Un diccionario para la PSP

Un diccionario; en XML, que permita importar nuevos ficheros a través de la conexión WiFi o del cable USB.

Katharina Maria Kaczmarczyk
katkac@ei.upv.es
Ingeniería Técnica Informática de Gestión (ITIG)

Director:

Manuel Agustí Melchor
magusti@disca.upv.es
Grup de Visió per Computaor (VxC)
Departament d'Informàtica de Sistemes i Computadors (DISCA)
Escola Tècnica Superior d'Enginyeria Informàtica

Código P.F.C: Disca-144

Valencia, 2011


ictionary

Definición del estudio

Breve resumen

Este estudio ilustra el desarrollo de una aplicación multimedia realizada en la famosa videoconsola portátil de Sony (PlayStation Portable) en el lenguaje de programación de C++. En concreto, una aplicación "Minis" con un tamaño máximo de 100Mbyte. Otro aspecto del estudio consiste en las mejoras de la interfaz. En la versión alfa, había el problema de velocidad. Quiere decir, cuando se querría escribir una palabra con uno de los dos teclados empleados, la aplicación necesitaba mucho tiempo para dibujar el carácter teclado. Por ese motivo, fue necesario cambiar completamente la arquitectura del programa. La versión alfa del diccionario también debería ser un traductor. Debido al resultado de los análisis de los diccionarios electrónicos, sería necesario un teclado "inteligente" para poder ver directamente las sugerencias y/o facilitar el uso de la aplicación al usuario. Por otra parte, analizar la funcionalidad de SDL, especialmente en la parte de reproducir imágenes y/o textos por pantalla para evitar un mal uso.

Palabras clave: Sony PlayStation Portable (PSP), diccionario, lenguaje de programación de C++, SDL, Minis

Abstract

This study illustrates the development of a multimedia application on Sony's popular handheld (PlayStation Portable) in the programming language C ++. In particular a "Minis" application with the maximal size of 100MByte .Another aspect of the project is the bettering of the interfaces. In the alpha version of the dictionary, there was a problem with the velocity. Which means, when you wanted to write a word with one of the two implemented keyboards, the application needed a lot of time to draw the written character, because of this, it was necessary to change completely the program architecture. The alpha version of the dictionary also should be a translator. Dude the result of the analyses of the electronic dictionaries, consist the need of an "intelligent" keyboard in order to see directly the suggestions and/or facilitate the use of the application to the user. On the other hand, analyze the functionality of SDL, especially to reproduce the images and / or text on the screen.

Key words: Sony PlayStation Portable (PSP), dictionary, programming language C++, SDL, Minis

Agradecimientos

En primer lugar, me gustaría dar las gracias a todo el personal de la universidad politécnica de Valencia para la formación impartida en los últimos años. En particular me gustaría agradecer a mi director del proyecto final de carrera Manuel Agustí Melchor, su paciencia e inspiración en el último año y en los últimos meses.

Mi mayor agradecimiento a mis mejores amigos Tobias Heinig, Patrick Glaser y Nuria Martinez Soriano por su apoyo constante durante este tiempo. Mi agradecimiento a la profesora María Lorenzo Hernández de la asignatura "fundamentos de animación 1" de bellas artes, por su ayuda con el diseño del personaje y las animaciones. Asimismo, me gustaría agradecer a Sabrina Takeuchi por su apoyo de colorear las imágenes del búho para los menús.

Por último, me gustaría dar las gracias al usuario "mowglisanu"[1], que me ha ayudado mucho en el tema de programación en la PSP. En particular con el problema de "Hola mundo"[2], "SDL música2"[3], "USB"[4] y "WiFi" y me gustaría dar las gracias a mi familia y amigos por su aliento y apoyo durante los últimos meses.

[1] Perfil del usuario "mowglisanu" en el foro de PSP-Programming: http://www.psp-programming.com/forums/index.php?action=profile;u=1138

[2] Enlace del problema de "Hola mundo": http://www.psp-programming.com/forums/index.php/topic,5048.0.html

[3] Enlace del problema de "SDL audio2": http://www.psp-programming.com/forums/index.php/topic,5056.0.html

[4] Enlace del problema de "USB" y "WiFi": http://www.psp-programming.com/forums/index.php/topic,5083.0.html

Índice

Lista de figuras

Lista de tablas

Lista de abreviaturas

AA	Aplicaciones de aprendizaje
AM	Aplicación Multimedia
CB	Code::Blocks
CFW	Costume Firmware
GUI	Graphical user interface
IDE	Integrated development environment
	Integrated design environment
	Integrated debugging environment
IMD	Integración de medios digitales
NDS	Nintendo DS
N3DS	Nindento 3DS
OFW	Original Firmware
OOP	Object-Oriented Programming
OSK	On-Screen-Keyboard
OS	Open Source
PSN	PlayStation Network
PSP	Sony PlayStation Potable
SCEE	Sony Computer Entertainment Europe
SDK	Software development kit
SDL	Simple DirectMedia Layer
SO	Sistema operativo
UPV	Universidad Politécnica de Valencia
VS	Visual Studio
WiFi	Wireless Fidelity

XMB	Xross Media Bar
XML	eXtensible Markup Language
XP	eXtreme Programming

Prólogo

En la asignatura IMD decidí realizar un trabajo en vez de hacer las prácticas. Tuve la posibilidad de ver el desarrollo de una aplicación multimedia y ganar experiencia. A me interesaba muchísimo crear un programa para la PSP. Entonces decidí desarrollar la versión alfa del diccionario. En mitad de octubre del año 2010 empecé a planificar, diseñar y programar. Aunque tengo un poca experiencia en la programación en C++, me costaba mucho tiempo entender y revisar todo, porque hace mucho tiempo que programaba en C (C, C++ o C#), y normalmente programo en Java. Todos los días me esforzaba para terminar el proyecto antes del día 22 de diciembre de 2010. Este día tenía que exponer mi trabajo (la versión alfa). Después de la exposición recibí muchos comentarios positivos del profesor, de mis compañeros de clase, de mi familia y de mis amigos, por lo que he decido finalizar el diccionario.

En mi opinión, es muy importante tener un diccionario electrónico en todo el momento, especialmente para la gente que viaja mucho por motivos de trabajo, también para las familias que quieren pasar sus vacaciones fuera de su país o por último para los estudiantes de ERASMUS.

1. Información general

Mientras tanto existe un gran número de videoconsolas en el mercado y la demanda aumenta mucho. Asimismo, sube la demanda de aplicaciones de aprendizaje o juegos interactivos para la Nintendo DS. Con la ayuda de la comunidad de programación de PSP, es posible escribir sus propios programas debido a su interés o gusto. La mayoría de los dispositivos portátiles ofrecen diccionarios. Entonces la pregunta es, porqué la PSP no ofrece ningún diccionario aunque es más potente que la NDS.

En este documente, en el capítulo cuatro se ilustra los pasos necesarios para poder programar en la PSP, el problema del IDE (Code::Blocks), los cambios entre un código dado en C a un código en C++ y por última se muestra unas pruebas de programación en C++ en la PSP cómo se puede realizar una animación o un menú. El siguiente capítulo, capítulo cinco enseña todos los pasos del diseño del diccionario y un breve resumen de la programación. Especialmente un resumen de cómo se puede programar un teclado en general para su aplicación. El capítulo seis trata de la programación, en concreto sobre la arquitectura del diccionario, el cual es el núcleo interno de cada programa. El último capítulo importante - capítulo siete – muestra todas las funcionalidades del diccionario y enseña unas ideas para el trabajo futuro en el diccionario como por ejemplo disponer de mini juegos para mejorar el aprendizaje de vocabulario.

Aparte de este documento en el apéndice se ilustra los análisis necesarios para el desarrollo del diccionario como en concreto el diseño. Además muestra unas informaciones adicionales como por ejemplo, cómo sacar el OSK original del SDK de la PSP.

2. Objetivo

El objetivo principal es, exponer el desarrollo de una aplicación multimedia de un tamaño máximo de 100Mbyte. En este caso concreto, desarrollar una aplicación de tipo "Minis" [1] de PSP. "Minis", es el nombre de los juegos descargables de la tienda online de Sony para PSP. Los "Minis" pueden ser juegos sencillos como por ejemplo "Tetris". Pueden ser en 2d, o también en 3d. El único criterio de "Minis" es, que el juego o la aplicación no superan el tamaño máximo de 100Mbyte. Todas las generaciones de la PSP pueden ejecutar los "Minis". Especialmente interesante son los "Minis" para la PSP GO, que no dispone de un lector de discos ópticos de UMD. Antes de empezar con el desarrollo de una aplicación "Minis", es importante comparar todas las videoconsolas portátiles, las que hay en el mercado. En concreto, comparar las especificaciones técnicas de la NDS y de la PSP. No solo de las generaciones actuales, sino también de sus sucesoras. Nintendo ha publicado la N3DS este año. Sony ha anunciado la publicación de una nueva generación de PSP (E-1000) y también su sucesora la PSP Vita. A parte de las videoconsolas, también es necesario analizar qué tipos de aplicaciones existen, en concreto de aprendizaje o un diccionario.

Otro objetivo consiste en aprender los significados de los términos, los cuales se usan en la comunidad de programación de PSP en los foros por internet [3]. Por ejemplo saber lo que es un "Homebrew" [6]. Para desarrollar un "Homebrew" o en este caso especifico un "Minis", es necesario instalar un SDK. Depende de si se quiere programar con un IDE o no. En el caso que sí, es necesario elegir un IDE de su gusto y modificarlo, para que este IDE se comporte bien con el SDK. La versión alfa se ha programado con CB en C con las librerías SDL. El trabajo de Anush Euredjian Chiappe y Juan Cortés Maiques del año 2009[2] fue programado con CB en C con las librerías SDL. En este proyecto, se ha programado en C++ también con las librerías SDL. Debido a tener un problema con el compilador y el enlazador de CB en el caso C++, fue preciso tomar una decisión – programar con IDE o sin IDE. Un IDE facilita la programación un poco para el programador. La mayoría de usuarios en los foros de programación de PSP usan VS como IDE para programar en C y/o C++. Además, la modificación de VS es más sencilla que de CB. Antes de cambiar el IDE, el proyecto fue programando con notepad++ y compilado con el propio símbolo del sistema de Windows 7.

¿Por qué en C++? - Por sus ventajas, por ejemplo la creación de clases y objetos, herencia, polimorfismo y usar "templates" (p.ej. "map"[5] es templete de diccionario en C++). C++ es una lenguaje de OOP, por este motivo, es recomendable usar la técnicas

[5] C++ Map template. http://en.wikipedia.org/wiki/Map_(C%2B%2B),

de divide y vencerás, sobrecarga de métodos para reducir el problema, su coste y el número total de líneas de código. Sirve mucho en el tema del desarrollo de proyectos grandes en un equipo de varios programadores, p.ej. un videojuego para una videoconsola.

¿Por qué SDL? – SDL es muy famoso para la programación de videojuegos simples para el ordenador en C y C++. Por internet hay muchos blogs, tutoriales y wikis sobre el tema SDL. El SDK MinPSPw dispone de las librerías SDL para la PSP. SDL es fácil de aprender por su estructura. Por estos motivos, el proyecto fue programado con VS en C++ con las librerías SDL.

Aparte de comparación de videoconsolas y programación, el objetivo consiste también en analizar los diccionarios electrónicos para saber que componentes son indispensables para el desarrollo de uno. En concreto, para el diseño de un diccionario para un dispositivo portable y/o videoconsolas portátiles (PSP, NDS). Como comentado antes, el objetivo general es crear un "Minis", esto indica la necesidad de mostrar todos los pasos de desarrollo de una GUI con SDL. Asimismo, cómo analizar, crear y especificar una GUI para una aplicación. En este caso concreto, el desarrollo del teclado "inteligente". Quiere decir, un teclado que dispone directamente un par de sugerencias a la hora de escribir la palabra. Debido al resultado del análisis de AA para las videoconsolas portátiles (en particular la NDS) y el objetivo general, el diseño de AM debería ser sencillo, para que los niños puedan usar el diccionario sin dificultad.

3. Introducción

Antes de todo, es importante decir que existían unos problemas de comunicación con el negocio de SCEE. Varias direcciones de correos resultaron estar en desuso, el número de fax de SCEE tampoco funcionaba, porque a partir de 30 segundos de transmisión del fax siempre ocurría un corte de señal. Se supone que en el negocio de Sony hay unas restricciones para evitar recibir "Junk Mails". Por estos motivos, no fue posible conseguir el SDK original, ni los documentos correspondientes de él. Asimismo, no fue posible ver las diferencias entre ambos SDK (oficial, no oficial).

La dirección de la página web de SCEE es la siguiente: http://www.technology.scee.net/psp-edu
En esta página pone, que el programa de PSP devkit está temporalmente cerrada. Para mostrar su interés, se debe enviar un correo a: "applications@ps-edu.scedev.net" con los datos de contacto y con la información de la universidad. Durante los intentos de tener contacto con SCEE, la página web fue actualizada. En particular, los contenidos del campo de "academia" fueron cambiados. Ahora, para mostrar su interés hay que rellenar un formulario.

Para tener más informaciones sobre el tema de licencia y de PSP devkit, se ha enviado un correo postal a SCEE a Londres. Pero nunca se ha recibido respuesta de ellos. Para no perder más tiempo, el proyecto fue programado con un SDK no oficial.

Alfred Bernhard Nobel dijo una vez:"saber cómo utilizar un diccionario es mejor que creer que eres uno", lo que demuestra la gran importancia de un diccionario. Hoy en día, hay muchas formas de tener documentos electrónicos en el dispositivo portable. El internet existen unos diccionarios online, lo más famoso y conocido es el traductor de Google. Al lado de Google, el diccionario "Leo"[6] ofrece al usuario un diccionario para el ordenador y dispositivo portable. Dicha aplicación puede traducir directamente una palabra marcada en un documento de texto (p.ej. PDF). La versión para los hogares es gratuita, las empresas deben comprar una licencia. La comunidad de "Leo" no posee solo diccionarios si no que también un foro. En el foro, el usuario tiene la posibilidad de preguntar dudas sobre la gramática, traducción de una palabra y las diferencias entre los idiomas. Un ejemplo concreto, el español de España y el español de México poseen de unas diferencias. La palabra "guay / mola" en España corresponde a la palabra "chévere" en México. En los dispositivos portables o móviles también existen diccionarios electrónicos. El Ipod e Iphone ofrecen aplicaciones de descarga, también diccionarios.

[6] Leo es un diccionario online famoso en Alemania. Dispone diccionarios online de inglés ↔ alemán, francés ↔ alemán, español ↔ alemán, italiano ↔ alemán, chino ↔ alemán y por última ruso ↔ alemán.

En el caso de videoconsolas, Nintendo dispone unas AA p.ej. "*Brain Magister con el Dr. Kawashima*"[7], también unos diccionarios p.ej. "*Langenscheidt diccionario NDS*[8]" y "*Berlitz: diccionario básico inglés*[9]". Por lo contrario, la PSP no dispone de ningún diccionario, solo una AA con el nombre "Play English"[10]. Sin embargo, la PSP es más potente que la NDS. Para ver exactamente la diferencia de PSP y NDS, es necesario comparar las especificaciones técnicas a través de una tabla. En este caso de la PSP de tercer generación (PSP 3000), de la nueva PSP (E-1000), de la PSP Vita, de la NDSi y por última la N3DS. La nueva generación de PSP (E-1000), solo se va a publicar en Europa como una PSP más barata [4]. La sucesora de la PSP (Vita) va a ser "región free", quiere decir, que es posible comprar una PSP Vita de Japón y jugar los juegos de Estado Unidos o Europa. Además Sony publica dos tipos de PSP Vita - versión 3G y la versión normal.

En la **Tabla 1** se muestra un breve resumen de los datos importantes de PSP (3000) y de la NDSi, para realizar una comparación rápida entre todas las videoconsolas.

	PSP 3000	**NDSi**
CPU	MIPS R4000-based; frecuencia de reloj 1~333MHz	Uno de 67.028 MHz ARM946E-S y otra de 33.514 MHz ARM7TDMI
Memoria	64 MB	4 MB
Soporte	Memory Stick Duo, Memory Stick PRO Duo (1, 2, 4, 8, 16, 32, 64 or 128 GB)	4 MB RAM Cartridge save
Pantalla	480 × 272 pixeles con 16.8M colores, 4:3 pulgadas TFT LCD	256 × 192 pixeles con 260K colores, 3:25 pulgadas, TFT LCD

Tabla 1: Comparación de PSP 3000 y NDSi.

[7] La página web del producto del Dr. Kawashima : http://www.nintendo.es/NOE/es_ES/games/nds/brain_training_del_dr_kawashima_cuntos_aos_tiene_tu_cerebro_3234.html

[8] Información de un blog sobre el "Langenscheidt diccionario" : http://babel20.blogspot.com/2008/04/langenscheidt-publica-un-diccionario.html

[9] Información sobre el "Berlitz" : http://www.penreader.com/java-software/de/Berlitz/Berlitz_Diccionario_B%C3%A1sico_Espa%C3%B1ol-Ingl%C3%A9s.html

[10] Información sobre el juego "Play English": http://www.pspgamesthemes.com/psp-games/other-games/play-english.html

En la **Tabla 2**, se muestra un breve resumen de los datos importantes de las sucesoras de PSP (PSPVita) y de NDS (N3DS)

	PSP Vita	**N3DS**
CPU	4 core ARM Cortex-A9 MPCore	Nintendo ARM
Memoria	512 MB RAM, 128 MB VRAM	128 MB FCRAM
Soporte	NVG Card de 4 GB a 32 GB	2GB NAND flash memory
Pantalla	960 × 544 pixels con 24-bit colores, 5 pulgadas, OLED multi-touch capacitive touchscreen	800 × 240pixeles (por ojo 400 × 240 pixeles de WQVGA) con 24-bit colores, 5.3pulgadas,
Camara	640×480, 60 fps, 320×240 120 fps	

Tabla 2: Comparaciones de los sucesores de PSP y NDS.

Por último, falta la comparación entre la PSP de la última generación (3000) y la nueva PSP más económica (E-1000)[11]. La diferencia entre ambas PSP consiste en el hecho de qué la PSP E-1000 no tiene ninguna conexión WiFi. Además, la dimensión de la PSP es distinta. En general no hay muchas diferencias entre ambas generaciones / versiones.

Aparte de estos datos, es importante ver las características de un diccionario. Mejor dicho, lo que debe hacer el diccionario. En concreto, traducir, dar ejemplos de cómo usar la palabra o en qué caso se usa, exponer la pronunciación en forma de texto o audio. Todos estos detalles depende de la implementación y si el dispositivo lo soporta por el tema de hardware (tarjeta gráfica, tarjeta de sonido, memoria, etc.).

[11] Según la página web de Sony [4], va a costar 99,99 euros, sin embargo una nueva PSP 3000 cuesta 129,99 euros en amazon.es.[9]. Por la falta de WiFi sería mejor comprarse una PSP 3000 en vez de la PSP E-1000.

4. Instalación

Para poder crear una AM, "Homebrew" o "Minis" [1] en la videoconsola son necesarios unos requisitos y modificaciones del firmware de videoconsola. En el caso de PSP, es necesario leer la documentación de Anush Euredjian Chiappe y Juan Cortés Maiques [2] sobre la instalación de un CFW. La razón para la instalación / cambio de firmware es la siguiente, una PSP con OFW no puede ejecutar "Homebrew", porque la PSP bloquea la ejecución por protección anti copia, si no el usuario podría descargar los ISOs por internet y cargarlos en la MemoryStick.

¿Qué es un "Homebrew"?

Son aplicaciones escritas caseras realizadas por programadores para mejorar el uso de la PSP. Quiere decir, como por ejemplo una aplicación que puede leer ficheros PDF o emular las antiguas consolas para poder jugar sus juegos en la PSP (p.ej poder jugar el primer Super Mario del Gamboy) [6]. También existe en la comunidad un emulador para la NDS. Dicho emulador no funciona en todas las PSP[12].

¿Qué es CFW o OFW?

OFW, como indica la abreviatura el "original Firmware" de Sony para la PSP. En este caso es como un SO de la PSP, porque cada nueva OFW de nuevas opciones p.ej. los nuevos ofrecen "Digital Comic Reader"[13]. Por el contrario, CFW permiten ejecutar "Homebrew", en el funcionamiento igual a OFW. [6]

¿Qué es PRX?

