Olvídate de la SDL: Desarrollo de juegos en C++ con Gosu

José Tomás Tocino García - <theom3ga@gmail.com>

¿Cuántas líneas hacen falta para hacer aparecer una imagen en pantalla usando la biblioteca SDL? ¿Cuántas veces has implementado una clase `Imagen` para usar orientación a objetos con la SDL? ¿Has visto algún juego hecho con SDL ir más allá de los 70fps?

Si has usado SDL para programar juegos, conocerás las respuestas a estas preguntas, y habrás sufrido una agonía de bajo nivel repleta de `SDL_Surface`, `SDL_DisplayFormatAlpha` y demás engendros del demonio. Una vez que tenías el motor del juego más o menos hecho, con suerte te quedaban fuerzas para hacer a lo sumo un par de niveles del juego.

Pero esto, señores, **s'acabao**.

Gosu (`http://www.libgosu.org`) es una librería para el desarrollo de videojuegos 2D utilizando Ruby y C++, y disponible para Mac OS X, Windows y, oh sí, **Linux**. Incluye la gestión de gráficos (con **aceleración 3D por OpenGL**), gestión de la entrada y reproducción de música y sonidos, así como funciones de red.

La librería está hecha en **C++**, por lo que la orientación a objetos es total. Además, está completamente adaptada a la **creación de juegos**, adaptando el flujo de ejecución al game loop típico consistente en gestionar la entrada de usuario, actualizar la lógica del juego y finalmente imprimir los gráficos en pantalla.

1. Requisitos previos

Es necesario que sepas C++. Es fácil, es bonito, mola. Si no sabes C++, corre a la biblioteca.

Al ser una librería poco conocida (al menos por ahora, pero esperad a que este tutorial se haga famoso, MWAHAHA), no está disponible en los repositorios de las distribuciones más famosas, ni tampoco está empaquetada, así que hay que instalarla partiendo del código fuente.

Afortunadamente es un proceso muy sencillo que no reviste dificultad alguna. Antes de nada, hay que instalar las dependencias de la librería, que son unas cuantas. Suponiendo que estás en un sistema basado en Debian:

```
$ sudo apt-get install g++ libgl1-mesa-dev libpango1.0-dev libboost1.40-dev libsdl-mixer1.2-dev libsdl-ttf2.0-dev
```

Una vez hecho, crea una carpeta para tu proyecto, por ejemplo `miJuego`, descarga el código fuente de Gosu desde la página de descargas (`http://code.google.com/p/gosu/downloads/list`) y descomprímelo. Debería aparecer una carpeta `gosu` dentro de la carpeta de tu proyecto.

Entra en gosu y luego en el subdirectorio `linux`, y teclea:

```
$ ./configure
$ make
```

Ya está compilada y lista para usarse. Gosu suele adjuntarse con cada proyecto, en lugar de instalarla a nivel de sistema, y se enlaza estáticamente. Aunque de eso hablaremos más adelante.

1.1. Conociendo boost

A lo largo del tutorial, y en todo el código de la propia librería, se hace uso de una parte de las librerías **Boost**. Para los que no lo conozcan, Boost es un conjunto de librerías open source que amplían las posibilidades del lenguaje C++. Después de la STL, se trata del conjunto de librerías **más importante** que existe para este lenguaje; tanto es así que la próxima versión de C++ (C++0x) incluirá por defecto partes de Boost de forma estándar.

Las partes de Boost que se utilizarán son los **punteros inteligentes** (smart pointers). Estos punteros se diferencian principalmente de los punteros tradicionales en que simplifican la gestión de la memoria, mediante tareas como la **liberación automática** de la misma al finalizar su uso. En nuestro caso usaremos concretamente dos tipos:

- **Scoped pointers:** Un scoped pointer (`boost::scoped_ptr`) es un puntero que no puede copiarse, es decir, el objeto al que apunta no puede ser referenciado por ningún otro puntero, asegurándonos de que el único punto de acceso al mismo sea nuestro puntero original. Un scoped pointer pertenece a un ámbito (scope), y es liberado cuando el ámbito termina. Por ejemplo, si es un miembro de una clase, el puntero será liberado cuando se llame al destructor de la misma.
- **Shared pointers:** Un shared pointer (`boost::shared_ptr`) es parecido a un scoped pointer, se diferencia de él en que puede ser copiado, siendo éste su punto fuerte. El shared pointer llevará la cuenta de cuántos punteros están haciendo referencia al objeto creado, liberándose automáticamente cuando ya no quede ningún puntero.

