficie con el formato correspondiente.

Por último y ya que estamos hablando del rendimiento sobre SDL, no voy a comentar ninguna función, solo comentar un problema que me he encontrado al ir desarrollando. En Windows deberíamos de utilizar la misma profundidad de color en el juego que la que estemos utilizando en la pantalla. Es decir, si utilizamos 16 bpp en la función SDL_SetVideoMode al crear nuestro screen para la pantalla, en las propiedades del sistema operativo deberíamos de tenerlo también, ya que en ordenadores más antiguos puede retardar demasiado el refresco de la pantalla. Seguramente tendrá varias formas de poder solucionarse este problema, como forzar al juego a utilizar los mismo bpp que tiene el sistema operativo, pero actualmente yo no se como es y esta seña es simplemente para que la tengáis en cuenta.

2.2.3. Otras funciones interesantes

SDL_SaveBMP es una función que nos da la posibilidad de crear a partir de una superficie un archivo BMP. Esto nos puede servir para crear ScreenShot del juego en tiempo de ejecución o para utilizarlo en alguna parte del juego. La función tiene la siguiente forma:

int SDL_SaveBMP(SDL_Surface *surface, const char *file);

Como primer parámetro se le pasa un puntero a la superficie y como segundo la ruta donde queremos guardar el archivo. Devuelve 0 en el caso que se cree la imagen y -1 si ha exisitido algún problema.

SDL_SetAlpha es una función que nos da la posibilidad de dar transparencia a una superficie. La función tiene la siguiente estructura:

int SDL_SetAlpha(SDL_Surface *surface, Uint32 flag, Uint8 alpha);

El primer parámetro es la superficie, el segundo los flags, y por último el grado de transparencia un número entre 0 a 255, siendo 255 el valor transparente. Con superficie cargadas desde PNG con zonas transparentes esta función tiene problemas para crear las transparencias.

SDL_SetColorKey esuna función que define un color transparente en la imagen. Al crear los tiles para un juego se define un color que será el que se colocará como transparente. Por ejemplo una nave se le coloca el fondo verde y con esta función se elimina ese color. La estructura de la función es la siguiente:

int SDL_SetColorKey(SDL_Surface *surface, Uint32 flag, Uint32 key);

El primer parámetro es la superficie a la cual queremos aplicar el color transparente, el segundo son los flags, y el tercer es el color. Para ello utilizaremos la función SDL_MapRGB que nos devuelve un valor Uint32 con el valor de un color RGB. A continuación vemos un ejemplo:

```
Uint32 color;
// Escificamos el verde como color para transparente
color = SDL_MapRGB(imagen->format, 0,255,0);
   SDL_SetColorKey(imagen, SDL_SRCCOLORKEY | SDL_RLEACCEL, color);
```