Es la extensión para ficheros de "Plug-in"[6], se usa para modificar el "Firmware" o añadir extra "features" como por ejemplo poder usar el "RemoteJoy" en la PSP para capturar la salida de la señal de la pantalla en el ordenador a través de USB. Asimismo, es posible cifrar y descifrar los PRX. Por ejemplo, algunos juegos requieren una modificación del "EBOOT.BIN" para poder jugarlos o ejecutarlos en el XMB, en este caso se necesita un descifrador de PRX. Por otra parte, para escribir un "Homebrew" se puede crear también un PRX para asignar el "Homebrew" con la instrucción en el Makefile **BUILD_PRX = 1**. Cuando se compila un código o un proyecto, se puede crear también un fichero PRX

[12] Enlace sobre la información del emulador de NDS para la PSP. http://psp.dashhacks.com/2010/05/21/dsonpsp-unofficial-v0-1-released-nintendo-ds-emulator-for-psp/

[13] Descargar comics de PSN y leerlos en la PSP: http://us.playstationcomics.com/main.html

. La estructura de un PRX se puede ver en la **Figura 1**.

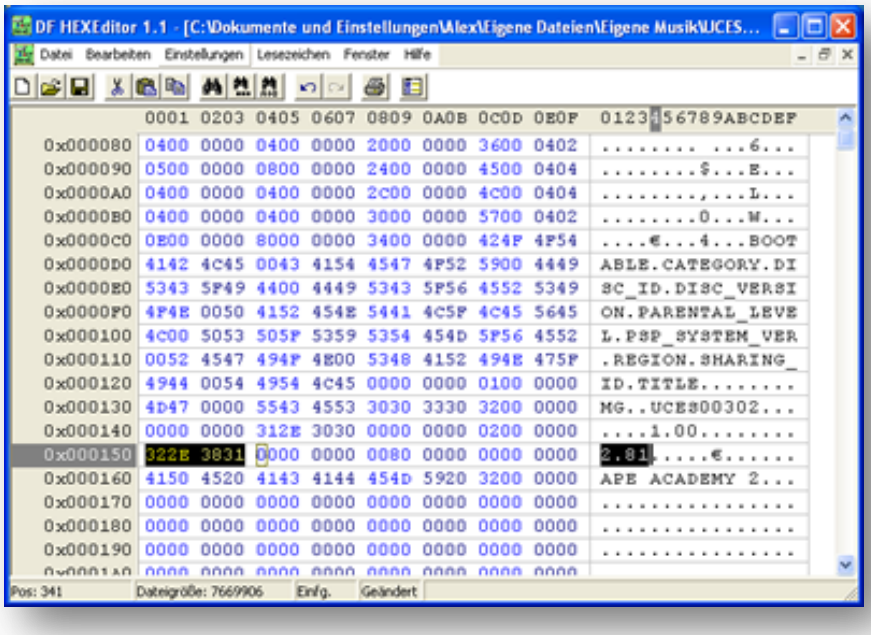


Figure 1: Estructura de un fichero PRX.

Un PRX es en realidad un programa que se ejecuta en la PSP en cierto momento. Quiere decir, el PRX vshmain es responsable para acceder en el menú principal de la PSP con todas las informaciones necesarias. Mejor dicho, cuando se arranca la PSP, arranca todos los programas del XMB. No es necesario arrancar el programa a mano a través del XMB como en el caso de EBOOT.PBP. Se puede programar su propio sistema de identificación cuando se arranca la PSP. La PSP muestra el XMB solo sí la contraseña fue correcta [7]. También es posible programar un EBOOT.PBP que usa un PRX. Para saber como programar un PRX, es recomendable saber primero cómo programar un EBOOT.PBP en ambos modos (modo usuario, modo kernel) [8]. En el ejemplo del SDK de WiFiscan[14], se crea un PRX que hace referencia a la librería de **sceNet_lib** del **pspnet.prx** para realizar un escaneo de WiFi. Más información de las funciones de un PRX se puede encontrar en la página web http://silverspring.lan.st/.

[14] Ruta para encontrar el ejemplo de WiFiscan: C:\pspsdk\psp\sdk\samples\net\wlanscan

Los requisitos son los siguientes:

- Instalación del SDK (MinPSPw[15] versión 11.2 ó 0.11.2r3)
- Instalación de un editor de texto (p.ej. notepad++)
- Instalación del nuevo CFW (p.ej. 5.50 Prome-2)
- Instalación de un emulador de PSP para el ordenador/portátil (jpcsp)
- Sony MemoryStick PRODuo (128 MB - 2GB)

Configuración de Visual Studio Ultimate con MinPSPw

La instalación de MinPSPw es muy sencilla sin ninguna complicación. El directorio principal de programa es "C:\pspsdk" donde instala todos los componentes necesarios para programar un "Homebrew". Este proyecto utiliza la versión actual 11.2 que se llama también 0.11.2r3 (*http://www.jetdrone.com/post/1/30/minpsp-11.2)* (**Figura 2** y **Figura 3**)

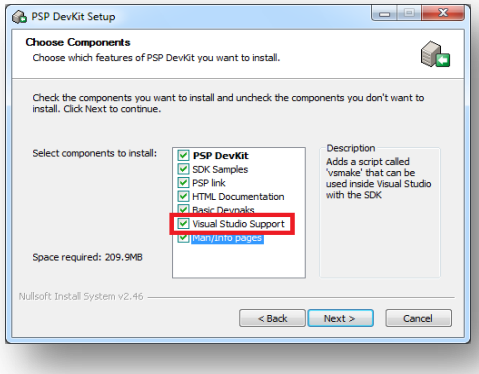


Figura 2: Instalación de MinPSPw.

OJO: Es muy importante marcar **Visual Studio Support** para los pasos siguientes y acordarse de la ruta de instalación de MinPSPw.

[15] Es importante mirar que versión de MinPSPw se descarga, porque la versión actual puede dar problemas de compilación en el caso de SDL audio.

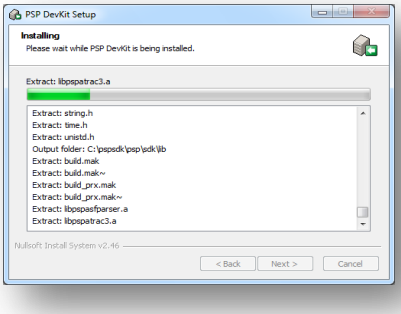


Figura 3: Instalación de los componentes necesarios de MinPSPw.

Después de la instalación de MinPSPw, se puede instalar Visual Studio Express 2010 o Visual Studio Ultimate 2010. También es posible programar con las versiones antiguas de Visual Studio (2003 – 2008). La instalación de VS dura mucho tiempo, porque VS instala muchos componentes.

Después de la instalación de VS se crea un nuevo proyecto.

OJO: Cuando se crea un nuevo proyecto, hay que volver a hacer todos los pasos en Visual Studio. Por lo demás es muy importante que la PSP esté conectada y que exista la carpeta con todos los componentes necesarios, que hacen falta (imágenes sonidos, etc.)

Visual Studio copiará únicamente el ejecutable **EBOOT.PBP** en la PSP. Los ficheros: *.o, *.PRX, *.elf y PARAM.SFO estarán en el directorio de proyecto.

Es muy recomendable usar el directorio del Emulador JPCPS, que es: **".\MS0\PSP\GAME\"** para los proyectos de Visual Studio. Debido a esto es posible trabajar con el emulador en cualquier momento, en vez tener la PSP conectada.

En la ventana principal hay que pinchar en **New Project**, como se muestra en la **Figura 4**.

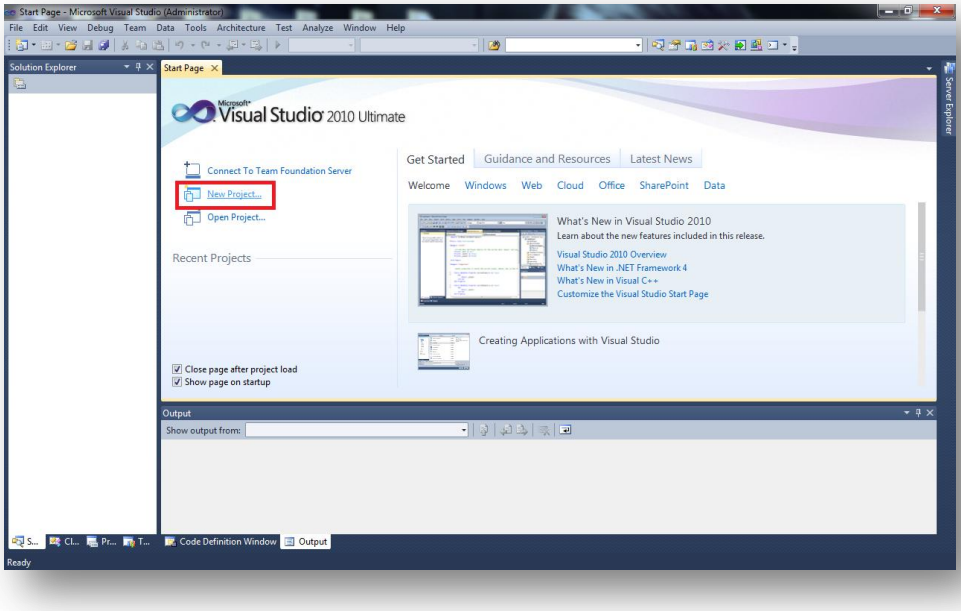


Figura 4: Ventana principal de Visual Studio Ultimate 2010.

Después hay que indicar el tipo de proyecto, en este caso: **Makefile Project**, como se puede ver en la **Figura 5.**

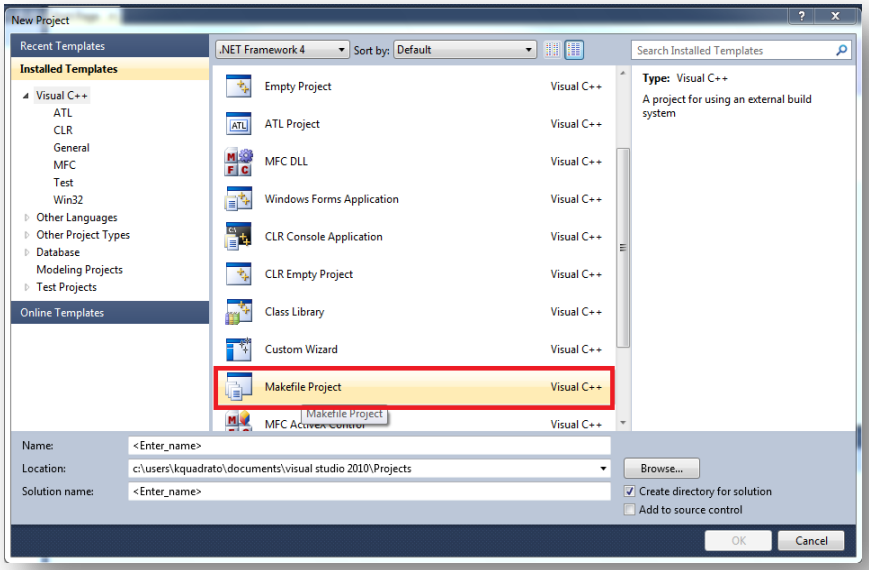


Figura 5: Selección de tipo de proyecto.

Ahora hay que decir donde debe estar guardado el proyecto y con qué nombre. En este caso, se debe poner en los campos de texto lo siguiente, como en la **Figura 6** mostrado:

Name: hola mundo
Location: (donde el emulador está instalado)\MS0\PSP\GAME\
Solution name: hola mundo

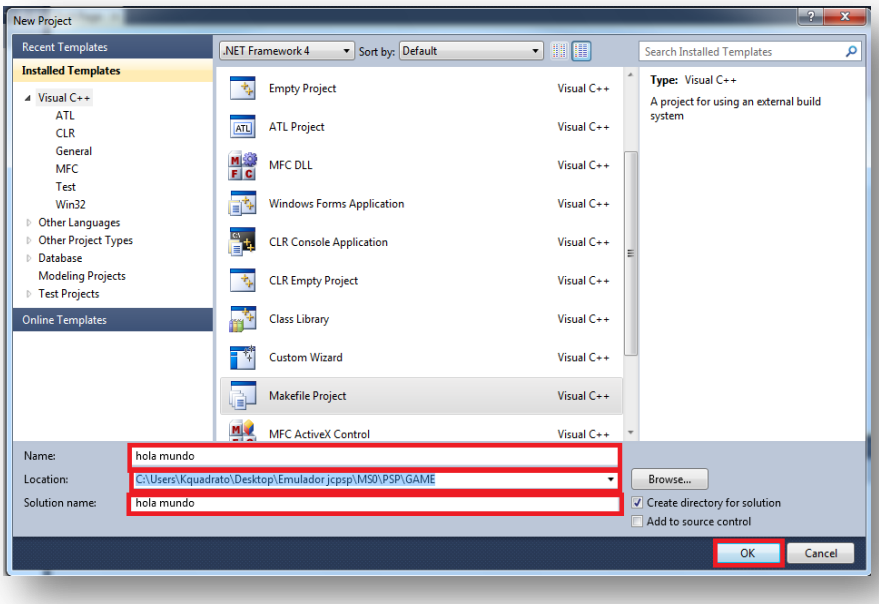


Figura 6: Indicar nombre y localización de proyecto de VS 2010.

Después de pinchar a **OK**, sale una nueva ventana donde habrá que pinchar en **Next** (**Figura 7**).

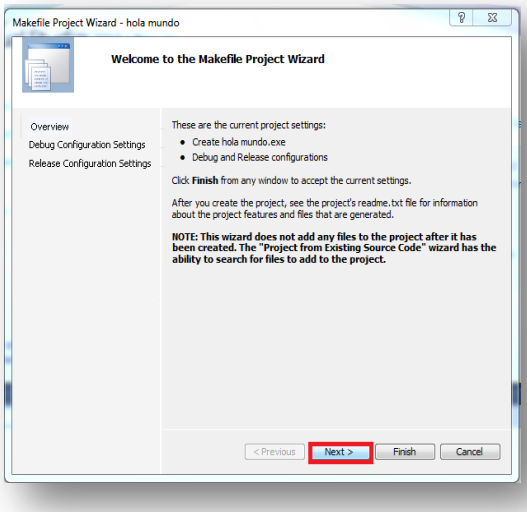


Figura 7: Ventana después de indicar la localización de proyecto y su nombre.

Ahora se debe ver, porque ha sido importante marcar Visual Studio Support en la instalación de MinPSPw, para poder copiar el ejecutable (EBOOT.PBP) en la PSP. Hay que introducir a Visual Studio dónde lo tiene que copiar y cuál es el fichero de salida de debugger y la ruta de include. Se muestra en la **Figura 8**.

Build command line: vsmake && copy EBOOT.PBP "(letra de la PSP):\PSP\GAME\(nombre de la aplicación)\"
clean commands: vsmake clean
Rebuild command line: vsmake clean && vsmake && copy EBOOT.PBP "(letra de la PSP):\PSP\GAME\(nombre de la aplicacion)\"
Output (for debugging): EBOOT.PBP
Include search path: C:\pspsdk\psp\sdk\include (si se ha instalado MinPSPw en la ruta general siendo ésta C: \pspsdk)

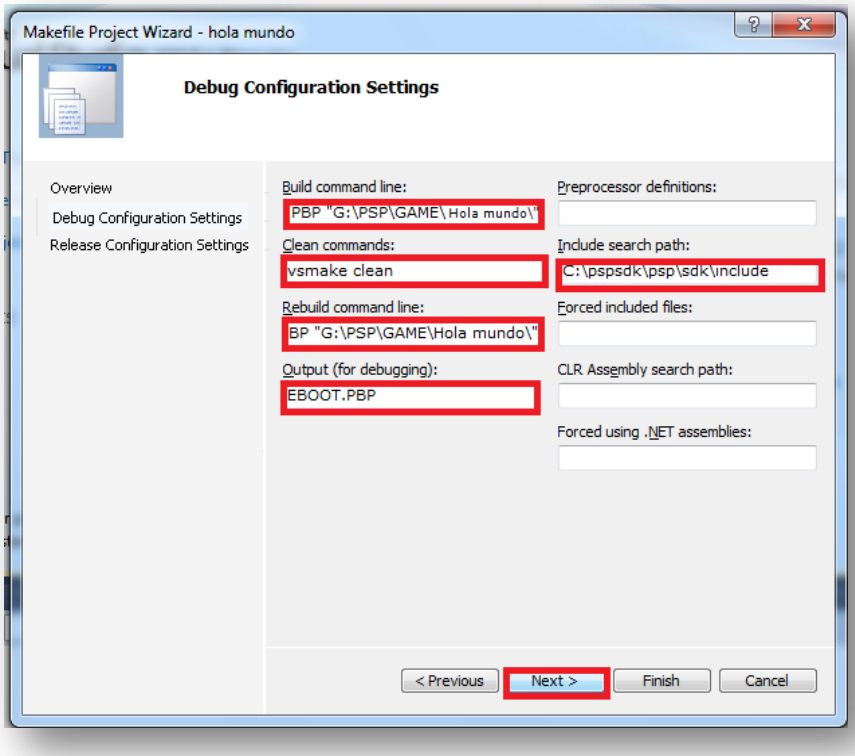


Figura 8: "Instalación" / "Modificación" de VS 2010.

OJO: Es muy importante tener los mismos nombres en **Build command line** y **Rebuild command line.** Si no se puede tener problemas de compilación y transmisión del ejecutable.

En la ventana siguiente hay que marcar **Same as debug configuation** y pinchar en **Next.** Como se puede ver en la **Figura 9**

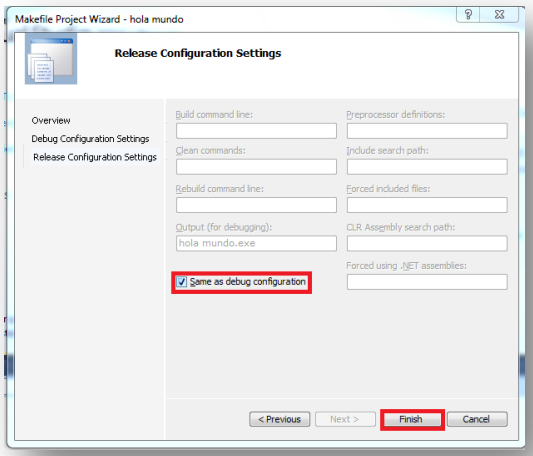


Figura 9: Marcar la opción "Same as debug configuration".

Ahora se ha creado el nuevo proyecto en VS. Así, se puede empezar a añadir un fichero *.cpp a través de pinchar con el botón derecho del ratón en el nombre de proyecto (que está dentro de la ventana Solution explorer) **Add->New Item** y seleccionar C++ File(.cpp) o también con las teclas **[Ctrl] + [Shift] + [A]**, como se muestra en la **Figura 10.**

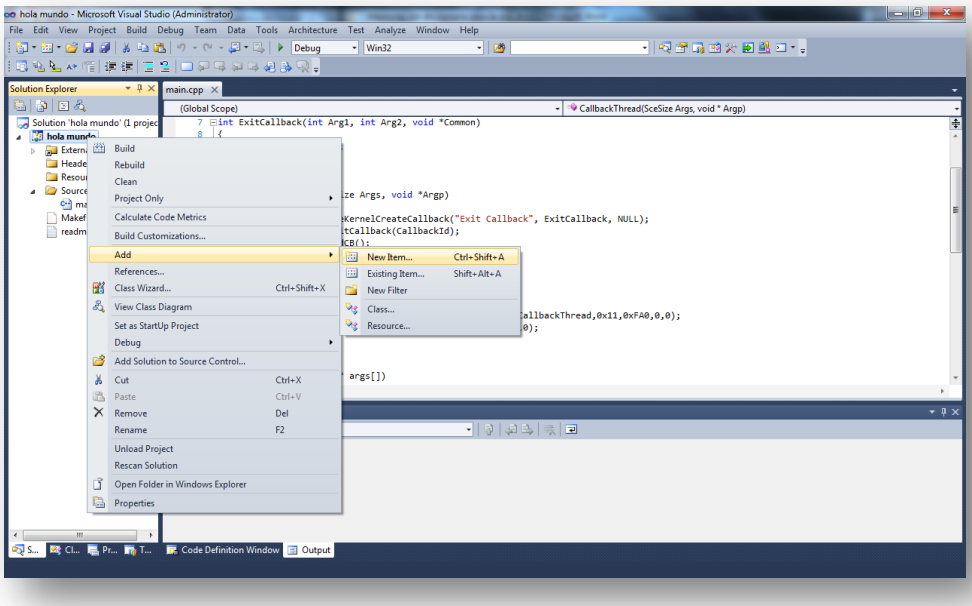


Figura 10: Inserción de un nuevo ítem en VS 2001.

En este estudio se programa en C++, por este motivo es necesario coger un fichero de tipo C++ (*.cpp) y en el caso de programación con clases es necesario añadir los ficheros headers (*.h) en el proyecto. Pero en el primero caso "hola mundo" solo se necesita un fichero *.cpp.