  Esto resulta muy útil a la hora de compartir un recurso entre varios objetos o clases, ya que de esta forma podemos olvidarnos de liberar el recurso cuand ya ningún objeto esté usándolo.

Cuando aparezcan en el tutorial, explicaremos la sintaxis de estos punteros, aunque realmente no se diferencian en nada de los punteros tradicionales, ya que también hacen uso del operador flecha.

2. Empezando por el principio

Bien, parece que quieres hacer un juego. Lo primero es hacer que aparezca una ventana, porque no pretenderás que los muñequitos vayan por la pantalla sin orden ni concierto.

El punto de partida para todas las aplicaciones Gosu es la clase `Gosu::Window`. Se trata de una clase abstracta de la que debemos heredar para empezar a hacer nuestro juego. Mucha palabrería pero en realidad, nada del otro mundo. Abre un nuevo fichero `main.cpp` y guárdalo en la carpeta principal de tu proyecto.

```
1 #include <iostream>
2 #include "Gosu/Gosu.hpp"
```

```
#include <boost/scoped_ptr.hpp>

using namespace std;

class Juego : public Gosu::Window { // HERENCIA

public:
Juego::Juego() : Gosu::Window(800, 600, false, 15){
cout << "Constructor!" << endl;
}
```

Estoy seguro de que muchos de vosotros estáis hartos de que cada vez que vais a las prácticas, os encontréis con los editores del OpenSuSe (u otra distribución) siempre desconfigurados: Que si el indentado está a 8 espacios, que si la letra se ve demasiado difuminada, que si la fuente no es legible, que si el ratón tiene tanta mierda que la bola no rueda... Por eso, para que la vida sea feliz y aprobéis las prácticas de una vez, os traigo la miniguía definitiva del editor **Emacs**!!

Probablemente lo hayáis ejecutado alguna vez y, al intentar cerrarlo, hayáis acabado reiniciando el ordenador. LOL!! Si queréis **salir del Emacs**, tendréis que pulsar `control + x`, levantarlo, y luego pulsar `control + c`. Eso se suele escribir de esta forma: `C-x C-c`.

Una vez que hayáis abierto Emacs, podéis escribir **directamente**, no como el *vi ese raro* que hay que liar una... Inicialmente os aparecerá una página con información sobre Emacs, y si pulsáis cualquier tecla aparecerá un búffer con tres líneas arriba. Este búffer es para tomar notas, si quieres escribir un archivo **nuevo**, pulsa `C-x C-f`. En la parte inferior de la ventana os pondrá `Find file`. Ahí podéis escribir el nombre del nuevo archivo, o introducir la ruta de un archivo que queráis abrir. Sí señores, Emacs utiliza el mismo comando para **abrir un archivo** y para crear uno nuevo. Seguro que el *vi ese raro* no hace lo mismo.

Podéis **moveros por el documento** usando las teclas de cursor, pero si realmente te sientes *friki 100 %*, tendrás que usar los comandos que trae el Emacs para moverte por el documento. Son un poco difíciles de aprender, pero hey, nadie dijo que ser programador fuera fácil. Realmente podéis usar las teclas normales para hacer los movimientos corrientes, a saber: teclas de cursor, ctrl + flechas para moverte por palabras, teclas de inicio y fin, av.pág. y re.pág., ctrl + inicio y ctrl + fin, etcétera. Emacs incluye muchas opciones de movimiento, que podéis encontrar en nuestro amigo Google. Las que os interesaría conocer a la hora de programar son:

- `C-M-a y C-M-e`: La M simboliza la tecla Meta (el alt, vamos). Con `C-M-a` vamos a la anterior función, y con `C-M-e` saltamos a la siguiente
- `M-a`: Pasamos al inicio de la sentencia anterior. Útil si la línea está muy indentada y la tecla de inicio nos deja muy lejos.
- `M-e`: Pasamos al final de la setencia siguiente. Interesante si tenemos sentencias de más de una línea (como las cabeceras de funciones con muchos parámetros).

