Desarrollo de videojuegos para PSP con C++ y SDL

David Saltares Márquez

Este documento posee una licencia Creative Commons (by-sa)

Índice

1. Introducción

Me he dispuesto a escribir este tutorial tras haber estado trabajando en el port a PSP de Granny's Bloodbath y encontrar algunas dificultades tanto en la instalación del Kit como en la escritura de un proyecto básico que funcione. Por internet circulan algunos tutoriales similares pero todos cuentan con algún otro error y, o yo soy muy torpe, o se hace bastante complicada la instalación.

Por otro lado, la mayoría se centraban en sistemas Windows y yo quería hacer la instalación sobre Ubuntu.

Espero que esta pequeña guía me sirva a mí como recordatorio para una próxima vez y, si alguien se ve beneficiado pues muchísimo mejor.

2. Instalación

Lo primero que debemos hacer es instalar las dependencias del kit de desarrollo de PSP. Para ello abriremos una terminal e introduciremos el siguiente comando:

```
sudo apt-get  install build-essential autoconf automake bison flex \
        libncurses5-dev libreadline-dev libusb-dev texinfo libmpfr-dev \
        libgmp3-dev libtool
```

Ahora debemos establecer algunas variables de entorno para que el sistema sepa dónde encontrar las nuevas librerías de PSP a la hora de compilar. Editamos el fichero */.bashrc* y añadimos al final las siguientes líneas:

```
export PSPDEV="/usr/local/pspdev"
export PSPSDK="$PSPDEV/psp/sdk"
export PATH="$PATH:$PSPDEV/bin:$PSPSDK/bin"
```

Cuando reiniciemos el fichero */.bashrc* volverá a cargarse pero no es necesario hacerlo, podemos ejecutar el siguiente comando:

```
source ~/.bashrc
```

El siguiente paso es descargarnos una copia del directorio *trunk* del repositorio de ps2dev, el cual contiene todo lo que necesitamos (y más). El repositorio tiene un tamaño considerable y, dependiendo de cómo ande el servidor, puede tardar bastante.

```
svn co svn://svn.ps2dev.org/psp/trunk/ pspsdk
```

Bueno, si habéis tenido la paciencia suficiente de llegar hasta aquí vamos por buen camino. Ahora toca instalar el *toolchain*, el kit básico:

```
cd pspsdk

cd toolchain

sudo ./toolchain-sudo.sh
```

Existe un pack de bibliotecas adicionales entre las que se encuentran las SDL llamado *psplibraries*. Este pack contiene: bzip2, freetype, jpeg, libbulletml, libmad, libmikmod, libogg, libpng, libpspvram, libTremor, libvorbis, lua, pspgl, pspirkeyb, SDL, SDL_gfx, SDL_image, SDL_mixer, SDL_ttf, smpeg-psp, sqlite, zlib y zziplib. Muchas son dependencias de las SDL pero algunas como sqlite (bases de datos), lua (lenguaje de scripting) o pspgl (versión de Open GL para PSP) no tienen nada que ver aunque son muy interesantes también. Lo instalamos de la siguiente manera:

```
cd ..

cd psplibraries

sudo ./libraries-sudo.sh
```

En teoría ya deberíamos estar listos para crear nuestros proyectos en C++ que usen las SDL para PSP, ¡pero no es así! Debe haber algún error en el script anterior porque SDL_mixer no se instala como debería. Hemos de compilar e instalar sus dependencias manualmente. Comenzamos cambiando el propietario de la carpeta donde se instala el kit de desarrollo, sino las librerias no pueden instalarse (al menos yo no he conseguido hacerlo):

```
sudo chown -R username:group /usr/local/pspdev
sudo chown username:group /usr/local/pspdev/*
```

Donde *group* y *username* son los nombres de nuestro grupo y usuario en el sistema.

Nos dirigimos a instalar libTremor manualmente, dependencia de SDL_mixer:

```
cd ..

cd libTremor

LDFLAGS="-L$(psp-config --pspsdk-path)/lib" LIBS="-lc -lpspuser" ./autogen.sh \
     --host psp --prefix=$(psp-config --psp-prefix)

make clean

make

make install
```

Finalmente le toca el turno a SDL_mixer y toca seguir el siguiente proceso:

```
cd ..
```

```
cd SDL_mixer

./autogen.sh

LDFLAGS="-L$(psp-config --pspsdk-path)/lib" LIBS="-lc -lpspuser" \
            ./configure --host psp --with-sdl-prefix=$(psp-config --psp-prefix) \
            --disable-music-mp3 --prefix=$(psp-config --psp-prefix) \
            --disable-music-libmikmod --enable-music-mod

make clean

make

make install
```

Con eso debería bastar, siento el rodeo, si alguien encuentra una manera mejor de instalar el kit de desarrollo para PSP con SDL en GNU/Linux, por favor, que me lo comunique. Entre todos podemos hacer una gran guía.