En este caso, seleccionar **C++ File(.cpp)** y pinchar en **add**, como se muestra en la **Figura 11**

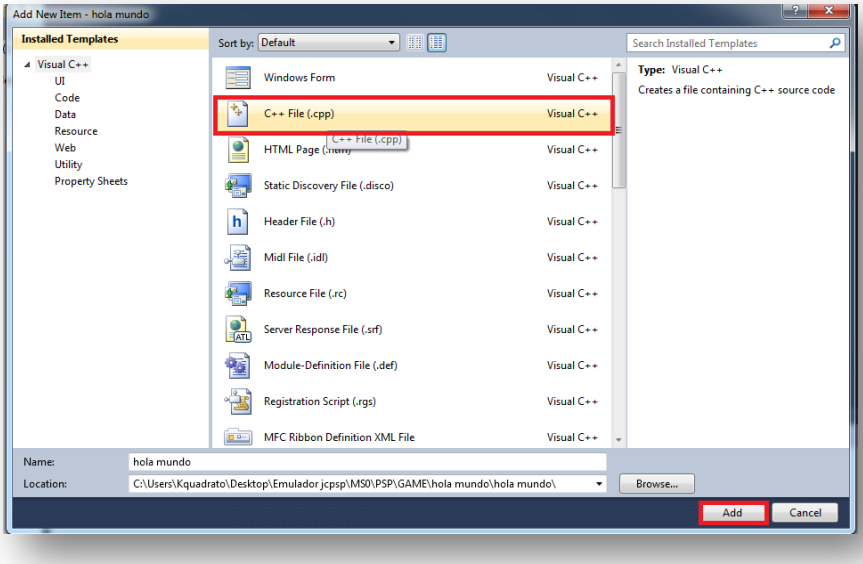


Figura 11: Selección de un fichero *.cpp.

Ahora hay que comprobar si todo está bien configurado en VS. La manera más rápida y fácil de comprobarlo, es copiar el código fuente de "holamundo.cpp" y compilarlo en VS. Luego para asegurar si el ejecutable funciona bien, hay que ejecutar "holamundo" en el XMB de la PSP. Como comentado antes, por la opción de instalación de MinPSPw, es posible copiar el ejecutable directamente en la PSP. Pero VS requiere que la carpeta ya exista en la PSP, si no, no copia el ejecutable y lanza un mensaje del compilador de error.

El mismo mensaje ocurre en el caso si la PSP no está conectada:

```
C:\Program Files\MSBuild\Microsoft.Cpp\v4.0\Microsoft.MakeFile.Targets(38,5): error
MSB3073: The command "vsmake && copy EBOOT.PBP "G:\PSP\GAME\PSPDictionary\"" exited
with code 1.
```

El código fuente de "holamundo.cpp":

```
#include <pspkernel.h>
#include <pspdebug.h>
#include <pspdisplay.h>

PSP_MODULE_INFO("Hello World",0,1,1);

int ExitCallback(int Arg1, int Arg2, void *Common)
{
     sceKernelExitGame();
   return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
   int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
   sceKernelRegisterExitCallback(CallbackId);
   sceKernelSleepThreadCB();
   return 0;
}

int SetupCallbacks(void)
{
   int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
      if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
   return ThreadId;
}

int main(int argc, char* args[])
{
   pspDebugScreenInit();
   SetupCallbacks();
   pspDebugScreenPrintf("Hello World");
   sceDisplayWaitVblankStart();
   sceKernelSleepThread();

return 0;
}
```

OJO: El código de "holamundo.cpp" es casi igual al código de "holamundo.c". La diferencia consiste en la instrucción: sceKernelSleepThread(); que es necesario en el caso C++. Si no la PSP cierra la aplicación y se queda bloqueada antes de volver a la XMB. Por el interbloque de hilos se apaga después de unos segundos

Hola Mundo

El código fuente de "Holamundo" sirve para aprender cómo imprimir un mensaje por consola en la PSP. Asimismo, es un famoso ejemplo para comprobar si la instalación de SDK fue un éxito. Las tres funciones importantes para crear un "Homebrew" y para poder volver en el XMB de la PSP son los siguientes:

- int ExitCallback(int Arg1, int Arg2, void *Common)
- int CallbackThread(SceSize Args, void *Argp)
- int SetupCallbacks(void)

OJO: Estos 3 métodos hay que escribirlos siempre, independientemente del caso (programación en C o en C++).

IntelliSense

VS dispone la posibilidad de un menú de IntelliSense, que es un autocompletar sirviendo como documentación y desambiguación de los nombres de variables, funciones y métodos de utilización de metadatos basados en la reflexión. Pero hay que llamarlo siempre con las teclas **[Ctrl] + [Space]**. Un ejemplo de uso de InteliiSense se muestra en la **Figura 12**

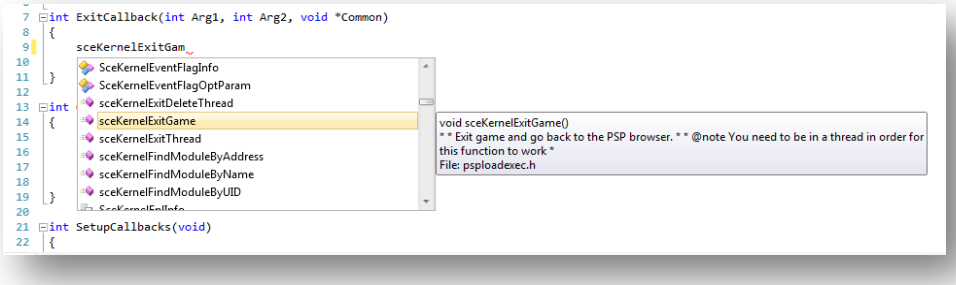


Figura 12: Menú IntelliSense en VS.

Ahora es menester crear el fichero Makefile, para compilar y crear el ejecutable. VS no permite crear ficheros sin extensión. A consecuencia de ello es obligatorio tener un editor de texto, que permita crear ficheros sin extensión como notepad++ o se quita la extensión a mano. Además se debe ver porqué el directorio de proyecto está situado en la carpeta de emulador con la ruta: "**. \MS0\PSP\GAME\".** En la **Figura 13**, se puede ver la ruta del proyecto de "Hola Mundo"

Para seguir, hay que escribir ahora el Makefile. Para la primera prueba se recomienda copiar el siguiente Makefile. La estructura de Makefile se explica más adelante.

Makefile:

```
TARGET = Hola

OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS = -lstdc++

BUILD_PRX = 1
PSP_FW_VERSION = 371
EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = Hola mundo

PSPSDK=$(shell psp-config --pspsdk-path)
include $(PSPSDK)/lib/build.mak
```

OJO: Si el Makefile está usando ficheros *.png y *.at3, es necesario tenerlas dentro del proyecto.

¡No es necesario añadirlos en VS, porque no hay que modificarlas en / con VS!

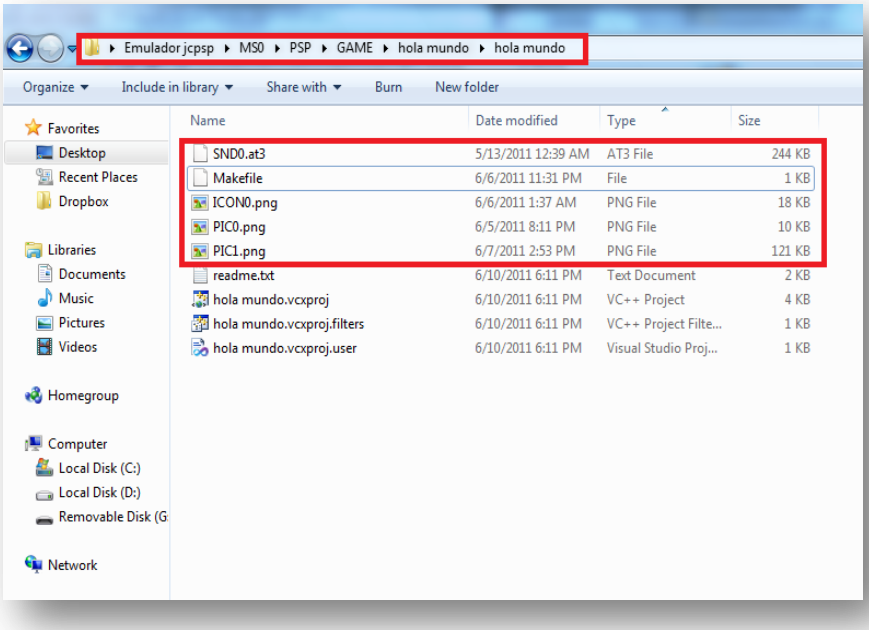


Figura 13: La ruta del proyecto de VS 2010.

Se puede añadir el Makefile a través del botón derecho del ratón en el nombre del proyecto (que está dentro de la ventana Solution explorer) **Add->Existing Item** y selecciona el Makefile o también con las teclas **[Shift] + [Alt] + [A]**, como se muestra en la **Figura 14**.

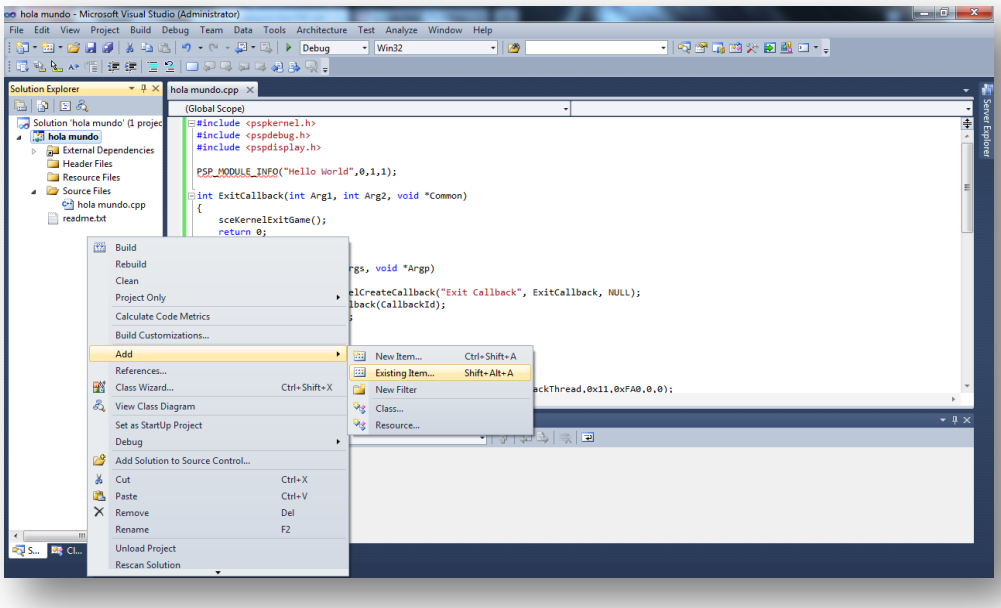


Figura 14: Inserción de Makefile en VS 2010.

Ahora se selecciona el "Makefile". VS sabe redirigir directamente a su ruta del proyecto, otra razón por la que es importante crear el proyecto en la ruta del emulador: "**.\MS0\PSP\GAME\".** En la **Figura 15** se puede ver. La **Figura 16** muestra la ruta del proyecto en la PSP

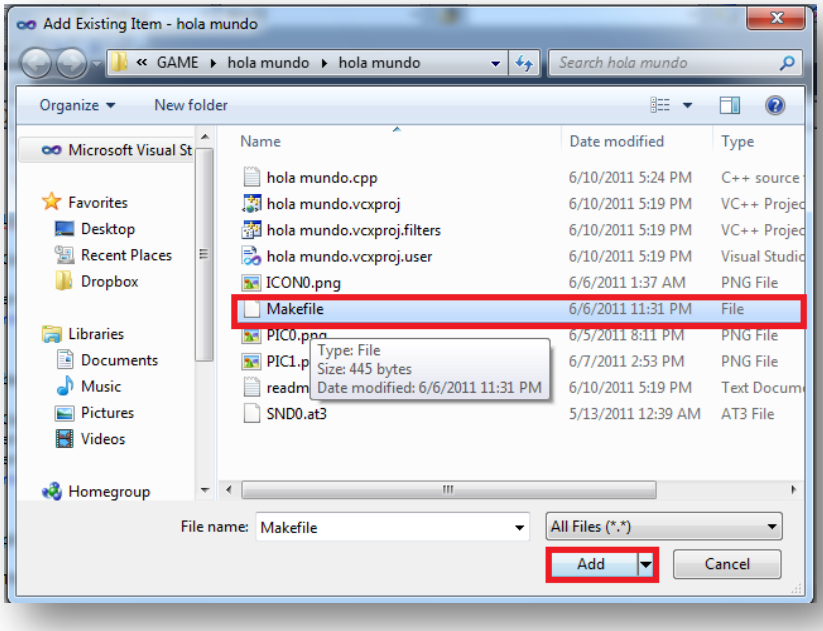


Figura 15: Selección de Makefile en VS 2010.

Antes de compilar el proyecto hay que conectar la PSP y crear dentro de la PSP una nueva carpeta con el nombre que se ha escrito en el apartado de "modificar" el proyecto de VS en la línea de **"Build command line"**:

Build command line: vsmake && copy EBOOT.PBP "(letra de la PSP):\PSP\GAME*(nombre de la aplicación)*\".

OJO: No es posible volver a mirar el nombre que se ha tecleado en **Build command line** y **Rebuild command line** en las opciones de VS.

Si no se acuerda del nombre, hay que mirar en el mensaje del compilador de VS después de compilar el proyecto. El último mensaje nos da el "nombre de la aplicación" de la siguiente forma: "`C:\Program Files\MSBuild\Microsoft.Cpp\v4.0\Microsoft.MakeFile.Targets(38,5): error MSB3073: The command "vsmake && copy EBOOT.PBP "(letra de la PSP):\PSP\GAME\nombre de aplicación\"" exited with code 1.`

En el peor caso, no se puede compilar el proyecto.

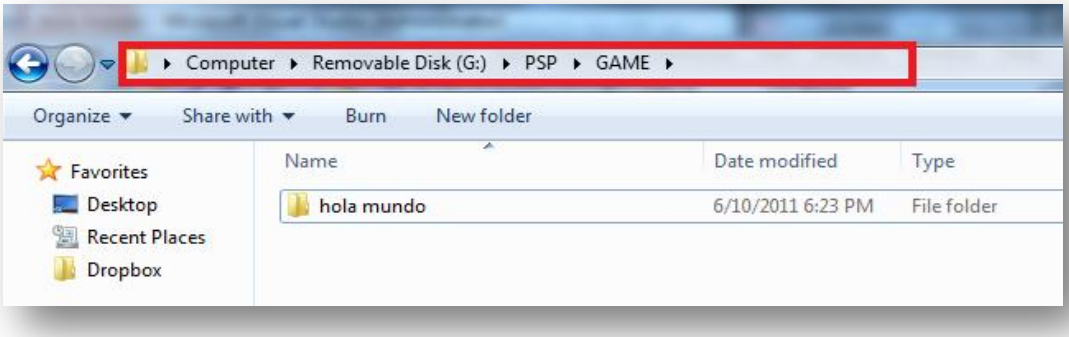


Figura 16: Ruta de creación de carpeta necesaria en la PSP.

Ahora es posible compilar y crear el ejecutable, porque VS contiene todos los ficheros necesarios, además la PSP está conectada. Hay que pinchar a **Build->Build(nombre de proyecto)** en la pestaña del menú principal o con las teclas **[Ctrl] + [Shift] + [B]**. Si han ocurrido errores de compilación, se pueden consultar en la lista de errores (Error List) o también con las teclas **[Ctrl] + [E]**, como se ilustra en la **Figura 17** y **Figura 18.**

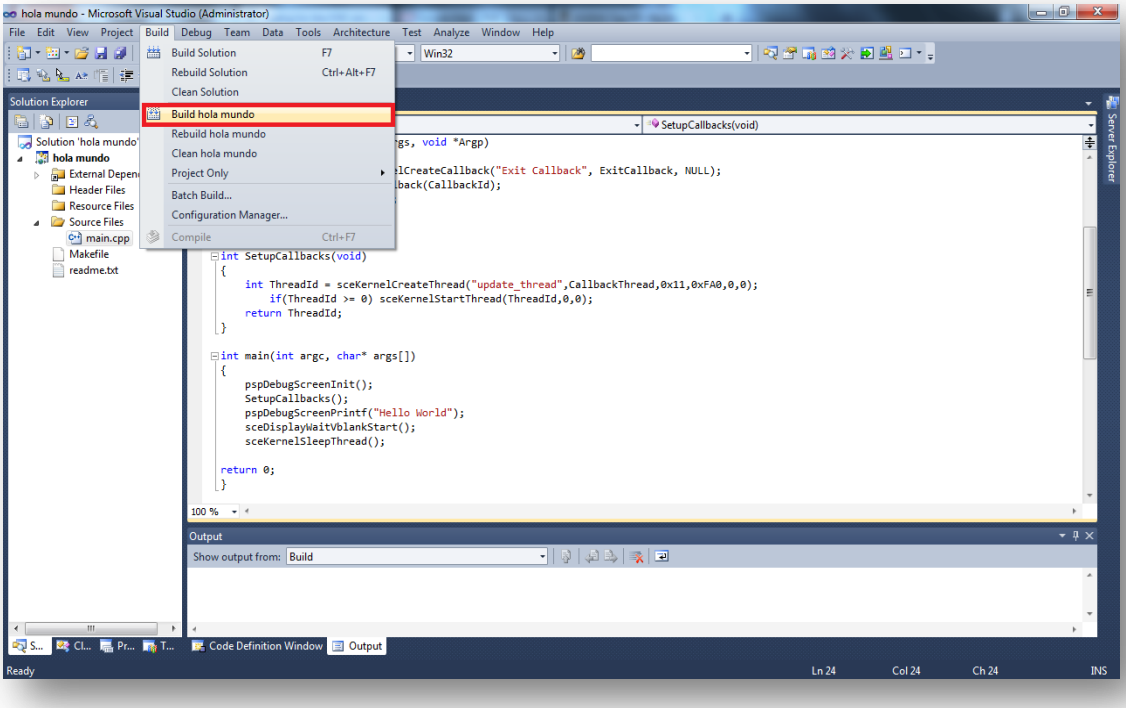


Figura 17: Ejemplo de cómo compilar el código fuente en VS.

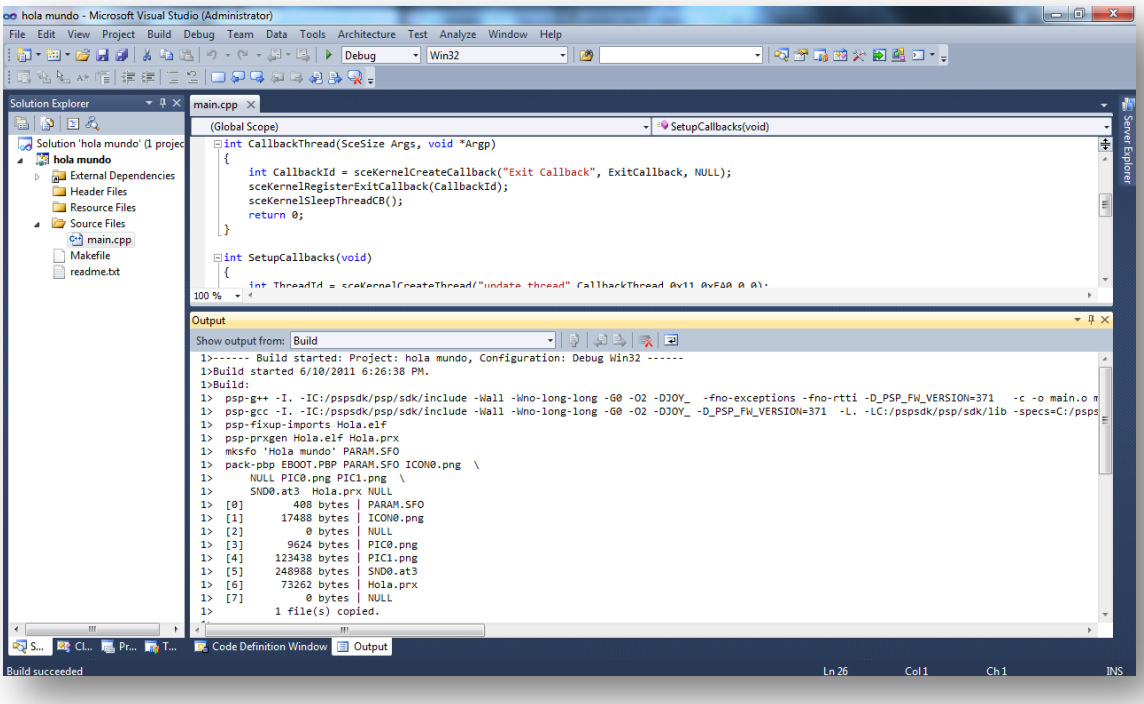


Figura 18: Compilación del proyecto en VS.