Una vez hayáis escrito vuestra vida y obra, podéis hacerle un favor a la humanidad y cerrar emacs como dijimos antes. Si, en cambio, queréis **guardar el texto**, tenéis que pulsar `C-x C-s`. Abajo aparecerá el texto `Wrote /home/juanito/archivo`, indicando que todo ha ido bien.

¿Y eso de los búffers qué es? Pff, mejor me vuelvo al Dev-C++... ¡De eso nada! Es sencillo: Al abrir un fichero, Emacs coloca su contenido en uno de sus búffers. Lo que iremos editando será ese búffer, no el fichero. Finalmente cuando queramos guardar las modificaciones, grabaremos el contenido del búffer de nuevo al fichero.

En Emacs, editamos los búffers en **ventanas**. Inicialmente en emacs sólo veremos una ventana que ocupa todo el área, pero podemos tener tantas como queramos. Prueba a coger y pulsar `C-x 2`. Tachán! Se **divide horizontalmente** la ventana en dos ventanas con el mismo contenido. Ahora pulsa `C-x 3`. Como verás, se divide verticalmente la ventana en la que estabas. Si quieres ir saltando de ventana en ventana, pulsa `C-x o`. Así, podrías tener en una ventana el archivo `Persona.h`, en otra `Persona.cpp` y en otra más grande `main.cpp`, e irlos editando sin tener que mover las manos del teclado.

Puede que al rato te canses de tener tantas ventanas y sólo quieras tener una sóla. Para ello, pulsa `C-x 0`. Ahora tendrás **una sóla ventana**, con un búffer abierto en ella, pero a la vez tendrás otros búffers abiertos pero escondidos, con el contenido de los otros archivos. Puedes **cambiar de búffer** pulsando `C-x b`. Abajo te aparecerá un mensaje: `Switch to buffer:`. Tendrás que indicar el nombre del búffer al que quieres cambiar. Puedes pulsar en ese momento la tecla `Tabulador` para ver

la lista de búffers abiertos. Emacs es como el shell bash, puedes empezar a escribir el nombre del búffer y pulsar `tabulador`, y Emacs intentará autocompletar el nombre para que no tengas que destrozarte los dedos.

Por último, si quieres **cerrar** un búffer, pulsa `C-x k`. Te pedirá confirmación, pulsa `intro` para mandar al búffer a mejor vida.

Maldito friki, todavía no me has convencido para que use Emacs en lugar de Microsoft Word. Bien, habrá que empezar a sacar la artillería pesada.

Copiando, buscando y compilando

Suponte que quieres tienes dos ventanas, en una tu práctica de POO que tienes que mandar esta noche, aún sin acabar. En la otra ventana, tienes la práctica de cualquier otro cateto al que hayas conseguido birlársela. Para **copiar un texto** y pegarlo en otro lado, primero deberás **marcarlo**. Coloca el cursor al inicio del bloque de texto a copiar y pulsa `C-Espacio`. Verás que abajo aparece el mensaje `Mark set`. Ahora coloca el cursor al final del bloque. Si quieres copiar el texto, pulsa `M-w`, y si quieres **cortarlo o eliminarlo** pulsa `C-w`.

Ahora puedes saltar a la otra ventana (recuerda, usando `C-x o`) y **pegar el bloque** de texto donde menos se note que no lo has hecho tú. Coloca el cursor donde quieras y pulsa `C-y`. Voilá! Ya tienes la práctica lista para mandar.

Aich. Parece que el tío al que le has copiado la práctica ha puesto su nombre en varios sitios del código. Puedes hacer una **búsqueda** pulsando `C-s` y escribiendo el término a buscar. Mientras vas escribiendo, se irán resaltando las coincidencias. Puedes ir a la **siguiente** coincidencia pulsando de nuevo `C-s`, o ir a la **anterior** con `C-r`. Puedes terminar la búsqueda con `C-g`. Este comando, además, **termina** cualquier otro comando que estés utilizando, así que si te **entra el pánico** en cualquier momento, pulsa repetidamente `C-g`.