3. El makefile

El makefile para compilar proyectos para PSP tiene ciertas particularidades dignas de mención en esta humilde guía.

Vamos a suponer que tenemos la siguiente jerarquía de directorios:

```
|- Proyecto
    |- makefile
    |- main.cpp
    |- engine
       |- ficheros.cpp
       |- ficheros.h
```

Nuestro makefile sería algo similar a lo siguiente:

```
#Proyecto
TARGET = Nombre del proyecto
SDL_CONFIG = $(PSPBIN)/sdl-config

# Rutas
MOTOR_DIR := engine

# Ficheros fuente del juego
SRC := main.cpp

# Ficheros fuente del motor.
SRC_MOTOR := ficheros.o del motor

# motor_dir + fuentes
SRC_DIR_MOTOR := $(foreach src, $(SRC_MOTOR),$(MOTOR_DIR)/$(src) )

# Objetos
OBJS := $(SRC:%.cpp=%.o) $(SRC_DIR_MOTOR:%.cpp=%.o)

INCDIR =
CFLAGS = $(shell $(SDL_CONFIG) --cflags) -G0 -Wall -O2 -DPSP
```

```
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti -D"TIXML_USE_STL"
ASFLAGS = $(CFLAGS)

LIBDIR =
LDFLAGS =
LIBS = -lstdc++ -lsupc++ -lSDL_gfx -lSDL_image -lSDL_mixer -lSDL_ttf -lfreetype \
        -lpng -ljpeg -lvorbisidec -lz -lm $(shell $(SDL_CONFIG) --libs)


EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = Nombre del proyecto
PSP_EBOOT_ICON= "icono.png"
PSP_EBOOT_PIC1= "fondo.png"
PSP_EBOOT_SND0= "sonido.at3"


PSPSDK = $(shell psp-config --pspsdk-path)
PSPBIN = $(shell psp-config --psp-prefix)/bin
USE_PSPSDK_LIBC=1
include $(PSPSDK)/lib/build.mak
```

Podemos personalizar la apariencia de nuestro *homebrew* en el menú de PSP mediante las siguientes variables:

- **PSP_EBOOT_ICON**: icono de 144x80 que identificara al juego en la sección *Juegos* del menú.
- **PSP_EBOOT_PIC1**: fondo que aparecerá en la consola cuando el juego esté seleccionado, debe tener 480x272.
- **PSP_EBOOT_SND0**: fichero de sonido en formato at3 que se escuchará cuando nuestro juego esté seleccionado en el menú.

4. Callbacks de salida al SO

Seguramente sabréis de qué hablo pero es probable que no os hayáis dado cuenta de que debe ser algo a controlar a la hora de portar la aplicación a PSP. Me refiero a la pausa del juego y a la vuelta al sistema operativo de la consola. En cualquier momento podemos pulsar el botón *HOME*, seleccionar 'salir del juego' y volver al menú principal. Si no tenemos cuidado provocaremos un cuelgue en la consola cada vez que queramos salir.

Tendremos que definir ciertas funciones que actúen como *callbacks*. Un fichero *main.cpp* de un port génerico podría ser el siguiente:

```
1  #include <pspkernel.h>
2  #include "engine/application.h"
3  #include "engine/keyboard.h"
4
5  int exit_callback(int arg1, int arg2, void *common) {
6      /* Codigo para cerrar correctamente nuestro sistema */
7      keyboard->set_quit();
8      return 0;
9  }
10
11 int CallbackThread(SceSize args, void *argp) {
```

```
12      int cbid;
13      cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
14      sceKernelRegisterExitCallback(cbid);
15      sceKernelSleepThreadCB();
16      return 0;
17  }
18
19  int SetupCallbacks(void) {
20      int thid = 0;
21      thid = sceKernelCreateThread("update_thread", CallbackThread,
22          0x11, 0xFA0, 0, 0);
23      if(thid >= 0) {
24          sceKernelStartThread(thid, 0, 0);
25      }
26      return thid;
27  }
28
29  /* Necesario para no tener problemas con C++ */
30  extern "C" int SDL_main (int argc, char* args[]);
31
32  int main(int argc, char *argv[])
33  {
34      /* Preparamos los callbacks */
35      SetupCallbacks();
36
37      /* Lanzamos nuestro sistema */
38      Application app("XML/configuration.xml");
39
40      app.run();
41
42      /* Volvemos al sistema operativo */
43      sceKernelExitGame();
44
45      return 0;
46  }
```

Es importante comprender que cuando el usuario decide salir del juego (esperemos que no sea demasiado a menudo) se llama a la función *exit_callback* por lo que debe ser ella la que se encargue de hacer los cambios pertinentes para romper el game-loop. En mi caso una forma podía ser llamar a la al método *set_quit*() de mi clase *keyboard* pero en el vuestro puede ser cualquier otro.