Salida de la compilación del proyecto en VS:

```
1>------ Build started: Project: hola mundo, Configuration: Debug Win32 ------
1>Build started 6/10/2011 6:26:38 PM.
1>Build:
1>  psp-g++ -I. -IC:/pspsdk/psp/sdk/include -Wall -Wno-long-long -G0 -O2 -DJOY_  -fno-exceptions -fno-rtti -
D_PSP_FW_VERSION=371   -c -o main.o main.cpp
1>  psp-gcc -I. -IC:/pspsdk/psp/sdk/include -Wall -Wno-long-long -G0 -O2 -DJOY_ -D_PSP_FW_VERSION=371  -L. -
LC:/pspsdk/psp/sdk/lib -specs=C:/pspsdk/psp/sdk/lib/prxspecs -Wl,-q,-TC:/pspsdk/psp/sdk/lib/linkfile.prx   main.o
C:/pspsdk/psp/sdk/lib/prxexports.o -lstdc++  -lpspdebug -lpspdisplay -lpspge -lpspctrl -lpspsdk -lc -lpspnet -lpspnet_inet -
lpspnet_apctl -lpspnet_resolver -lpsputility -lpspuser -lpspkernel -o Hola.elf
1>  psp-fixup-imports Hola.elf
1>  psp-prxgen Hola.elf Hola.prx
1>  mksfo 'Hola mundo' PARAM.SFO
1>  pack-pbp EBOOT.PBP PARAM.SFO ICON0.png  \
1>    NULL PIC0.png PIC1.png  \
1>    SND0.at3  Hola.prx NULL
1>  [0]        408 bytes | PARAM.SFO
1>  [1]      17488 bytes | ICON0.png
1>  [2]          0 bytes | NULL
1>  [3]      73262 bytes | Hola.prx
1>  [4]          0 bytes | NULL
1>         1 file(s) copied.
1>
1>Build succeeded.
1>
1>Time Elapsed 00:00:01.26
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

El resultado de la ejecución de "Hola Mundo" en el emulador o en la PSP, se ve en la **Figura 19**.

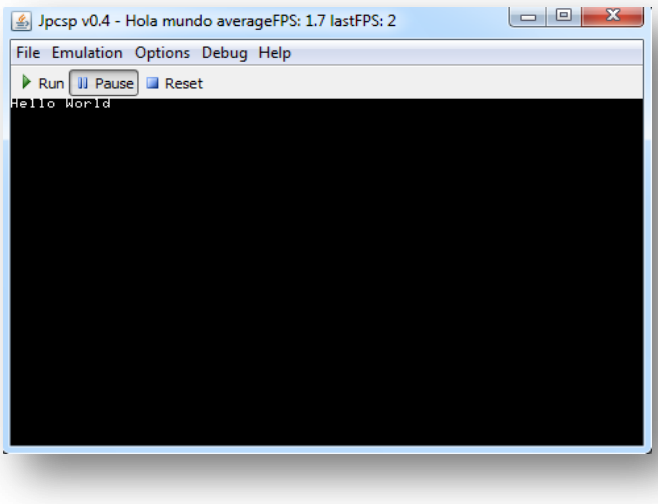


Figura 19: Captura de pantalla del emulador que está ejecutando "Hola mundo" - la versión C++.

Ahora es posible desconectar la PSP del ordenador y comprobar la aplicación directamente en la PSP o también con un emulador.

OJO: Visual Studio solo copia el ejecutable en la PSP. Si se usa imágenes, sonidos o cualquier otro tipo de fichero.
¡Hay que copiarlos en la PSP, en la carpeta de su aplicación!
Si no la PSP no muestra nada por pantalla o en el peor caso, la PSP se apaga por no encontrar los ficheros.

Programar con o sin IDE

Para mostrar que no es obstante programar con un IDE, el código fuente de "Hola mundo.cpp" fue compilado en el símbolo del sistema de Windows 7. El resultado de la compilación es igual. La diferencia entre programar con IDE y sin IDE consiste en los mensajes del compilador. Quiere decir, el IDE ofrece la opción de acceder directamente a la línea de código donde ocurrió el fallo. Asimismo, el programador tiene la opción de compilar y ejecutar su programa paso por paso, para poder ver el cambio de variables. En el caso de PSP, esta opción no está disponible porque el emulador JPCSP no lo ofrece.

La **Figura 20** muestra la compilación en el propio símbolo del sistema de Windows

La compilación de proyecto en símbolo del sistema:

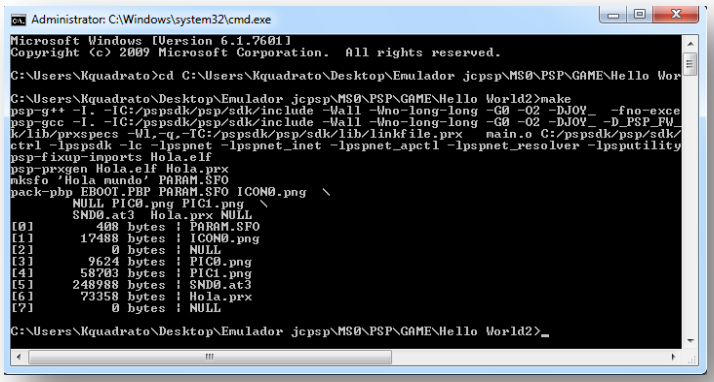


Figura 20: Compilación del proyecto en el símbolo del sistema.

Salida de la compilación del proyecto en el símbolo del sistema:

```
Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation.  All rights reserved.
C:\Users\Kquadrato>cd C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\hola mundo\hola mundo
C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\hola mundo\hola mundo>make
psp-g++ -I. -IC:/pspsdk/psp/sdk/include -Wall -Wno-long-long -G0 -O2 -DJOY_  -fno-exceptions -fno-rtti -
D_PSP_FW_VERSION=371   -c -o main.o main.cpp
psp-gcc -I. -IC:/pspsdk/psp/sdk/include -Wall -Wno-long-long -G0 -O2 -DJOY_ -D_PSP_FW_VERSION=371  -L. -
LC:/pspsdk/psp/sdk/lib -specs=C:/pspsdk/psp/sdk/lib/prxspecs -Wl,-q,-TC:/pspsdk/psp/sdk/lib/link
file.prx   main.o C:/pspsdk/psp/sdk/lib/prxexports.o -lstdc++  -lpspdebug -lpspdisplay -lpspge -lpspctrl -lpspsdk -lc -lpspnet -
lpspnet_inet -lpspnet_apctl -lpspnet_resolver -lpsputility -lpspuser -lp
spkernel -o Hola.elf
psp-fixup-imports Hola.elf
psp-prxgen Hola.elf Hola.prx
mksfo 'Hola mundo' PARAM.SFO
pack-pbp EBOOT.PBP PARAM.SFO ICON0.png  \
       NULL PIC0.png PIC1.png  \
       SND0.at3  Hola.prx NULL
[0]       408 bytes | PARAM.SFO
[1]     17488 bytes | ICON0.png
[2]         0 bytes | NULL
[3]     73358 bytes | Hola.prx
[4]         0 bytes | NULL

C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\hola mundo\hola mundo>
```

4.1. Herramientas

Este apartado del estudio, habla de las herramientas usadas para realizar el diccionario. Desafortunadamente existían problemas con una de ellas, en concreto con Code::Blocks.

MinPSPw:

"Es una adaptación del BSD PSPSDK de ps2dev.org de libre distribución. Incluye una versión compilada de las distintas librerías y cabeceras necesarias para compilar Hombrew en la PSP. Según su propio autor, este proyecto nació con la idea de hacer más sencilla la instalación en sistemas Windows y no depender de Cygwin o programas similares de forma que una vez se instale el SDK y un par de DEVPAKs se pueda utilizar en Windows de forma "nativa". Este proyecto es diferente del oficial PSP SDK y DevkitPro y es posible que sus librerías defieran de la última versión de estos." (Anush Euredjian Chiappe; Juan Cortés Maiques,2009)[2].

Code::Blocks:

"Es […] un entorno de desarrollo Open Source y multiplataforma completamente configurable desarrollado en C++ construido como un núcleo altamente expansible mediante complementos (plu-gins). Actualmente la mayor parte de la funcionalidad viene provista por los complementos incluidos predeterminadamente. No es un IDE autónomo que acepta complementos, sino que es un núcleo abstracto donde los complementos se convierten en una parte vital del sistema.

Esto lo convierte en una plataforma muy dinámica y potente, no solo por la facilidad con que puede incluirse nueva funcionalidad, sino por la capacidad de poder usarla para construir otras herramientas de desarrollo tan solo añadiendo complementos." (Anush Euredjian Chiappe; Juan Cortés Maiques,2009)[2].

Visual Studio:

Es un IDE de Microsoft y sólo usable en Windows. Es un entorno de programación de C++, C#, F# y por último Visual Basic.

Emulador jcpsp:

Es un emulador de PSP programando en Java para el PC. Permite ejecutar una grande parte de los ejecutables de PSP sin grandes problemas (aprox. 200 juegos). No depende de ningún sistema operativo y es OS. En algunas ocasiones el emulador no está capaz arrancar o ejecutar el "Homebrew". Por ejemplo la programación de mostrar el teclado original de PSP. El OSK original no es visible en el emulador. Sin embargo es una herramienta potente para realizar pruebas de programación en vez de conectar la PSP con el PC, transmitir los ficheros a través de USB a la PSP.

RemoteJoyLite v0.19

Es una aplicación que permite grabar la señal de salida de la pantalla de PSP a través de un USB en un ordenador, que tiene instalado PSP Type B Drivers. Es OS. Posee muchos problemas con la grabación de vídeo. Sólo transmite la información de vídeo a través de USB, no de audio. Para grabar el audio hace falta tener un cable (3.5mm male to male cable).

Notepad++

Es un editor de texto al estilo bloc de notas de Windows y es OS. Soporta varios lenguajes de programación. Solo disponible para Windows. Se distribuye bajos términos de la Licencia Pública General de GNU. Da la posibilidad guardar ficheros de texto sin extensión, por ejemplo para escribir un fichero Makefile muy importante.

4.2 Problemas con Code::Blocks

Como en los objetivos del proyecto comentado, la versión alfa del diccionario fue programada con Code::Blocks en C. Pero en la versión alfa había un problema de compilación en el caso de programación con las librerías de SDL. La solución en este momento fue, modificar el proyecto de CB de "Metro Valencia" de Anush Euredjian Chiappe y Juan Cortés Maiques[2] o programar sin IDE, ya que el símbolo del sistema de Windows fue modificado para reconocer la orden **make**, para poder generar ejecutables. Se supone que en la documentación de modificación de CB, en el caso de usar librerías SDL, falta un detalle importante. Sin esta información, el compilador de CB lanza errores de no reconocer las variables ni las funciones de SDL. Sin embargo en el Makefile las librerías están puestas. No es imposible programar con CB en C, porque los ejemplos sin SDL (hola mundo, acceso a los controles, acceso a datos de la PSP, WiFi) se pueden compilar sin problema en CB.

La versión actual fue programada en C++, por las ventajas de C++ y para poder usar el ejemplo de "TinyXML", que está escrito en C++. Como en el caso de C, las modificaciones de CB fueron los mismos.

El primero paso para poder compilar códigos de PSP en CB consiste en asignar a CB, donde están los "include" y las librerías :

```
Search Directories → Compiler
C:\pspsdk\include
C:\pspsdk\psp\include
C:\pspsdk\psp\sdk\include

Search Directories -→ Linker:
C:\pspsdk\lib
C:\pspsdk\psp\lib
C:\pspsdk\psp\sdk\lib
```

Ahora interviene el paso de "toolchain":

```
Pestaña →  Toolchain directories:
Compiler's installation directory: C:\pspsdk\

in programm Files:
C compiler: psp-gcc.exe
C++ compiler: psp-g++.exe
Linker for dynamic libs: psp-g++.exe
Linker for static libs: psp-ar.exe
Debuger: psp-gdb.exe
```

Resource compiler: -/-
Make program: make.ex

Press OK

Crear nuevo proyecto:

File→New→Project, select Empty Project
escribe el nombre del "proyect".
Pincha en next.
En la ventana siguiente, hay que selecionar lo siguiente:
[X] Create "Debug" configuration
[X] Create Release configuration
En "Release" , hay que cambiar "Objects output dir into" a "."
Pincha en Finish.

Para crear el ejecutable:

Project→Properties→Build Targets.
En " target options":
Debug and Release!

cambia output filename: .\X.elf <------- X = nombre del proyecto!
desactivar la opción []auto-generate filename extension
cambiar executing working dir: "."
cambia output dir: "."\

Por último la configuración del "Build":

Project→Build Options
Pestañana →Linker Setting→Other Linker options: -
lpsplibc -lpspdebug -lpspdisplay -lpspge -lpspctrl -lpspsdk -lpspnet -lpspnet_inet -lpspnet_apctl -lpspnet_resolver -lpsputility -lpspuser –lpspkernel

Con estas configuraciones es posibles programar un "Homebrew" en C **sin SDL** en CB. Es posible configura CB con un script, p.ej. para que abra después de compilar el emulador. En el emulador solo hay que seleccionar el "EBOOT.PBP", que se quiere ejecutar. En la

Figura 21, se puede ver el resultado de la configuración en CB en el caso de C y de un script (abrir el Emulador después de compilar el proyecto).

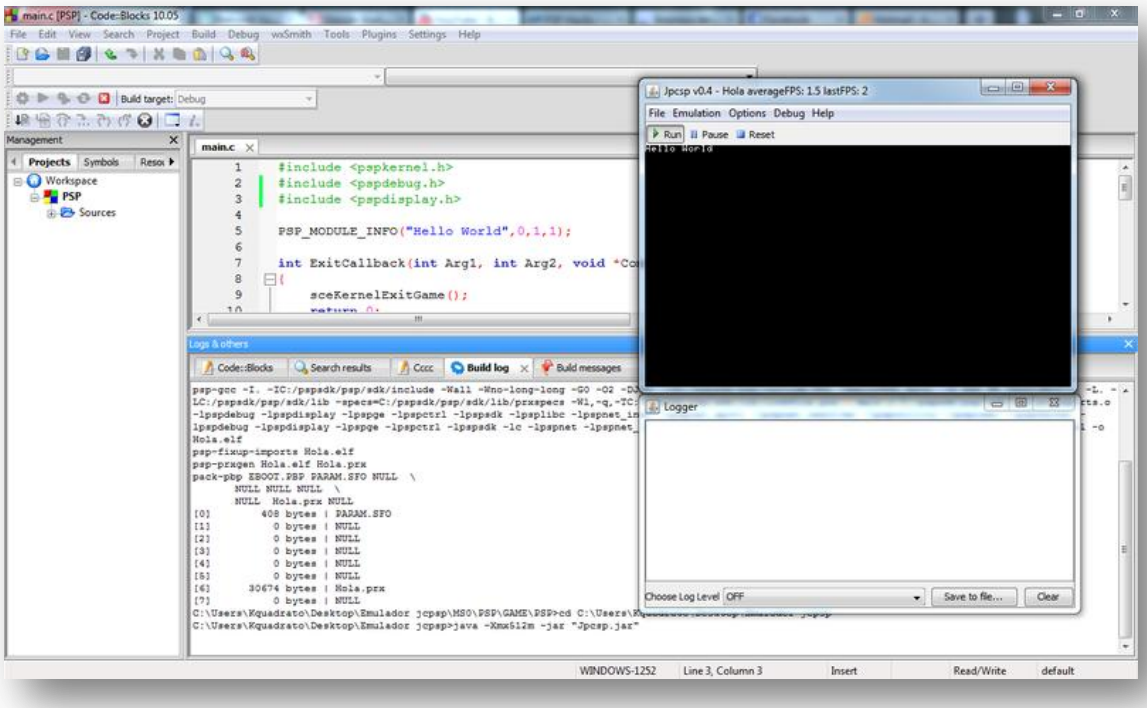


Figura 21: Configuracion de CB en C con éxito.

Respecto a la **Figura 21**, el resultado de la compilación es el siguiente:

```
---------- Build: Debug in PSP ----------
Compiling: main.c
Linking console executable: .\PSP.elf
Output size is 105.75 KB
Running target post-build steps
jpcsp.bat
C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\PSP>make
psp-gcc -I. -IC:/pspsdk/psp/sdk/include -Wall -Wno-long-long -G0 -O2 -DJOY_ -
IC:/pspsdk/psp/include/SDL -Dmain=SDL_main -D_PSP_FW_VERSION=371 -L. -
LC:/pspsdk/psp/sdk/lib -specs=C:/pspsdk/psp/sdk/lib/prxspecs -Wl,-q,-
TC:/pspsdk/psp/sdk/lib/linkfile.prx main.o C:/pspsdk/psp/sdk/lib/prxexports.o -
lpspdebug -lpspdisplay -lpspge -lpspctrl -lpspsdk -lpsplibc -lpspnet_inet -lpspnet_apctl -
lpspnet_resolver -lpsputility -lpspuser -lpspkernel -lpspdebug -lpspdisplay -lpspge -
lpspctrl -lpspsdk -lc -lpspnet -lpspnet_inet -lpspnet_apctl -lpspnet_resolver -lpsputility -
lpspuser -lpspkernel -o Hola.elf
psp-fixup-imports Hola.elf
psp-prxgen Hola.elf Hola.prx
```

```
pack-pbp EBOOT.PBP PARAM.SFO NULL \
NULL NULL NULL \
NULL Hola.prx NULL
[0] 408 bytes | PARAM.SFO
[1] 0 bytes | NULL
[2] 0 bytes | NULL
[3] 0 bytes | NULL
[4] 0 bytes | NULL
[5] 0 bytes | NULL
[6] 30674 bytes | Hola.prx
[7] 0 bytes | NULL
C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\PSP>cd
C:\Users\Kquadrato\Desktop\Emulador jcpsp
C:\Users\Kquadrato\Desktop\Emulador jcpsp>java -Xmx512m -jar "Jpcsp.jar"
Process terminated with status 0 (0 minutes, 9 seconds)
0 errors, 0 warnings
```

Sin embargo, ocurrió el problema de compilación desde el principio. Como antes comentado, para comprobar si la modificación de un IDE fue un éxito o no, se compila el ejemplo de "hola mundo". En el caso de programación en C++, la modificación de CB no fue un éxito.

En la **Figura 22**, se muestra la salida del compilador de CB en el caso de "hola mundo" de en C++

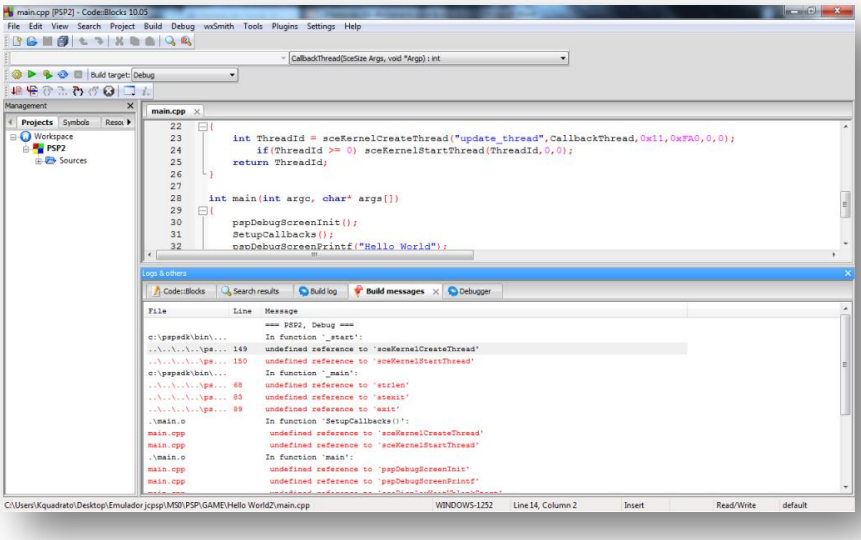


Figura 22: "Hola mundo" en C++ con CB.

Los errores de compilación en C++ son los siguientes:

```
C:\pspsdk\psp\lib\libc.a(_exit.o)||In function `_exit'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|900|undefined reference to
`sceKernelSelfStopUnloadModule'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|907|undefined reference to
`sceKernelExitThread'|
C:\pspsdk\psp\lib\libc.a(_sbrk.o)||In function `__psp_free_heap'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|576|undefined reference to
`sceKernelFreePartitionMemory'|
C:\pspsdk\psp\lib\libc.a(_sbrk.o)||In function `_sbrk'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|549|undefined reference to
`sceKernelMaxFreeMemSize'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|553|undefined reference to
`sceKernelAllocPartitionMemory'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|555|undefined reference to
`sceKernelGetBlockHeadAddr'|
C:\pspsdk\psp\lib\libc.a(_write.o)||In function `_write'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|308|undefined reference to
`sceIoWrite'|
C:\pspsdk\psp\lib\libc.a(fdman.o)||In function `__psp_fdman_init'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\fdman.c|29|undefined reference to
`sceKernelStdin'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\fdman.c|35|undefined reference to
`sceKernelStdout'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\fdman.c|41|undefined reference to
`sceKernelStderr'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function
`__psp_pipe_nonblocking_write'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|293|undefined reference to
`sceKernelTrySendMsgPipe'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function `__psp_pipe_write'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|262|undefined reference to
`sceKernelSendMsgPipe'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function `__psp_pipe_read'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
```