Emacs no sólo puede buscar, también puede **reemplazar**, porque Emacs es *awesome*. Para reemplazar, pulsa `M-%` (esto es, alt + shift + 5). Emacs te preguntará por la cadena a buscar, y al pulsar intro, te preguntará por la cadena de reemplazo. Seguidamente irá recorriendo todas las ocurrencias de la cadena original, preguntándote si quieres reemplazar. Simplemente pulsa "y" o "n". Si quieres salir del modo de reemplazo, usa `C-g`.

Bueno, ya tienes el código preparado. No queda ni rastro del autor original del mismo, seguro que Paco ni se da cuenta. Lo que queda es **compilarlo**. Como Emacs es *awesome*, puedes compilar directamente desde el editor. Para ello, tendrás que teclear un comando escrito. En Emacs, para **introducir un comando**, sea cual sea, tienes que pulsar `M-x`. Seguidamente, podrás escribir el comando. En nuestro caso, tendrías que introducir el comando `compile`, y pulsar intro. Emacs te pedirá que confirmes el comando que usará para compilar, por defecto `make -k`. Si no tienes un `makefile`, podrás cambiar el comando a ejecutar a algo como `g++ -o programa main.cpp`.

Tras pulsar intro, se abrirá una ventana de Emacs y en ella aparecerá la información de compilación. Si hay errores, aparecerán resaltados. Además, si haces click sobre ellos podrás navegar directamente al archivo y línea en la que se produjo el error. Si al final no hubo error, seguro que estás deseando **ejecutar el programa** a ver qué tal va.

Hay dos formas de hacerlo. Por un lado, tienes la posibilidad de tener en una de las ventanas una **shell completa** en la que introducir comandos. Esto lo puedes hacer pulsando `M-x` y escribiendo `shell`. Si lo que realmente te interesa es **ejecutar un sólo comando**, también puedes hacerlo pulsando `M-!` (es decir, alt + shift + 1) e introduciendo el comando, por ejemplo `./programa`. En el minibúffer aparecerán los resultados del ejecutable.

Indentando y comentando

Esta parte es realmente interesante. Emacs es capaz de **indentar el código automáticamente**. Puedes tener el código hecho unos zorros, que Emacs te lo pone fino y seguro en un plisplás. Hay muchas formas de hacerlo. La más directa es, simplemente, ir escribiendo. A medida que vas escribiendo, Emacs va indentando el código que escribes. De todas formas, puede que la hayas pifiado en algún sitio y quieres que Emacs te lo indente. Lo único que tienes que hacer es ir a esa línea (no hace falta que sea al inicio de la misma) y pulsar `tabulador`. Te lo indentará correctamente respecto a las líneas anteriores.

Otra forma sería ir indentar todo el contenido de un **bloque**, una función por ejemplo. Colocamos el cursor en el brazo de apertura {, y pulsamos `C-M-q`. El código dentro del bloque quedará indentado. Por último, podemos indentar un

bloque **seleccionado manualmente**: Nos vamos al inicio, pulsamos `C-Espacio`, nos vamos al final y metemos el comando `indent-region` (para meter el comando recuerda que es con `M-x`).

Por último, podemos hacer algo parecido a la indentación pero para **comentar las líneas**. Selecciona un bloque con `C-Espacio`, coloca el cursor al final del mismo y pulsa `C-c C-c`. El código quedará comentado. Para **descomentarlo**, puedes usar el comando `uncomment-region` (con poner `unc` y pulsar `tabulador` ya aparece).

Bueno, por ahora eso es todo. Emacs es **mucho más** que esta μ-guía, pero con lo de aquí puedes tirar *pa'lante* más o menos. Siempre puedes mirar la ayuda del modo pulsando `C-h m`, o buscar en San Google. ¡Espero que sirva! Un saludo.