5. La pantalla

Las dimensiones de la pantalla de la PSP son de 480x272 píxeles por lo que la superficie principal de SDL a crear debe ser exactamente de dichas dimensiones. Por otro lado está la profundidad de color, inicialmente pensé que sería más eficiente bajar dichos niveles de 32bpp a 16bpp o incluso 8bpp si fuera necesario por cuestiones de rendimiento. No obstante, el port de SDL para PSP parece no funcionar correctamente en los dos últimos modos. Finalmente acabé usando 32bpp aunque el rendimiento final es bastante bueno.

La llamada para inicializar SDL en PSP puede ser algo similar a esto:

```
1  SDL_Surface *screen = SDL_SetVideoMode(480, 272, 32, SDL_HWSURFACE | SDL_DOUBLEBUF);
```

Con las correspondientes comprobaciones posteriores.

6. Leyendo la entrada de PSP

Teóricamente, como la mayoría de tutoriales apuntan, podemos usar *SDL_Joystick* para manejar la entrada del jugador. Se nos presenta una equivalencia entre botones de joystick y de PSP para que podamos manejarla pero lo cierto es que, o soy tremendamente torpe, o este sistema no funciona del todo bien.

Finalmente, para *Granny'sbloodbath* decidí cambiar la implementación de la clase que manejaba la entrada del jugador (utilizando la API de las librerías de PSP) sin cambiar su interfaz, de ese modo me adaptaba a las circunstancias de PSP sin afectar al resto del sistema.

Las clases que utilicé son las siguientes y funcionan bastante bien:

keyboard.h:

```
1  /*
2      This file is part of Granny's Bloodbath.
3
4      Granny's Bloodbath is free software: you can redistribute it and/or modify
5      it under the terms of the GNU General Public License as published by
6      the Free Software Foundation, either version 3 of the License, or
7      (at your option) any later version.
8
9      Granny's Bloodbath is distributed in the hope that it will be useful,
10     but WITHOUT ANY WARRANTY; without even the implied warranty of
11     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
12     GNU General Public License for more details.
13
14     You should have received a copy of the GNU General Public License
15     along with Granny's Bloodbath.  If not, see <http://www.gnu.org/licenses/>.
16 */
17
18 #ifndef _KEYBOARD_
19 #define _KEYBOARD_
20
21 #include <pspkernel.h>
22 #include <pspctrl.h>
23 #include <valarray>
24 #include <map>
25
26 //! Gestiona el teclado, consultamos teclas pulsadas
27
28 /**
29     @author David Saltares Marquez
30     @version 1.0
31
32     Clase que sigue el patron de diseno Singleton (una sola instancia accesible
33     desde todo el sistema).
34     Lleva el control de que teclas estan pulsadas en un momento determinado,
35     cuales se sueltan y cuales se vuelven a presionar.
36
37     Ejemplo de uso
38
39     \code
40         // Game loop
41         while(!quit){
42             keyboard->update() // Equivalente a Keyboard::get_instance()->update();
43
44             if(keyboard->pressed(Keyboard::KEY_UP)) // Si pulsamos arriba
```

```
45                  ...
46              if(keyboard->released(Keyboard::KEY_HIT)) // Si dejamos de pulsar golpear
47                  ...
48              if(keyboard->newpressed(Keyboard::KEY_SHOOT)) // Si antes soltamos y
49                                      // ahora pulsamos disparar
50          }
51      \endcode
52  */
53  class Keyboard{
54      public:
55
56          /**
57              Controles del teclado utilizados en el juego
58          */
59          enum keys{
60              KEY_UP,
61              KEY_DOWN,
62              KEY_RIGHT,
63              KEY_LEFT,
64              KEY_SHOOT,
65              KEY_HIT,
66              KEY_KNEEL,
67              KEY_EXIT,
68              KEY_ENTER
69          };
70
71          /**
72              @return Si es la primera vez que se usa Keyboard crea su instancia y
73              la devuelve. Sino simplemente la devuelve.
74          */
75          static Keyboard* get_instance(){
76              /* Si es la primera vez que necesitamos Keyboard, lo creamos */
77              if(_instance == 0)
78                  _instance = new Keyboard;
79              return _instance;
80          }
81
82          /**
83              Actualiza el estado del teclado. Debe llamarse una vez al comienzo
84              del game loop.
85          */
86          void update();
87
88          /**
89              @param key Tecla a consultar
90
91              @return true si la tecla esta pulsada, false en caso contrario.
92          */
93          bool pressed(keys key);
94
95          /**
96              @param key Tecla a consultar
97
98              @return true si la tecla estaba antes pulsada en la ultima actualizacion
99              y ahora no lo esta, false en caso contrario.
100         */
101         bool released(keys key);
102
103         /**
104             @param key Tecla a consultar
105
106             @return true si la tecla no estaba pulsada en la ultima actualizacion y
107             ahora lo esta, false en caso contrario.
108         */
109         bool newpressed(keys key);
110
111         /**
112             @return true si se ha producido algun evento de salida, false en caso
```

```
113                 contrario
114             */
115             bool quit();
116
117             void set_quit() {_quit = true;}
118
119         protected:
120             Keyboard();
121             ~Keyboard();
122             Keyboard(const Keyboard& k);
123             Keyboard& operator = (const Keyboard& k);
124         private:
125             static Keyboard* _instance;
126             std::valarray<bool> actual_keyboard;
127             std::valarray<bool> old_keyboard;
128             bool _quit;
129             std::map<keys, PspCtrlButtons> configured_keys;
130             SceCtrlData buttonInput;
131             int n_keys;
132     };
133
134     #define keyboard Keyboard::get_instance()
135
136     #endif
```