```
1.18.0\newlib\libc\sys\psp\pipe.c|218|undefined reference to
`sceKernelReceiveMsgPipe'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function `__psp_pipe_close'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|104|undefined reference to
`sceKernelDeleteMsgPipe'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function `pipe'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|54|undefined reference to
`sceKernelCreateMsgPipe'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|74|undefined reference to
`sceKernelDeleteMsgPipe'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function `__psp_pipe_peekmsgsize'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|38|undefined reference to
`sceKernelReferMsgPipeStatus'|
C:\pspsdk\psp\lib\libc.a(pipe.o)||In function
`__psp_pipe_nonblocking_read'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\pipe.c|156|undefined reference to
`sceKernelTryReceiveMsgPipe'|
C:\pspsdk\psp\lib\libc.a(chdir.o)||In function `chdir'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|104|undefined reference to
`sceIoDopen'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|109|undefined reference to
`sceIoDclose'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|111|undefined reference to
`sceIoChdir'|
C:\pspsdk\psp\lib\libc.a(_close.o)||In function `_close'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|236|undefined reference to
`sceIoClose'|
C:\pspsdk\psp\lib\libc.a(_fstat.o)||In function `_fstat'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|606|undefined reference to
`sceIoLseek'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|608|undefined reference to
`sceIoLseek'|
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|609|undefined reference to
`sceIoLseek'|
C:\pspsdk\psp\lib\libc.a(_lseek.o)||In function `_lseek'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|345|undefined reference to
`sceIoLseek'|
```

```
C:\pspsdk\psp\lib\libc.a(_read.o)||In function `_read'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|271|undefined reference to
`sceIoRead'|
C:\pspsdk\psp\lib\libc.a(_stat.o)||In function `_stat'
C:\msys\home\jetdrone\minpspw\psp\build\newlib-
1.18.0\psp\newlib\libc\sys\psp\..\..\..\..\..\..\..\newlib-
1.18.0\newlib\libc\sys\psp\libcglue.c|678|undefined reference to
`sceIoGetstat'|
||=== Build finished: 28 errors, 0 warnings ===|
```

Aparte de esta configuración, se ha añadido también las librerías en la opción "Project→Build Options→Linker Setting", como se muestra en la **Figura 23**.

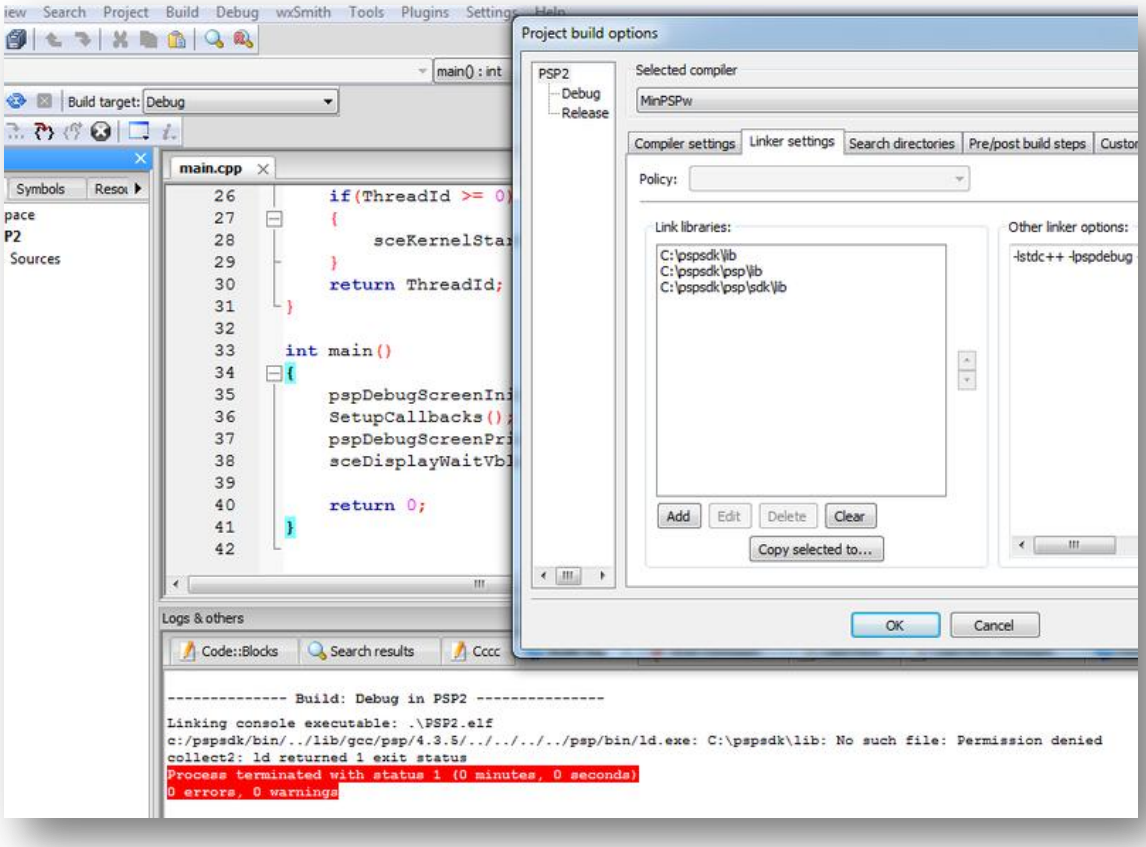


Figura 23: Modificación de CB en el caso de C++.

A causa del problema con CB, el estudio llevaba un retraso de varios meses. Sin embargo, existe una manera de compilar el código fuente en CB. No muy recomendable, porque se crean ficheros corruptos o mejor dicho "ejecutables corruptos", excepto en el

caso de "hola mundo" en C++. El código de la prueba "SDL imágenes" fue el primer caso donde se descubrió los "ejecutables corruptos". La única conclusión que se debe hacer en este caso especial. Debe haber algún fallo de configuración en CB, porque se puede compilar todos los ejemplos sin problema en el símbolo del sistema de Windows 7 o en VS. **Figura 24** y **Figura 25** muestran el ejecutable corrupto.

La manera fue la siguiente:

1. Compilar el código fuente en Code::Blocks
2. Abrir el símbolo del sistema, volver a compilar el código fuente en el símbolo del sistema **SIN CERRAR** Code::Bocks
3. Volver a compilar en Code::Blocks, si hace falta (p.ej. en "hola mundo" solo los pasos 1 y 2 fueron necesarios
4. Ahora se ha creado el ejecutable corrupto, que no muestra nada por pantalla ni en el emulador ni en la videoconsola.

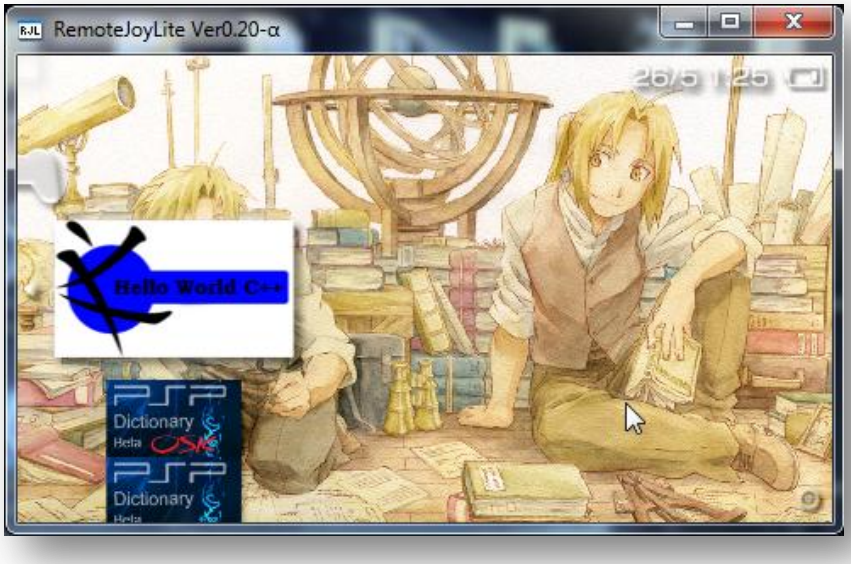


Figura 24: Menú XMB de la PSP con el "ejecutable corrupto".

Cuando se ejecuta el "ejecutable corrupto", la PSP se apaga después de unos segundos. Sin embargo en el emulador la aplicación se queda en el modo de ejecución. Los valores de imágenes por segundo están cambiando.

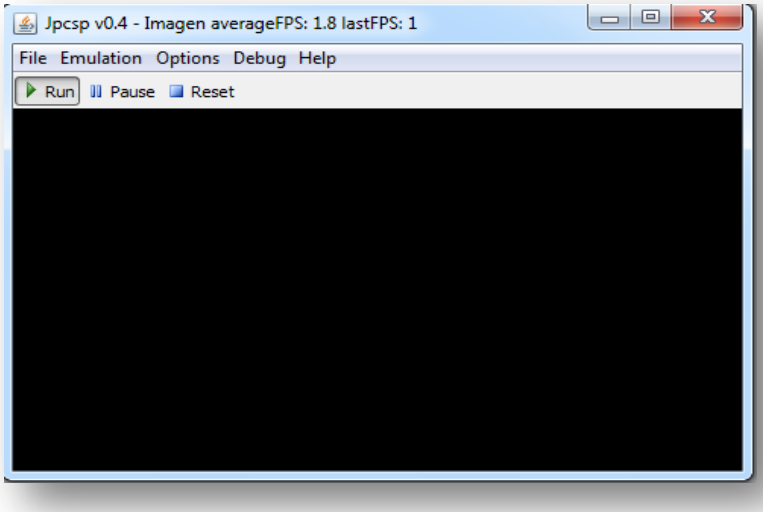


Figura 25: Ejemplo de ejecución de un "ejecutable corrupto".

No fue posible resolver el problema de compilación. Varios intentos y tutoriales[16] fueron revisados sin éxito. Siempre ocurría el mismo fallo de compilador. Antes de cambiar el IDE, el proyecto fue programado con notepad++ y el símbolo del sistemas de Windows7. En la comunidad de programación de PSP[3], los usuarios recomiendan usar como IDE: VS, Eclipse o Dev C++, porque el problema de compilador de CB es muy conocido en la comunidad.

[16] Tutorial de como configurar CB para DevikitPro: http://niozero.frostdisk.com/2008/08/tutorial-codeblocks-devkitpro.html

Tutorial como configurar CB para MinPSPw en C++ : http://forums.dashhacks.com/f141/how-to-setup-minpspw-with-codeblocks-t244336/

4.3 Pruebas en C++

Antes de empezar a programar el diccionario fue menester hacer unas pruebas en C++ para saber y aprender las diferencias de ambos lenguajes de programación. Para las pruebas de SDL imagen, SDL audio, se ha utilizado los ejemplos en C de Anush Euredjian Chiappe y Juan Cortés Maiques[2]. Para poder usar los códigos escrito en C en C++, es necesario realizar un par de modificaciones como por ejemplo cambiar la estructura del propio main y borrar la instrucción, que indica el tipo de modulo del "Homebrew" (en el caso de SDL). Otro ejemplo de cambio de código consiste en añadir la instrucción "SleepThread" o "GameExit", para evitar interbloques de hilos.

C++ dispone de la oportunidad de crear clases con objetos, para dividir el problema en varias partes sobre varios ficheros (la idea de OOP y técnica de divide y vencerás). Además dispone la posibilidad de sobrecarga de métodos como Java, encapsulación de variables (consultor – get / modificador - set). Por último, C++ posee una colección muy grande de librerías p.ej. librerías de XML o template "map".

OJO: En este estudio es importante tener un conocimiento básico de programación con clases en Java o mejor en C++. No es parte de este estudio enseñar todos los conceptos de programación con clases, si no de enseñar un ejemplo de programación con clase. En la programación hay muchas maneras de resolver un problema. Así, es difícil decir cuál es la solución correcta.

En casi todos los foros siempre pone: "Tutorial de programación de C y C++ para la PSP"[3]. El primero fallo que está cometiendo por la mayoría de usurarios. No tendría que dejar de mencionar que C++ siempre requiera la librería C++ runtime library en el Makefile, si no pueden ocurrir muchos errores de enlazador en tiempo de compilación.

```
• LIBS = -lstdc++
```

Adicionalmente, en el caso de C++, antes de salir de la aplicación es importante usar la llamada "SleepThread" Si no la PSP cierra la aplicación y se ha quedado bloqueado antes de volver a la XMB. Por el interbloque de hilos se la apaga después de unos segundos:

```
• sceKernelSleepThread();
```

La mayoría de funciones fueron escritas en C. Para poder usarlas en C++ hay que indicar al compilador de C++ que las siguientes funciones son de C y se requiere un cambio de compilador (de C++ a C). Esto se lo realiza con la instrucción 'extern "C" (nombre de función)' en el código fuente. Para saber si una función fue compilada en C o C++, hay que analizar el fichero header (*.h).

Si dentro del fichero header se encuentra la siguiente estructura:

```
#ifdef __cplusplus
extern "C"
{
#endif
  ... // Aquí está escrita la declaración de la función
#ifdef __cplusplus
}
#endif
```

Indica que es obligatorio hacer la transformación:

- extern "C" (nombre de función)

O según la necesidad de transformar varios headers o funciones

- extern "C" {conjunto de funciones / headers}

Aparte de esto, es importante entender los mensajes del compilador. Si ocurre un mensaje de 'undefined reference to' significa que falten librerías en el Makefile o que hay que modificar el código fuente como por ejemplo en SDL. En la mayoría de casos, falta una librería en el Makefile. Además el orden de las librerías es importante. Si librería X necesita librería Y, entonces librería Y viene después de librería X.

- LIBS = -lX –lY

En el caso si ocurre un mensaje de 'multiple definition of' significa que ya existe la definición en una librería. Para el caso de SDL, la solución consiste en borrar la instrucción.

SDL

En SDL se encuentra dicha estructura en los headers. Sin hacer la transformación, ocurre un mensaje de compilador de "undefined reference to 'SDL_main'", que se ha comentado antes:

```
1>  C:/pspsdk/psp/lib\libSDLmain.a(SDL_psp_main.o): In function `main':
1>
C:\msys\home\jetdrone\minpspw\devpaks\017_SDL\build\SDL\src\main/psp/SDL_psp_main.c
(86) : undefined reference to `SDL_main'
```

Asimismo, ocurre otro mensaje de compilador de "multiple definition of 'module_info'":

```
1>  C:/pspsdk/psp/lib\libSDLmain.a(SDL_psp_main.o):(.rodata.sceModuleInfo+0x0):
multiple definition of `module_info'
1>  main.o:(.rodata.sceModuleInfo+0x0): first defined here
1>  collect2: ld returned 1 exit status
```

Para evitar el fallo de "undefined reference to 'SDL_main", es obligatorio escribir el main de la siguiente manera en su código:

```
• extern "C" int SDL_main (int argc, char* args[]);
• int main(int argc, char *args[]) {…}
```

Para evitar el fallo de "multiple definition of 'module_info'", es obligatorio borrar la instrucción de modul_info en su código:

```
• PSP_MODULE_INFO("Hello World",0,1,1);
```

Los ejecutables:

El EBOOT.PBP de USBTest en C++ mide 64 692 Bytes = 63.17578125 Kbytes ≅ 63.16 Kbytes. El mismo EBOOT.PBP en C mide lo mismo, entonces 63,16 Kbytes. En la **Figura 26** se puede ver el tamaño del EBOOT.PBP en ambos casos.

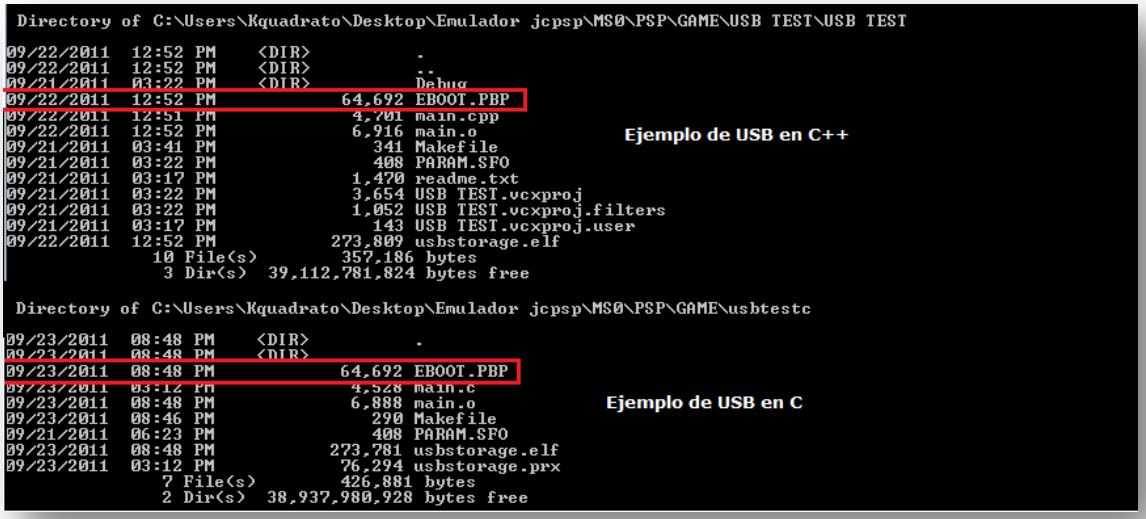


Figura 26: Diferencia del tamaño de EBOOT.PBP entre C y C++.

Pero el tamaño del ejecutable en ambos caso se cambia si se escribe dentro del Makefile la instrucción **BUILD_PRX = 1** para crear también un PRX. En este caso, el EBOOT.PBP es de 76 742 Bytes = 74.943359375 Kbytes ≅ 74,94 Kbytes. Existe una diferencia de 12 250 Bytes = 11.962890625 Kbytes ≅ 11,96 Kbytes. En la **Figura 27** se puede ver el tamaño del EBOOT.PBP en el caso sin PRX y con PRX.

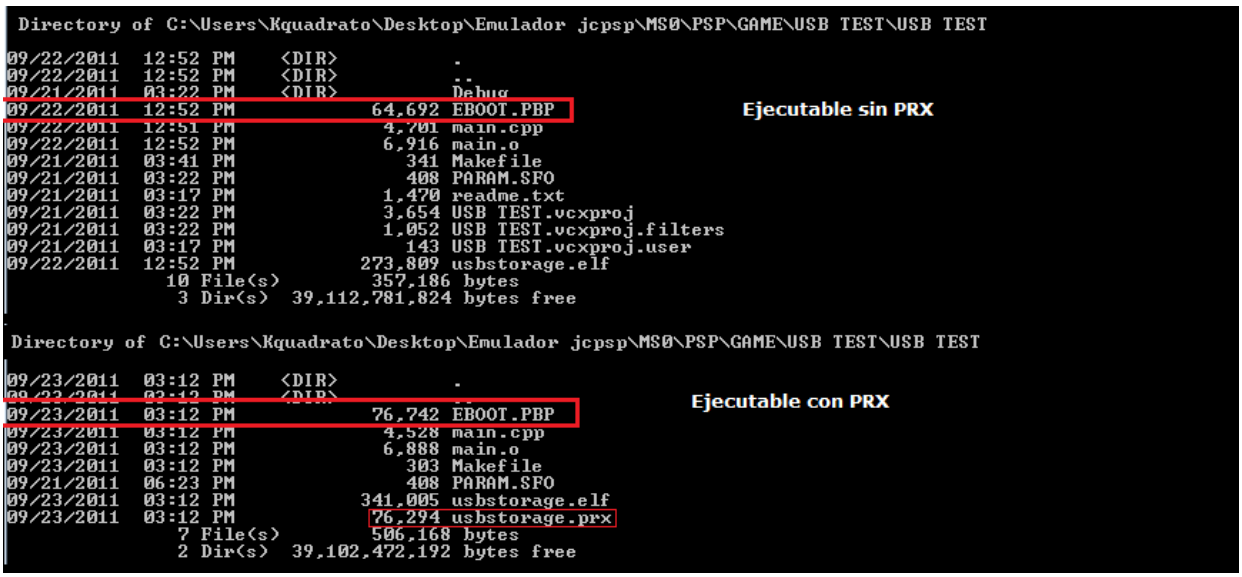


Figura 27: Diferencia entre ejecutable con o sin PRX.

4.3.1 Makefile

El Makefile es necesario para compilar todo el proyecto en una pasada y crear directamente el EBOOT.PBP o el PRX. Para compilar un proyecto o un fichero, el enlazador tiene que saber los "Flags" y las librerías. El Malefile tiene la siguiente estructura si no se usa SDL y no se quiere mostrar unas imágenes en el XMB (icono, fuente)

```
TARGET = nombredelproyecto
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin

OBJS = nombredelfichero.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS =  -librerias

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = titulo del EBOOT.PBP en el XMB

include $(PSPSDK)/lib/build.mak
```

Para un proyecto con varios ficheros o con varias clases, en OBJS se debe poner todos los objetos que se crea. En el diccionario se ha creado los siguientes objetos.

```
OBJS = main.o CSurface.o CMenuSurface.o CMenu.o CKeyboardSurface.o CKeyboard.o CAudio.o
CAnimation.o
```

Cuando se quiere programar con SDL, las librerías son los siguientes en el caso de "imagen" y aparece un nuevo "flag":

```
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)

LIBS =  -lSDL_image -lpng -ljpeg -lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lstdc++ -lpspirkeyb –
lpsppower
```

Si se tiene interés en mostrar una imagen en el XMB o poner sonido los siguientes elementos son necesarios en el Makefile. El nombre de las variables no se cambia. Los ficheros tienen que llamarse así.

```
PSP_EBOOT_ICON = ICON0.png
PSP_EBOOT_UNKPNG = PIC0.png
PSP_EBOOT_PIC1 = PIC1.png
PSP_EBOOT_SND0= SND0.at3
```

- ICON0.png, es el icono que sale en el XMB con la resolución tiene que ser de 141 x 80 pixeles
- PIC1.png, es la imagen del fondo de la aplicación en el XMB con 310 x 180 pixeles
- PIC0.png, es la imagen transparente frontal del fondo de la aplicación en el XMB. Se la usa mucho para mostrar texto. Es importante tener la imagen transparente con la resolución de 310 x 180 pixeles
- SND0.at3, es el audio del formato "Atrac" que se reproduce en el XMB cuando se muestra la aplicación en el menú de XMB. Cómo se puede crear una pista de audio en un bucle infinito en el XMB se muestra en el apéndice.

4.3.2 "SDL imagen"

En este subcapítulo se aprende como mostrar una imagen con ayuda de SDL en C++. Es una base para crear una "Homebrew" para PSP o un juego o aplicación para PC.

En la **Figura 28**, se muestra un ejemplo de reproducir una imagen en la PSP

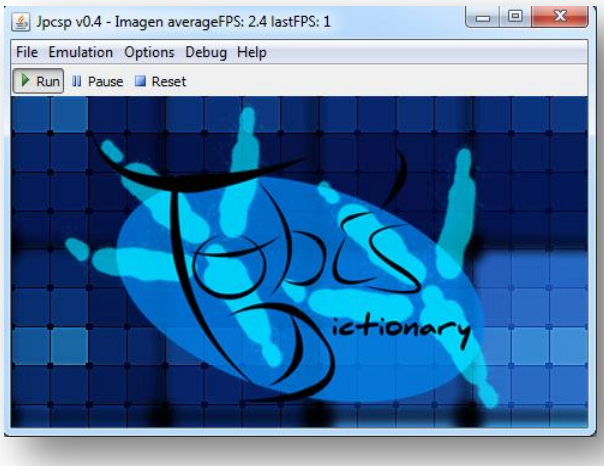


Figura 28: Captura de pantalla del emulador que está ejecutando "SDL imagen" - la versión C++.

Como antes comentado, en el caso de C++ con SDL, es obligatorio siembre hacer estas modificaciones., si no ocurren dos errores de "undefined reference to SDL_main" y "multiple definition of modul_info". Entonces la modificación de este código consiste en los siguientes pasos:

- Borrar la instrucción de model_info:
  - PSP_MODULE_INFO("imagenes",0,1,0);
- Modificar el main:
  - extern "C" int SDL_main (int argc, char* args[]);
  - int main(int argc, char *args[]) {…}
- Añadir un sistema de control de ficheros

El código original en C de SDL imagen:

```
#include <pspkernel.h>
#include <SDL/sdl.h>
#include <SDL/SDL_image.h>

PSP_MODULE_INFO("imagenes", 0, 1, 0);
PSP_HEAP_SIZE_KB(20480);

int exit_callback(int arg1, int arg2, void *common) {
   sceKernelExitGame();
   return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
   int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
   sceKernelRegisterExitCallback(cbid);
   sceKernelSleepThreadCB();
   return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
     int thid = sceKernelCreateThread("update_thread", CallbackThread, 0x11, 0xFA0, 0, 0);
   if(thid >= 0) { sceKernelStartThread(thid, 0, 0); }
     return thid;
}

int main()
{
SetupCallbacks();
SDL_Surface *PantallaV;
SDL_Init(SDL_INIT_VIDEO);
SDL_ShowCursor(0);
PantallaV = SDL_SetVideoMode(480, 272, 32,SDL_HWSURFACE|SDL_DOUBLEBUF);
SDL_Surface *imagen = IMG_Load("fondo.jpg");
SDL_BlitSurface(imagen,NULL,PantallaV,NULL);
SDL_Flip(PantallaV);
SDL_FreeSurface(imagen);

return 0;
}
```

Asimismo, es obligatorio siempre comprobar si el fichero existe o no. Si no se hace esta comprobación, la PSP empieza a buscar dicho fichero en la memoria y por no poder encontrarlo se apaga de seguido. Para avisar al usuario qué ha ocurrido un fallo, se hace con un mensaje por pantalla por la consola. En "hola mundo.cpp" se ha ultilizado la consola de PSP para escribir "hola mundo". La palabra mágica para hacerlo es "pspDebugScreenPrintf". Antes de llamar dicha función, hay que vaciar la pantalla de la consola con "pspDebugScreen".

La función "SDL_Init" devuelve un valor, si este valor es distinto de cero, indica que no fue posible inicializar SDL.

```
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0)
                {       pspDebugScreenInit();
                        pspDebugScreenPrintf("unable to initialize SDL");
                }
```

La función "IMG_Load" también devuelve un valor, en este caso es la dirección donde está situado el fichero. Si el vector de "IMG_Load" está vacía, quiere decir, que el fichero no existe o que ha ocurrido un fallo de cargar de la imagen en la memoria

```
SDL_Surface *image = IMG_Load("intro.png");
if(image == NULL)
{
    pspDebugScreenInit();
    printf("unable to load image");
}
```

El código fuente de "SDL imagen" en C++:

```
#include <pspkernel.h>
#include <SDL/sdl.h>
#include <SDL/SDL_image.h>

#define printf pspDebugScreenPrintf

PSP_HEAP_SIZE_KB(20480);

int ExitCallback(int Arg1, int Arg2, void *Common)
{
   sceKernelExitGame();
   return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
   int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
   sceKernelRegisterExitCallback(CallbackId);
   sceKernelSleepThreadCB();
   return 0;
}

int SetupCallbacks(void)
{
   int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
      if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
   return ThreadId;
}

extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
   SetupCallbacks();

   SDL_Surface *screen;
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0)
                {          pspDebugScreenInit();
                           pspDebugScreenPrintf("unable to initialize SDL");
                }
   SDL_ShowCursor(0);

   screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);

   SDL_Surface *image = IMG_Load("intro.png");

   SDL_BlitSurface(image,NULL,screen,NULL);
   SDL_Flip(screen);
   SDL_FreeSurface(image);
     if(image == NULL)
     {
                pspDebugScreenInit();
                printf("unable to load image");
     }

   sceKernelSleepThread();

return 0;
}
```

Y su Makefile respectivo:

```
TARGET = Imagen
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin

OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS =  -lSDL_image -lpng -ljpeg -lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lstdc++ -lpspirkeyb –
lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = Imagen

include $(PSPSDK)/lib/build.mak
```

4.3.3 "SDL audio"

En este ejemplo se aprende como reproducir audio y mostrar una imagen por pantalla en el mismo tiempo. Ambas partes forman parte de una base de una AM para PSP o PC.

OJO: Dependiendo de el SDK MinPSPw, pueden ocurrir muchos fallos de compilación en el caso de SDL música, porqué existen dos librerías de mikmod en este SDK. La segunda sobrescribe la primera la cual SDL necesita. La solución, instalar una versión más antigua[17].

En la **Figura 28**, se muestra un ejemplo de reproducir una imagen en la PSP.

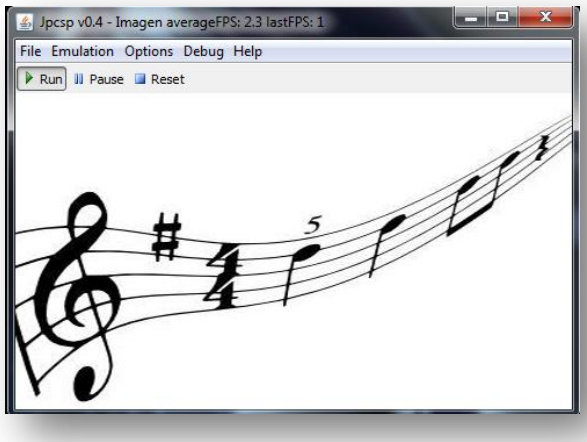


Figura 29: Captura de pantalla del emulador que está ejecutando "SDL audio" - la versión C++.

[17] Todos los fallos que occuriren y la conclusión se puede ver en mi blog: http://psppproblemsdashboard.blogspot.com/2011/06/psp-audio-linking-problem.html#comments

De nuevo hay que hacer las modificaciones para poder usar el código en C++ correctamente.

- Borrar la instrucción de model_info:
  - PSP_MODULE_INFO("imagenes",0,1,0);
- Modificar el main:
  - extern "C" int SDL_main (int argc, char* args[]);
  - int main(int argc, char *args[]) {…}
- Añadir un sisteme de control de ficheros

El código original en C de SDL audio:

```
#include <pspkernel.h>
#include <SDL/sdl.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>

PSP_MODULE_INFO("audio", 0, 1, 0);
PSP_HEAP_SIZE_KB(20480);

int exit_callback(int arg1, int arg2, void *common) {
    sceKernelExitGame();
    return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
    int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
    sceKernelRegisterExitCallback(cbid);
    sceKernelSleepThreadCB();
    return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
      int thid = sceKernelCreateThread("update_thread", CallbackThread, 0x11, 0xFA0, 0, 0);
    if(thid >= 0) { sceKernelStartThread(thid, 0, 0); }
      return thid;
}

int main()
{
SetupCallbacks();
SDL_Surface *PantallaV;
SDL_Event event;

//AUDIO
Mix_Music *bgm;
int music_chanel;

//AUDIO
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
SDL_ShowCursor(0);
PantallaV = SDL_SetVideoMode(480, 272, 32,SDL_HWSURFACE|SDL_DOUBLEBUF);
//AUDIO
 Mix_OpenAudio(44100, AUDIO_S16SYS, 2, 4096);

SDL_Surface *imagen = IMG_Load("fondo.jpg");
SDL_BlitSurface(imagen,NULL,PantallaV,NULL);
```

```
SDL_Flip(PantallaV);
SDL_FreeSurface(imagen);

//AUDIO
bgm = Mix_LoadMUS("audio.ogg");
music_chanel = Mix_PlayMusic(bgm,-1);
SDL_FreeSurface(PantallaV);
Mix_CloseAudio();
return 0;
}
```

Ahora se trabaja adicionalmente con un fichero de audio. Entonces, es obligatorio comprobar si el fichero de audio existe o no. Como en el caso de imágenes, las funciones devuelven un valor para saber si el fichero existe o no. Aparte de esto, hay que inicializar el audio. Esto indica otra comparación más para saber si fue posible inicializar el mixer de audio o no. Y por último puede ocurrir el caso que se ha podido inicializar el mixer, cargar el audio en la memoria, pero no se ha podido reproducirlo. En este caso hay que hacer tres comprobaciones:

- ¿Fue posible inicializar audio mixer?
- ¿Fue posible cargar el audio?
- ¿Fue posible reproducir el audio?

```
if(Mix_OpenAudio(22050, AUDIO_S16SYS, 2, 4096) != 0)
        {       pspDebugScreenInit();
                pspDebugScreenPrintf("unable to initialize audio");
        }


Mix_Music *sound = Mix_LoadMUS("sound.mp3");
if(sound == NULL)
        {       pspDebugScreenInit();
                pspDebugScreenPrintf("unable to load audio");
        }

int channel = Mix_PlayMusic(sound, -1);
if(channel == -1)
        {       pspDebugScreenInit();
                pspDebugScreenPrintf("unable to play audio");
        }
```

El código fuente de “SDL audio” en C++:

```
#include <pspkernel.h>
#include <pspctrl.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>



PSP_HEAP_SIZE_KB(20480);

int ExitCallback(int Arg1, int Arg2, void *Common)
{
   sceKernelExitGame();
     return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
   int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
   sceKernelRegisterExitCallback(CallbackId);
   sceKernelSleepThreadCB();
     return 0;
}

int SetupCallbacks(void)
{
   int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
      if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
   return ThreadId;
}

extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
   SetupCallbacks();

   SDL_Surface *screen;
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0)
     {          pspDebugScreenInit();
                pspDebugScreenPrintf("unable to initialize SDL");
     }

   SDL_ShowCursor(0);

   screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);

   SDL_Surface *image = IMG_Load("intro.png");
     SDL_BlitSurface(image,NULL,screen,NULL);
     SDL_Flip(screen);
     SDL_FreeSurface(image);
     if(image == NULL)
               {          pspDebugScreenInit();
                          pspDebugScreenPrintf("unable to load image");
               }
```

```
	if(Mix_OpenAudio(22050, AUDIO_S16SYS, 2, 4096) != 0)
		{	pspDebugScreenInit();
			pspDebugScreenPrintf("unable to initialize audio");
		}


	Mix_Music *sound = Mix_LoadMUS("sound.mp3");
	if(sound == NULL)
		{	pspDebugScreenInit();
			pspDebugScreenPrintf("unable to load audio");
		}

	int channel = Mix_PlayMusic(sound, -1);
	if(channel == -1)
		{	pspDebugScreenInit();
			pspDebugScreenPrintf("unable to play audio");
		}


   sceKernelSleepThread();
}
```

Y su Makefile respectivo:

```
TARGET = Audio
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin
OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS = -lSDL_mixer -lsmpeg -lstdc++ -lSDL_ttf -lfreetype -lvorbisidec -logg -lSDL_image -lpng -ljpeg -
lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lpspirkeyb -lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = audio

include $(PSPSDK)/lib/build.mak
```

4.3.4 "SDL audio2"

Otro ejemplo más de audio para aprender cómo controlar la reproducción de una pista de audio con los botones de la PSP. En este ejemplo la reproducción de audio se activa con el botón ⊗ y se desactiva con el botón ◎ con ayuda de SDL en C++. En el código original también se ha mostrado un texto por pantalla con SDL. En este ejemplo no se lo hace, por que el tema de mostrar texto por pantalla es un poco complicado sin saber y entender cómo funciona el "blitting" correctamente.

El código original sin mostrar un texto por pantalla en C:

```
#include <pspkernel.h>
#include <SDL/sdl.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>
#include <pspctrl.h>


PSP_MODULE_INFO("audio", 0, 1, 0);
PSP_HEAP_SIZE_KB(20480);

int exit_callback(int arg1, int arg2, void *common) {
    sceKernelExitGame();
    return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
    int cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
    sceKernelRegisterExitCallback(cbid);
    sceKernelSleepThreadCB();
    return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
     int thid = sceKernelCreateThread("update_thread", CallbackThread, 0x11, 0xFA0, 0, 0);
    if(thid >= 0) { sceKernelStartThread(thid, 0, 0); }
     return thid;
}

int main()
{
SetupCallbacks();
SDL_Surface *PantallaV;
SDL_Event event;
SceCtrlData pad;

//FUENTE
SDL_Rect dest;

//AUDIO
Mix_Music *bgm;
int music_chanel;

//AUDIO
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO);
SDL_ShowCursor(0);
PantallaV = SDL_SetVideoMode(480, 272, 32,SDL_HWSURFACE|SDL_DOUBLEBUF);
//AUDIO
 Mix_OpenAudio(44100, AUDIO_S16SYS, 2, 4096);
SDL_Surface *imagen = IMG_Load("fondo.jpg");
SDL_BlitSurface(imagen,NULL,PantallaV,NULL);

//AUDIO
bgm = Mix_LoadMUS("audio.ogg");

SDL_BlitSurface(imagen, NULL, PantallaV, NULL);
SDL_Flip(PantallaV);
SDL_FreeSurface(imagen);

while(1){
      while(1)
   {
```

```
            sceCtrlReadBufferPositive(&pad, 1);
              if(pad.Buttons & PSP_CTRL_CROSS) {
                if(Mix_PausedMusic()){Mix_ResumeMusic();}
                else{music_chanel = Mix_PlayMusic(bgm,-1);}
                      break;
              }
    }

      while(1)
    {
           sceCtrlReadBufferPositive(&pad, 1);
            if(pad.Buttons & PSP_CTRL_CIRCLE) {
              Mix_PauseMusic();
                    break;
            }

    }}
 return 0;
 }
```

Como se ha visto antes, de nuevo hay que hacer las modificaciones para que el compilador lo compile sin problemas y para estabilizar la aplicación.

- Borrar la instrucción de model_info:
  - PSP_MODULE_INFO("imagenes",0,1,0);
- Modificar el main:
  - extern "C" int SDL_main (int argc, char* args[]);
  - int main(int argc, char *args[]) {…}
- Añadir un sistema de control de ficheros
  - ¿Fue posible inicializar?
  - ¿Fue posible cargar el fichero?
  - ¿Fue posible reproducir el fichero?

Todas las comparaciones que hay que hacer:

```
if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0)
    {       pspDebugScreenInit();
            pspDebugScreenPrintf("unable to initialize SDL");
    }

SDL_Surface *image = IMG_Load("intro.png");
if(image == NULL)
{
    pspDebugScreenInit();
    printf("unable to load image");
}

if(Mix_OpenAudio(22050, AUDIO_S16SYS, 2, 4096) != 0)
    {       pspDebugScreenInit();
            pspDebugScreenPrintf("unable to initialize audio");
    }
```

```
Mix_Music *sound = Mix_LoadMUS("sound.mp3");
if(sound == NULL)
	{		pspDebugScreenInit();
			pspDebugScreenPrintf("unable to load audio");
	}

int channel = Mix_PlayMusic(sound, -1);
if(channel == -1)
	{		pspDebugScreenInit();
			pspDebugScreenPrintf("unable to play audio");
	}
```

En el código original se encuentra un fallo fatal en la estructura. Para la comprobación del botón que fue pulsado, es necesario usar un único bucle infinito (while(1) o while(true)). Dentro del bucle "while" se realiza las pruebas de qué botón fue pulsado o mejor dicho, qué pasa si el botón X fue pulsado.

Bucle "while" del código original:

```
    while(1){
        while(1)
    {
        sceCtrlReadBufferPositive(&pad, 1);
            if(pad.Buttons & PSP_CTRL_CROSS) {
              if(Mix_PausedMusic()){Mix_ResumeMusic();}
              else{music_chanel = Mix_PlayMusic(bgm,-1);}
                  break;
            }
    }

        while(1)
    {
         sceCtrlReadBufferPositive(&pad, 1);
            if(pad.Buttons & PSP_CTRL_CIRCLE) {
              Mix_PauseMusic();
                  break;
            }
    }
}
```

El resultado del nuevo bucle "while":

```
        while(1)
        {
                sceCtrlReadBufferPositive(&pad, 1);

                        if(pad.Buttons & PSP_CTRL_CROSS)
                        {
                                if(Mix_PausedMusic())       Mix_ResumeMusic();

                                else{
                                        if ((music_chanel = Mix_PlayMusic(sound,-1)) == -1)
                                                {       pspDebugScreenInit();
                                                        pspDebugScreenPrintf("unable to play
audio");
                                                }
                                        }
                        }


                        if(pad.Buttons & PSP_CTRL_CIRCLE)
                        {
                                Mix_PauseMusic();
                        }
        }
```

El código fuente de “SDL audio2” en C++:

```
#include <pspkernel.h>
#include <pspctrl.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>



PSP_HEAP_SIZE_KB(20480);

int ExitCallback(int Arg1, int Arg2, void *Common)
{
   sceKernelExitGame();
     return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
   int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
   sceKernelRegisterExitCallback(CallbackId);
   sceKernelSleepThreadCB();
     return 0;
}

int SetupCallbacks(void)
{
   int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
      if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
   return ThreadId;
}

extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
   SetupCallbacks();

   SDL_Surface *screen;
     Mix_Music *sound;
     int music_chanel;

     SceCtrlData pad;

     if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0)
               {       pspDebugScreenInit();
                       pspDebugScreenPrintf("unable to initialize SDL");
               }

   SDL_ShowCursor(0);

   screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);