keyboard.cpp:

```
1   #include <iostream>
2   #include <algorithm>
3   #include "keyboard.h"
4
5   using namespace std;
6
7   Keyboard* Keyboard::_instance = 0;
8
9   Keyboard::Keyboard()
10  {
11      /* Configuracion predeterminada */
12      configured_keys[KEY_UP] = PSP_CTRL_UP;
13      configured_keys[KEY_DOWN] = PSP_CTRL_DOWN;
14      configured_keys[KEY_LEFT] = PSP_CTRL_LEFT;
15      configured_keys[KEY_RIGHT] = PSP_CTRL_RIGHT;
16      configured_keys[KEY_HIT] = PSP_CTRL_CROSS;
17      configured_keys[KEY_SHOOT] = PSP_CTRL_CIRCLE;
18      configured_keys[KEY_EXIT] = PSP_CTRL_SELECT;
19      configured_keys[KEY_ENTER] = PSP_CTRL_START;
20
21      /* Inicializamos el estado del teclado */
22      n_keys = configured_keys.size();
23      actual_keyboard.resize(n_keys);
24      old_keyboard.resize(n_keys);
25
26      for(int i = 0; i < n_keys; ++i){
27          actual_keyboard[false];
28          old_keyboard[false];
29      }
30
31      _quit = false;
32
33      sceCtrlSetSamplingCycle(0);
34      sceCtrlSetSamplingMode(PSP_CTRL_MODE_ANALOG);
35  }
36
37  Keyboard::~Keyboard()
38  {
39  }
```

```
40
41
42  void Keyboard::update()
43  {
44      /* Ahora el teclado nuevo es el viejo */
45      old_keyboard = actual_keyboard;
46
47      /* Actualizamos el estado de la entrada PSP */
48      sceCtrlPeekBufferPositive(&buttonInput, 1);
49
50      /* Actualizamos el estado del teclado */
51      for(map<keys, PspCtrlButtons>::iterator i = configured_keys.begin();
52          i != configured_keys.end(); ++i)
53              actual_keyboard[i->first] = (buttonInput.Buttons & i->second)?
54                                  actual_keyboard[i->first] = true :
55                                      actual_keyboard[i->first] = false;
56  }
57
58  bool Keyboard::pressed(keys key)
59  {
60      /* Devolvemos si la tecla indicada esta pulsada */
61      return actual_keyboard[key];
62  }
63
64  bool Keyboard::released(keys key)
65  {
66      /* Comprobamos si la tecla indicada no esta pulsada y antes si */
67      return (!actual_keyboard[key] && old_keyboard[key]);
68  }
69
70  bool Keyboard::newpressed(keys key)
71  {
72      /* Comprobamos si la tecla indicada esta pulsada y antes no */
73      return (actual_keyboard[key] && !old_keyboard[key]);
74  }
75
76  bool Keyboard::quit()
77  {
78      return _quit;
79  }
```

7. Conclusiones

Con todos los detalles anteriormente mencionados creo que es suficiente para poder comenzar a portar los juegos que se creen en PC a PSP (siempre que la potencia, control y detalles similares lo permitan). Si alguien encuentra algún problema con este tutorial o tiene alguna sugerencia, por favor, les animo a enviar un mensaje en el blog de *Granny'sBloodbath'*:

http://grannysbloodbath.wordpress.com