   SDL_Surface *image = IMG_Load("intro.png");
     SDL_BlitSurface(image,NULL,screen,NULL);
     SDL_Flip(screen);
     SDL_FreeSurface(image);
               if(image == NULL)
               {       pspDebugScreenInit();
                       pspDebugScreenPrintf("unable to load image");
               }


     if(Mix_OpenAudio(22050, AUDIO_S16SYS, 2, 4096) != 0)
```

```
		{	pspDebugScreenInit();
			pspDebugScreenPrintf("unable to initialize audio");
		}


	sound = Mix_LoadMUS("sound.mp3");
	if(sound == NULL)
		{	pspDebugScreenInit();
			pspDebugScreenPrintf("unable to load audio");
		}


	while(1)
	{
		sceCtrlReadBufferPositive(&pad, 1);

			if(pad.Buttons & PSP_CTRL_CROSS)
			{
				if(Mix_PausedMusic())	Mix_ResumeMusic();

				else{
					if ((music_chanel = Mix_PlayMusic(sound,-1)) == -1)
						{	pspDebugScreenInit();
							pspDebugScreenPrintf("unable to play
audio");
						}
				}
			}


			if(pad.Buttons & PSP_CTRL_CIRCLE)
			{
				Mix_PauseMusic();
			}
	}

   sceKernelSleepThread();
}
```

Y su Makefile:

```
TARGET = Audio
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin
OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS = -lSDL_mixer -lsmpeg -lstdc++ -lSDL_ttf -lfreetype -lvorbisidec -logg -lSDL_image -lpng -ljpeg -
lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lpspirkeyb -lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = audio
include $(PSPSDK)/lib/build.mak
```

4.3.5 "SDL animaciones"

El objetivo de este apartado es crear una animación con "sprites" con ayuda de SDL en C++. Es necesario tener una imagen que contenga varios frames de una animación. Aparte de esto, se aprender cómo funciona y cómo usar la función "SDL_BlitSurface" de SDL correctamente. Para crear un videojuego simple, por ejemplo de tipo "libro ilustrado" es muy importante entender el funcionamiento de "bliting".

OJO: Para crear un videojuego general en 2D, se debe estudiar también la parte de colusión de imágenes. La colusión no forma parte de este estudio. Para crear un videojuego en 3D, es necesario usar SDL con OpenGL.

La función "SDL_BlitSurface" de SDL requiere 4 parámetros (fuente de superficie, fuente de rectángulo de superficie, destino de superficie en la pantalla, destino de rectángulo de superficie en la pantalla).

En el primero caso de "bliting", es importante conocer la fuente de superficie y su destino. Los parámetros de rectángulos de superficie quedan con el valor "NULL", porque se muestra toda la imagen en el destino de superficie. Para facilitar, se substituye el nombre de "fuente de superficie" con la palabra "imagen" y también se substituye el nombre de "destino de superficie" con la palabra "pantalla". Antes de "bliting" de superficie, el programador indica la resolución de pantalla. En el caso de PSP la resolución de pantalla es "480 x 272". Como en los apartados anteriores, se "aprendió" como mostrar una imagen por pantalla sin leer o entender la teoría de "bliting".

Es importante no confundir el concepto de "bliting" por el mal uso de términos o lógica. Si no, se cometeran unos fallos muy tontos y muy peligros cuando crezca la aplicación.

El sistema de coordenadas es de la siguiente manera. Las coordenadas de X e Y empiezan por la esquina superior izquierda. La coordenada X crece hacia la derecha y la coordenada Y crece hacia la parte inferior de la pantalla.

En la **Figura 30** se muestra el sistema de coordenadas de la pantalla de la PSP

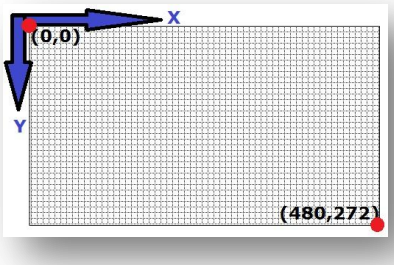


Figura 30: Ejemplo de sistemas de coordenadas de la pantalla de la PSP.

Ahora se quiere mostrar una imagen de la misma resolución en la pantalla de PSP. Entonces se carga la imagen en el buffer con la llamada de sistema de SDL "IMG_Load". La PSP todavía no muestra la imagen por pantalla, porque falta el "bliting" lo que realiza que la imagen se muestre por pantalla después de llamar la función "SDL_Flip(pantalla)".

Para entenderlo mejor se muestra la **Figura 31**.

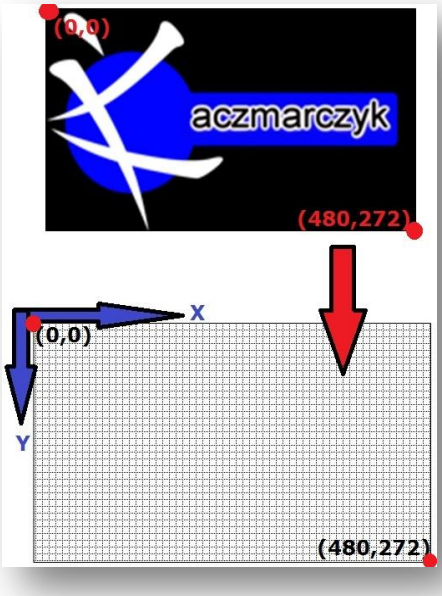


Figura 31: Ejemplo de "bliting" con imágenes de misma resolución de la pantalla.

El resultado de "bliting" en el primer caso será el siguiente, como se esperaba se muestra el resultado en la **Figura 32**.

Figura 32: Resultado de "bliting" de imagen con la misma resolución como la pantalla.

Como comentado, en el primer caso de "bliting" los valores de rectángulos quedan nulos (NULL), porque se copia toda la imagen en la pantalla. En el caso de que la imagen no sea de la misma resolución, se copia también toda la imagen a la pantalla. Pero por la pantalla se muestra la imagen y lo que queda "blanco" se muestra en negro (**Figura 33**).

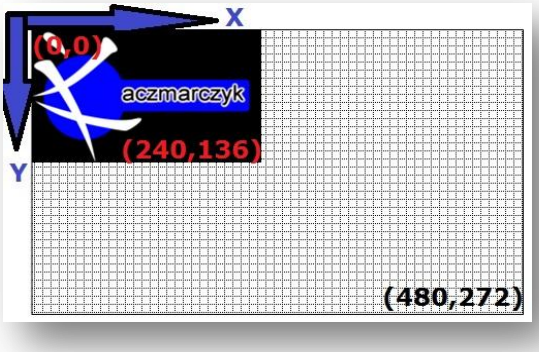


Figura 33: Resultado de "bliting" de imagen con distinta resolución como la pantalla.

El código fuente es el siguiente:

```
SDL_BlitSurface(imagen, NULL, pantalla, NULL);
```

En el segundo caso de "bliting", es necesario saber donde hay que realizar el "bliting" de la imagen, quiere decir, donde se cambian las coordenadas x e y de una imagen / superficie en la pantalla. Antes de mostrar el resultado de "bliting" en este caso, hay que explicar lo que es un rectángulo.

El rectángulo de superficie contiene las siguientes coordinadas x,y,h,w de una superficie. La coordinada h y w describen la altura (h) y el ancho (w) del rectángulo de superficie. Por ejemplo en el caso de mostrar una pieza de una imagen en la pantalla, es importante conocer las 4 coordenadas. Especialmente para el caso de realizar una animación con un "sprite".

En la **Figura 34** se puede ver el ejemplo de rectángulo.

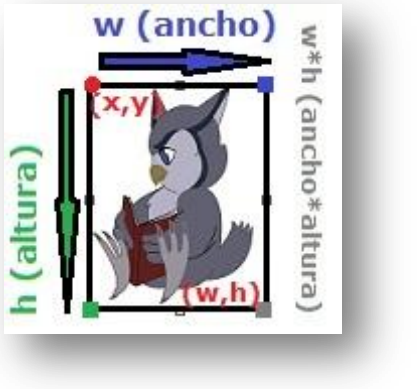


Figura 34: Ejemplo de rectángulo de superficie con sus parámetros (x,y,w,h).

Con el conocimiento de rectángulo de superficie, ya se es capaz de mostrar texto por pantalla en cualquier posición en la pantalla o/y también mostrar una imagen en cualquier posición en la pantalla. Para realizar esto, se indica las coordinadas X e Y de rectángulo de superficie en la pantalla e indicar el rectángulo de superficie de destino en la función "SDL_BlitSurface" (último parámetro).

Un ejemplo concreto, se ve en la **Figura 35**.

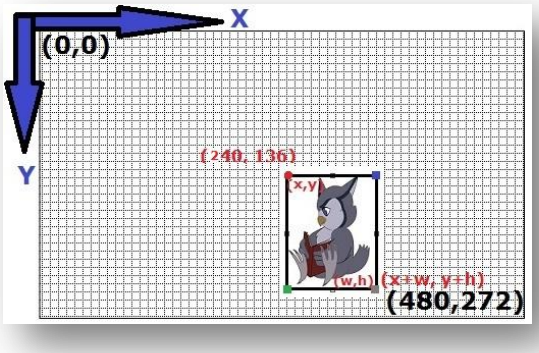


Figura 35: Resultado de "bliting" en el segundo caso.

El código fuente es el siguiente:

```
recDestino.x = 240;
recDestino.y = 136;
// recDestino.w = 96;          no es necesario si no se modifica
// recDestino.h = 96;          los parámetros w y h
SDL_BlitSurface(imagen, NULL, pantalla, & recDestino);
```

Ahora, se conoce el concepto de superficie y de rectángulo de superficie. Con la ayuda de este concepto se puede realizar el "clipping" (indicar al programa la fuente del rectángulo de superficie) de una imagen. Se necesita para crear una animación con un "sprite". Como se ha comentado antes, los "sprites" contienen varios frames en una imagen se muestra en la **Figure 36**:

Figure 36: Ejemplo de un "sprite" de "RPG Maker".

Para hacer el "clipping", fue programada una función que cambia las coordenadas. El ancho y altura no se cambian en ningún momento en la aplicación. Entonces hay que

manipular las coordenadas x e y (donde empieza el rectángulo de superficie en la imagen cargada), para elegir la imagen la cual se quiere mostrar por pantalla en el tiempo t.

El código de dicha función es el siguiente:

(Antes se ha declarado las variables SDL_Rect spriteSoruce y spriteDestination global).

```
void clippingSprite(int x, int y)
{
    spriteSoruce.x = x;
    spriteSoruce.y = y;
    spriteSoruce.w = 96;
    spriteSoruce.h = 96;
}
```

El último caso de "bliting". Se carga toda la imagen en el buffer de memoria y se muestra solo una pieza de imagen en la posición x e y por pantalla. En este caso, es necesario usar todos los parámetros de la función "SDL_BlitSurface". Para realizar una animación, hay que cambiar la imagen en tiempo t.

Antes de mostrar el código fuente final de dicha prueba, se debe saber que es posible hacer el "bliting" de una imagen a otra imagen, es decir, no es siempre necesario hacer el "bliting" de una imagen a la pantalla sino también de imagen i a imagen j y luego de imagen j a pantalla.

El código fuente:

```
#include <pspkernel.h>
#include <SDL/sdl.h>
#include <SDL/SDL_image.h>

#define printf pspDebugScreenPrintf

PSP_HEAP_SIZE_KB(20480);
    SDL_Surface *temp, *sprite, *bg;
    SDL_Rect spriteSoruce, spriteDestination;

int ExitCallback(int Arg1, int Arg2, void *Common)
{
   sceKernelExitGame();
   return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
   int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
   sceKernelRegisterExitCallback(CallbackId);
   sceKernelSleepThreadCB();
   return 0;
}

int SetupCallbacks(void)
{
   int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
      if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
   return ThreadId;
}

void clippingSprite(int x, int y)
{
    spriteSoruce.x = 0 + x;
    spriteSoruce.y = 0 + y;
    spriteSoruce.w = 96;
    spriteSoruce.h = 96;
}

extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
   SetupCallbacks();

   SDL_Init(SDL_INIT_VIDEO);
   SDL_ShowCursor(0);

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

    bool animation = true;


    bg = IMG_Load("bg.jpg");

    /*load sprite*/
    temp = IMG_Load("sprite.png");
    sprite = SDL_DisplayFormat(temp); //fast bliting?
    spriteDestination.x = 15;
    spriteDestination.y = 161;
    spriteDestination.w = 96;
    spriteDestination.h = 96;
```

```
while(animation)
{
    for(int i=1; i<9; i++)
    {
            switch(i)
            {
                    case 1:
                            {
                                    SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                    SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));

                                    clippingSprite(0,0);
                                    SDL_BlitSurface(bg, NULL, screen, NULL);
                                    SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                    SDL_Flip(screen);
                                    sceKernelDelayThread(0.4*1000*1000);
                                    break;
                            }

                    case 2:
                            {
                                    SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                    SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));
                                    clippingSprite(96,0);
                                    SDL_BlitSurface(bg, NULL, screen, NULL);
                                    SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                    SDL_Flip(screen);
                                    sceKernelDelayThread(0.4*1000*1000);
                                    break;
                            }

                    case 3:
                            {

                                    SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                    SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));
                                    clippingSprite(192,0);
                                    SDL_BlitSurface(bg, NULL, screen, NULL);
                                    SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                    SDL_Flip(screen);
                                    sceKernelDelayThread(0.4*1000*1000);
                                    break;
                            }

                    case 4:
                            {
                                    SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                    SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));

                                    clippingSprite(288,0);
                                    SDL_BlitSurface(bg, NULL, screen, NULL);
                                    SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
```

```
                                        SDL_Flip(screen);
                                        sceKernelDelayThread(0.4*1000*1000);
                                        break;
                                }

                        case 5:
                                {
                                        SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                        SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));

                                        clippingSprite(0,96);
                                        SDL_BlitSurface(bg, NULL, screen, NULL);
                                        SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                        SDL_Flip(screen);
                                        sceKernelDelayThread(0.4*1000*1000);
                                        break;
                                }

                        case 6:
                                {
                                        SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                        SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));

                                        clippingSprite(96,96);
                                        SDL_BlitSurface(bg, NULL, screen, NULL);
                                        SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                        SDL_Flip(screen);
                                        sceKernelDelayThread(0.4*1000*1000);
                                        break;
                                }

                        case 8:
                                {
                                        SDL_FillRect(screen, &spriteDestination, SDL_MapRGB(sprite-
>format, 255, 255, 255));
                                        SDL_SetColorKey(sprite,
SDL_SRCCOLORKEY|SDL_RLEACCEL, SDL_MapRGBA(sprite->format,255,255,255,255));

                                        clippingSprite(288,96);
                                        SDL_BlitSurface(bg, NULL, screen, NULL);
                                        SDL_BlitSurface(sprite, &spriteSoruce, screen,
&spriteDestination);
                                        SDL_Flip(screen);
                                        sceKernelDelayThread(0.4*1000*1000);
                                        i = 0;
                                        break;
                                }
                }       }
}
    SDL_FreeSurface(temp);
    SDL_FreeSurface(sprite);
    SDL_FreeSurface(bg);
   sceKernelSleepThread();

return 0;
```

Y su Makefile:

```
TARGET = Imagen
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin

OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS =  -lSDL_image -lpng -ljpeg -lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lstdc++ -lpspirkeyb -
lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = Imagen
include $(PSPSDK)/lib/build.mak
```

El resultado final de la prueba se puede ver en la **Figura 37**:

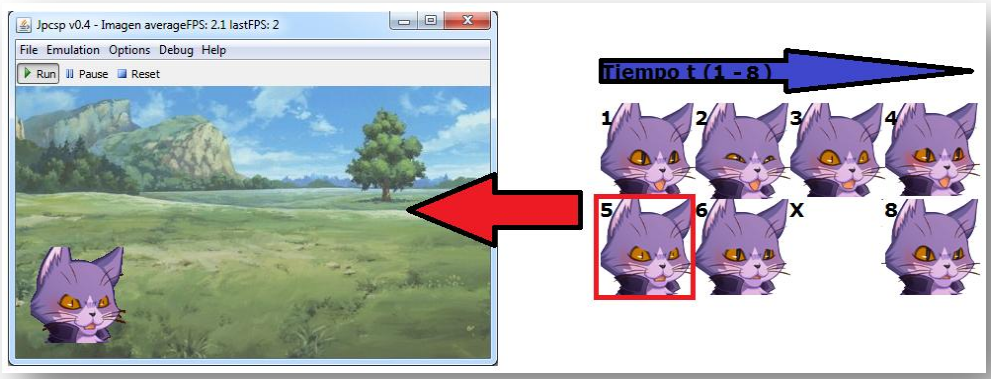


Figura 37: Captura de pantalla del Emulator de la prueba "SDL Animaciones".

Por alguna razón la séptima imagen está "rota". No fue posible hacerla transparente como las demás. Además la cuarta imagen salta un poco hacia arriba en la animación, porque dicha imagen está situada más arriba que las otras en el sprite.

4.3.6 "SDL menú" – sin clases

Hoy en día todas las AM poseen un menú con varios ítems y/o varios sub-menús. En esta parte del estudio se debe conocer todas las funciones necesarias para crear un menú simple con ayuda de SDL en C++. La idea general consiste en crear el típico menú de un videojuego para una videoconsola. Quiere decir que el menú principal contiene: "Pulsa START" y luego hay un sub-menú con las niveles de dificultad.

Para realizar un menú es necesario tener una fuente y unas imágenes, para poder mostrar al usuario que esté en el sub-menú o salta entre los dos menús (menú principal y sub-menú). En el proyecto de Metro Valencia de Anush Euredjian Chiappe y Juan Cortés Maiques [2]. Se creó un ejemplo de un menú muy sencillo en C con SDL. Pero hay que decir que la arquitectura de este programa contiene unos fallos muy peligros y riesgos. Cuando crezca el proyecto, el riesgo también crecerá, tal riesgo hace que sea muy elevado que se produzcan fallos de direccionamiento de memoria y que el número de líneas de código fuente suba muchísimo. La idea general de ambas está bastante bien pensada. Pero en el tiempo de la realización cometían fallos de mal uso y de mal entendimiento de lógica de "SDL_BlitSurface". Por ejemplo, en el código de Metro Valencia se crea para cada cambio de color de texto del menú una nueva función. Estas funciones cargan la imagen de fondo cada vez de nuevo al principio. Luego, se crea todos textos. Al final de dichas funciones, se liberan todas las superficies usadas para no causar problemas de relleno de memoria. La única diferencia entre estas dichas funciones insiste en un único valor. Cuando el menú contenga muchos ítems, entonces hubo que crear la misma cantidad de funciones como ítems (p.ej. si se quiere diseñar un teclado por pantalla de tipo "QWERTY", entonces hay más que 30 ítems).

Primera pregunta, antes de empezar a programar un menú sencillo - ¿Qué contiene un menú? - Como se ha comentado, es necesario tener varias imágenes por si acaso. También se sabe, que cada menú posee de unas áreas de texto. Entonces también es importante tener varias capas de texto. La única cosa que falta, es el control de botones lo que es responsable para visualizar el cambio de los ítems en el menú y/o también cambio de menú (p.ej. menú principal ↔ sub-menú).

Ahora empieza el problema de la lógica de visualización. La primera duda que sale es - ¿Hay que cargar la imagen cada vez? – NO, no hay que cargarla cada tiempo t, si no se comete un fallo muy tonto y peligroso en el programa. Esto lo puede causar un aumento muy elevado de código fuente y una reducción de velocidad de la aplicación, es decir, la aplicación se ejecuta más lento como debería por la mala arquitectura.

La segunda duda – ¿hay qué dibujar cada capa de texto? – SÍ y NO. Hay que dibujar cada capa de texto en el control, si no, no sale nada por pantalla. Pero no hay que dibujar todas en la lógica del menú. La lógica del menú es la parte responsable de cambiar el color de la capa de texto y en ninguna ocasión para dibujar. Cuesta su tiempo entender como SDL imprime texto por pantalla, porque contiene muchas piezas de lógica y de visualización. Además se debe entender bastante bien como hay que trabajar con punteros en C y C++.

La tercera duda - ¿Cuándo hay que actualizar la pantalla? – SIEMPRE, cuando hay un cambio de contexto / superficie. Sin llamar "SDL_Flip(pantalla)", el usuario no ve el cambio por pantalla. Pero en la aplicación en el buffer insiste el cambio de superficie.

En este ejemplo, no se ha resuelto la segunda duda de creación de un menú. El siguiente apartado, si que se ha resuelto este problema. Además se tiene la posibilidad de ver la programación con clases en C++.

Entonces, un menú debe tener dos partes de lógica, una de ellas es la parte lógica de control y otra es la parte lógica de visualización. Para no escribir cada vez el mismo código fuente que genera una capa de texto por pantalla, se programaba una función que lo hará.

El código fuente de dicha función:

```
void drawlabels(const char* labeltitel, int posx, int posy, SDL_Color lcolor)
{
    text = TTF_RenderText_Blended(fontLabel, labeltitel, lcolor);
    font.x = posx;
    font.y = posy;
    SDL_BlitSurface(text, NULL, screen, &font); }
```

En este ejemplo, el menú principal contiene un único ítem y el sub-menú posee 3 ítems. La lógica de visualización es el primer núcleo importante para realizar el cambio entre los ítems. Es muy importante saber cuántos ítems hay en un menú. En la **Tabla 3**, se enseña un ejemplo de la lógica.

Caso 1	Caso2	Caso3
ABC	ABC	ABC
DEF	**DEF**	DEF
GHI	GHI	**GHI**

Tabla 3: Ejemplo de lógica de visualización de un menú con 3 ítems.

A través de la **Tabla 3**, se puede ver que es necesario programar con "switch case" y un contendedor de ítems del menú. Luego en la lógica del menú, se programa el control de botones además de modificar el contendedor.

El código fuente de la parte de lógica de visualización:

```
void controlmenu(int menucounter)
{
  switch(menucounter)
  {
        case 1:
        {
              drawlabels("ABC", 100, 100, markedtext);
              drawlabels("DEF", 100, 150, unmarkedtext);
              drawlabels("GHI", 100, 200, unmarkedtext);
              break;
        }

        case 2:
        {
              drawlabels("ABC", 100, 100, unmarkedtext);
              drawlabels("DEF", 100, 150, markedtext);
              drawlabels("GHI", 100, 200, unmarkedtext);
              break;
        }

        case 3:
        {
              drawlabels("ABC", 100, 100, unmarkedtext);
              drawlabels("DEF", 100, 150, unmarkedtext);
              drawlabels("GHI", 100, 200, markedtext);
              break;
        }
  }
}
```

Ya se trabajaba un poco con el control de botones de la PSP. En este caso hay un cambio sencillo para poder usar varias veces el mismo botón. Antes de todas las comprobaciones, se comprueba si el último botón es equivalente al botón actual. Después se indica todos los valores posibles del contador. El cambio siempre pasa en la pulsación de los botones "arriba" y "abajo". Esta parte es el núcleo interno de toda la aplicación. La aplicación tiene que ejecutar esta parte en su tiempo de ejecución y acabarla cuando finaliza su ejecución. Cada aplicación lleva un bucle infinito (while(1) o while(true)).

El código fuente:

```
while(true)
{
	sceCtrlReadBufferPositive(&pad, 1);

	if(pad.Buttons != lastpad.Buttons)
	{
	lastpad = pad;

		if(pad.Buttons & PSP_CTRL_START)
		{
			if(OnStartGame)
			{
					OnStartGame = false;
					OnMainMenu = true;
					UpDownSetting = 1;

					SDL_BlitSurface(bg, NULL, screen, NULL);
					drawlabels("ABC", 100, 100, markedtext);
					drawlabels("DEF", 100, 150, unmarkedtext);
					drawlabels("GHI", 100, 200, unmarkedtext);

					SDL_Flip(screen);
			}
		}

		if(pad.Buttons & PSP_CTRL_CIRCLE)
		{
					OnStartGame = true;
					OnMainMenu = false;
					UpDownSetting = 1;
					SDL_BlitSurface(intro, NULL, screen, NULL);
					drawlabels("Press Start", 200, 200, unmarkedtext);
					SDL_Flip(screen);
		}

		if(pad.Buttons & PSP_CTRL_UP)
		{
			if(OnMainMenu)
			{

				UpDownSetting--;
				if (UpDownSetting <= 1) UpDownSetting = 1;
				SDL_BlitSurface(bg, NULL, screen, NULL);
				controlmenu(UpDownSetting);
				SDL_Flip(screen);
			}
		}

		if(pad.Buttons & PSP_CTRL_DOWN)
		{
			if(OnMainMenu)
			{
				UpDownSetting++;
				if (UpDownSetting >= 3) UpDownSetting = 3;
				SDL_BlitSurface(bg, NULL, screen, NULL);
				controlmenu(UpDownSetting);
				SDL_Flip(screen);
			}
		}}}
```

Con esta arquitectura de programa se puede genera un menú sin dificultad en C y C++ con SDL. Para este proyecto, la arquitectura del menú y su visualización son las bases más importantes. Con ayuda de creación del menú se puede programar fácilmente un teclado por pantalla. La **Figura 38** muestra en la funcionalidad del menú.

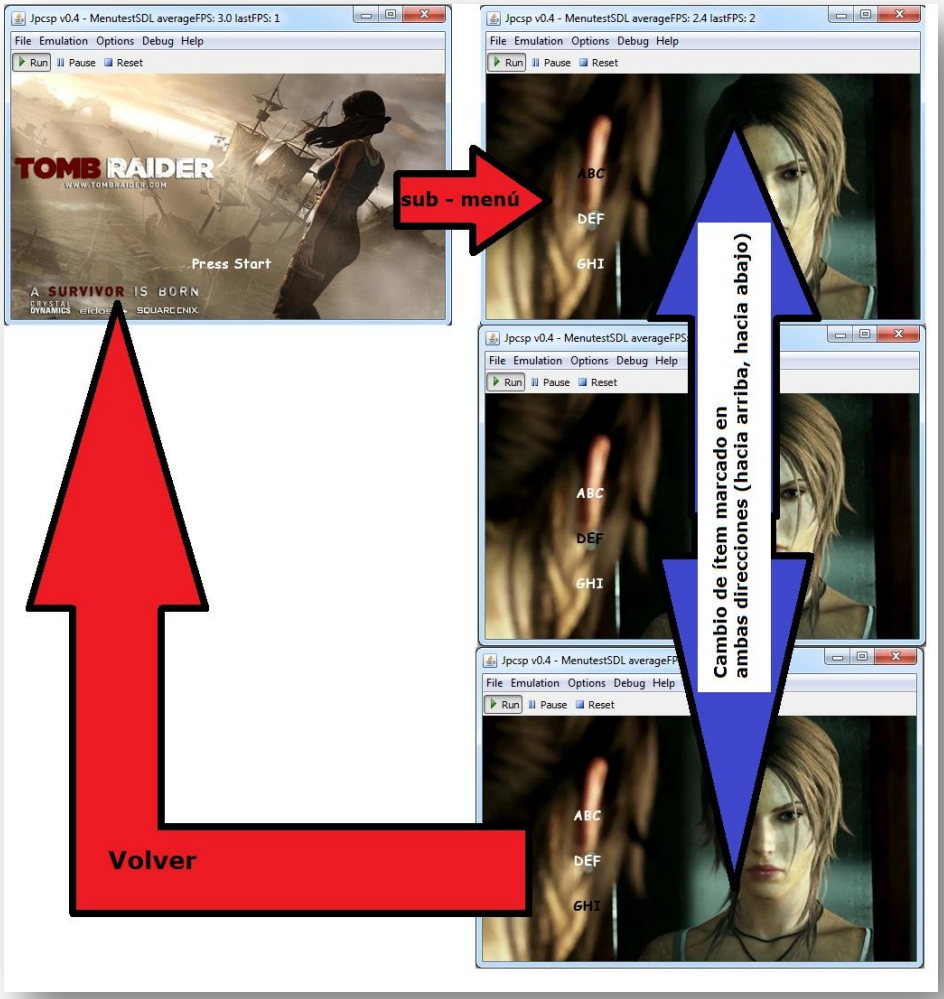


Figura 38: Capturas de pantalla de Emulador con la prueba del menú en SDL.

El código fuente final

```
#include <pspkernel.h>
#include <pspctrl.h>

#include <SDL/SDL.h>
#include <SDL/SDL_ttf.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>


#define printf pspDebugScreenPrintf

PSP_HEAP_SIZE_KB(20480);

   SDL_Surface *screen;
   SDL_Surface *temp, *bg, *intro;
   SDL_Surface *text;

   SDL_Rect font;
   SDL_Rect spriteSoruce, spriteDestination;

   TTF_Font *fontLabel;
   SDL_Color unmarkedtext, markedtext;


int ExitCallback(int Arg1, int Arg2, void *Common)
{
    sceKernelExitGame();
    return 0;
}

int CallbackThread(SceSize Args, void *Argp)
{
    int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
    sceKernelRegisterExitCallback(CallbackId);
    sceKernelSleepThreadCB();
    return 0;
}

int SetupCallbacks(void)
{
    int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
        if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
    return ThreadId;
}


void drawlabels(const char* labeltitel, int posx, int posy, SDL_Color lcolor)
{
   text = TTF_RenderText_Blended(fontLabel, labeltitel, lcolor);
   font.x = posx;
   font.y = posy;
   SDL_BlitSurface(text, NULL, screen, &font);

}

void controlmenu(int menucounter)
{
   switch(menucounter)
   {
            case 1:
            {
                     drawlabels("ABC", 100, 100, markedtext);
                     drawlabels("DEF", 100, 150, unmarkedtext);
```

```
                        drawlabels("GHI", 100, 200, unmarkedtext);
                        break;
                }

                case 2:
                {
                        drawlabels("ABC", 100, 100, unmarkedtext);
                        drawlabels("DEF", 100, 150, markedtext);
                        drawlabels("GHI", 100, 200, unmarkedtext);
                        break;
                }

                case 3:
                {
                        drawlabels("ABC", 100, 100, unmarkedtext);
                        drawlabels("DEF", 100, 150, unmarkedtext);
                        drawlabels("GHI", 100, 200, markedtext);
                        break;
                }
        }
}


extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
    SetupCallbacks();
    bool OnStartGame = true;
    bool OnMainMenu = false;

    int UpDownSetting = 1;

    SceCtrlData pad, lastpad;
    sceCtrlReadBufferPositive(&lastpad, 1);

    SDL_Init(SDL_INIT_VIDEO);
    SDL_ShowCursor(0);

    screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);
    TTF_Init();

    fontLabel = TTF_OpenFont("fonts/comic.ttf", 14);
    TTF_SetFontStyle(fontLabel, TTF_STYLE_BOLD);

    bg = IMG_Load("mainmenu.jpg");
    intro = IMG_Load("intro.jpg");
    SDL_BlitSurface(intro, NULL, screen, NULL);


    unmarkedtext.r = 255;      markedtext.r = 0;
    unmarkedtext.g = 255;      markedtext.g = 0;
    unmarkedtext.b = 255;      markedtext.b = 0;

    drawlabels("Press Start", 200, 200, unmarkedtext);
    SDL_Flip(screen);

    while(true)
    {
                sceCtrlReadBufferPositive(&pad, 1);

                if(pad.Buttons != lastpad.Buttons)
                {
                lastpad = pad;
```

```
                        if(pad.Buttons & PSP_CTRL_START)
                        {
                                if(OnStartGame)
                                {
                                                OnStartGame = false;
                                                OnMainMenu = true;
                                                UpDownSetting = 1;

                                                SDL_BlitSurface(bg, NULL, screen, NULL);
                                                drawlabels("ABC", 100, 100, markedtext);
                                                drawlabels("DEF", 100, 150, unmarkedtext);
                                                drawlabels("GHI", 100, 200, unmarkedtext);

                                                SDL_Flip(screen);
                                }
                        }

                        if(pad.Buttons & PSP_CTRL_CIRCLE)
                        {
                                                OnStartGame = true;
                                                OnMainMenu = false;
                                                UpDownSetting = 1;
                                                SDL_BlitSurface(intro, NULL, screen, NULL);
                                                drawlabels("Press Start", 200, 200, unmarkedtext);
                                                SDL_Flip(screen);
                        }

                        if(pad.Buttons & PSP_CTRL_UP)
                        {
                                if(OnMainMenu)
                                {

                                        UpDownSetting--;
                                        if (UpDownSetting <= 1) UpDownSetting = 1;
                                        SDL_BlitSurface(bg, NULL, screen, NULL);
                                        controlmenu(UpDownSetting);
                                        SDL_Flip(screen);
                                }
                        }

                        if(pad.Buttons & PSP_CTRL_DOWN)
                        {
                                if(OnMainMenu)
                                {
                                        UpDownSetting++;
                                        if (UpDownSetting >= 3) UpDownSetting = 3;
                                        SDL_BlitSurface(bg, NULL, screen, NULL);
                                        controlmenu(UpDownSetting);
                                        SDL_Flip(screen);
                                }
                        }
                }
        }

        SDL_FreeSurface(text);
        SDL_FreeSurface(intro);
        SDL_FreeSurface(bg);

         sceKernelSleepThread();

    return 0;
    }
```

Y el Makefile:

```
TARGET = MenuTest
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin

OBJS = main.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS =  -lSDL_mixer -lsmpeg -lstdc++ -lSDL_ttf -lfreetype -lvorbisidec -logg -lSDL_image -lpng -
ljpeg -lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lpspirkeyb -lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = MenutestSDL

include $(PSPSDK)/lib/build.mak
```

4.3.7 "SDL menú" – con clases

Este apartado ilustra el mismo objetivo que el apartado anterior. La diferencia entre ambos apartados consiste en la arquitectura de programa. En esta parte de la memoria, se ve por primera vez como se puede programar con clases en C++ con SDL en la PSP. Después de ver las funciones más importantes de SDL en este proyecto, es importante mostrar las ventajas de C++. Unas de ventajas de C++ es el uso de clases y objetos. La siguiente ventaja son los "strings" (cadena de caracteres). Cómo se convierte un "string" a un vector de caracters para poder usarlo con SDL, se lo muestra en el trabajo final "PSP Dictionary" alias "Tobí 'Dictionary".

Antes de todo, en esta prueba fue necesario crear varias clases (CSurface, CMenu y CMenuSurface) para realizar la técnica de divide y vencerás. Con la ayuda de la **Figura 39**, se ve todos los ficheros usados en dicha prueba.

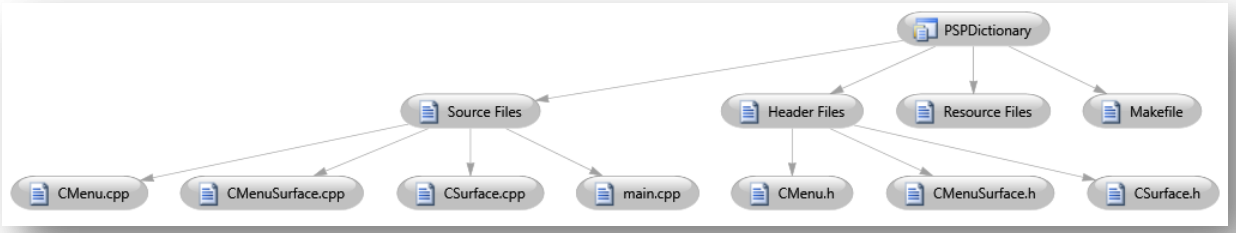


Figura 39: Diagrama de ficheros de la prueba "SDL Menú con clases".

Para especificar todas las clases es importante trabajar con los ficheros "headers". En el diagrama de clases, se ilustra la definición de cada clase usada.

- CSurface:
  Sirve para dibujar una superficie/ imagen por la pantalla.
  Contiene las funciones de cargar la imagen y dibujarla

- CMenuSurface:
  Sirve para cargar una fuente y cerrar / liberar recurso

- CMenu:
  Describe la estructura de un menú en el nivel de texto (capa de texto)

- Main:
  Es el núcleo de la aplicación que contiene un controlador de botones, dibuja por pantalla todo lo que requiere el escenario elegido y libera el recurso de imágenes y variables usadas en el main.

En la **Figura 40** se muestra todos los diagramas de clase.

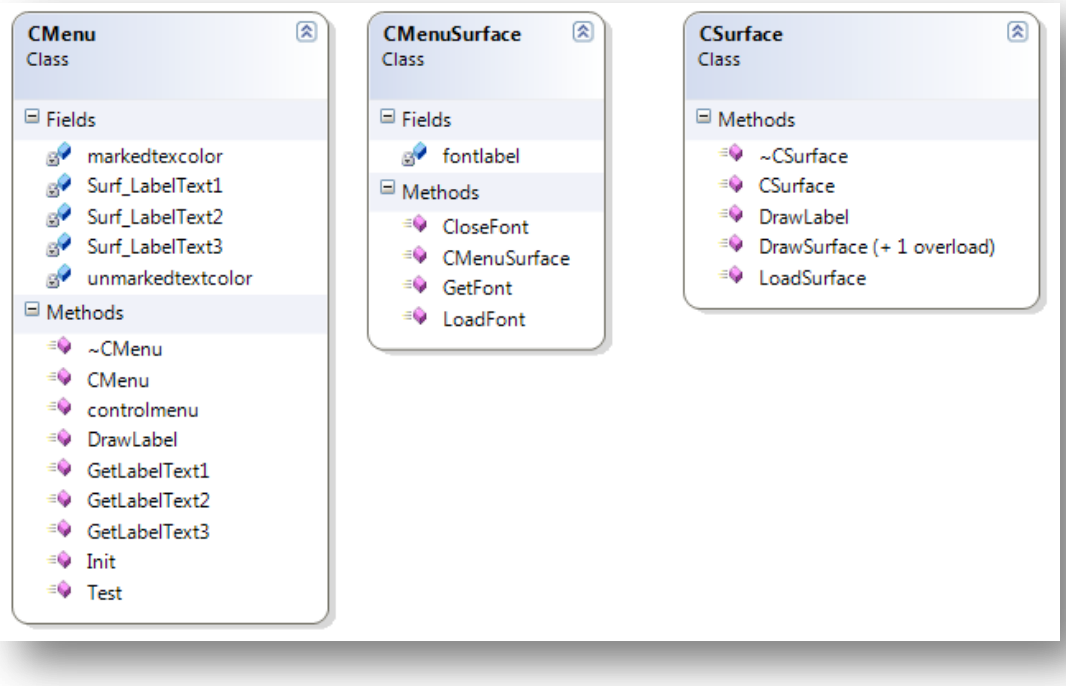


Figura 40: Diagrama de clase de la prueba "SDL Menú con clases".

OJO: CMenuSurface fue creado, porque existían unos problemas de puntero de fuente. Cuando se carga una fuente con SDL, se necesita un puntero para usarlo luego en la función "`TTF_RenderText_Blended(TTF_Font *fuente, const char *texto, SDL_Color color_de_texto)`" que crea una superficie (contiene las características de la fuente cargada, una cadena de caracteres cual se quiere mostrar por pantalla y el color de la cadena de caracteres).

Antes de empezar, hay que fijarse muy bien en la estructura de una superficie de tipo texto. El primer fallo de creación del menú fue en la lógica de usar imágenes y texto con el mal uso de la función "SDL_BlitSurface". El segundo fallo consistía en no entender el funcionamiento de la función: "TTF_RenderText_Blended(…)". En vez de cambiar el color

de capa de texto, se dibujó para cada caso una nueva superficie de tipo texto. Entonces, para un menú con 3 ítems se dibujó 9 veces una superficie de tipo texto por pantalla. Es igual decir que se dibujó n^2 una superficie por pantalla. En el ejemplo de creación de un menú con SDL se muestra este fallo**.** El fallo no es directamente visible. Solo cuando la aplicación crece. En el caso de la aplicación "Metro Valencia" de Anush Euredjian Chiappe y Juan Cortés Maiques[2]. El fallo de la lógica fue invisible, por el tamaño y el funcionamiento de la aplicación.

Como comentando antes, para dibujar un texto por pantalla, la función "TTF_RenderText_Blended" requiere 3 valores: fuente cargada, cadena de caracteres y color de cadena de caracteres. Además devuelve una superficie la cual se usará en el tema de "blitting".

El código fuente es el siguiente para imprimir "Hola" en color negro en la posición x=100 e y=150 por pantalla:

```
SDL_Surface *Surf_SrcText, screen;
SDL_Color color;
SDL_Rect text;

SDL_Init(SDL_INIT_VIDEO);
TTF_Init();
SDL_ShowCursor(0);

screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);

color.r = 0;                    text.x = 100;
color.b = 0;                    text.y = 150;
color.g = 0;

fontLabel = TTF_OpenFont("fonts/comic.ttf", 14);
TTF_SetFontStyle(fontLabel, TTF_STYLE_BOLD);

Surf_SrcText = TTF_RenderText_Blended(fuente, "Hola", color);

SDL_BlitSurface(SrcText _Src, NULL, screen, &text);
SDL_Flip(screen)
```

Con este ejemplo sencillo, se puede reconocer unos elementos fundamentales en el tema de SDL:

- Superficie destina / pantalla, superficie de tipo texto
- Rectángulo de superficie
- Uso de "TTF_RenderText(…) //crear superficie de texto
- Uso de "BlitSurface" //"blit" superficie a pantalla
- SDL_Flip(…) //actualizar la pantalla

Primero se creó la clase CSurface que sirve para dibujar una superficie por pantalla de cualquier tipo (imagen o texto) con los métodos "DrawSurface(…)". En este caso, se ha usado la tecnología de sobrecarga, porque en ambos métodos se hace casi lo mismo. La única diferencia consiste en el uso de "SDL_BlitSurface"[18]. Además se encuentra en el código fuente de cargar un fichero de imagen.

OJO: En C /C++ hay que tener mucho cuidado con los "#include" de una librería. La mejor manera de hacerlo correctamente, es trabajar con "#ifndef, #define, #endif" para asegurar que una librería no está cargando dos veces en el compilador

El código fuente del fichero "CSurface.h":

```
#ifndef CSurface_H_INCLUDED
#define CSurface_H_INCLUDED

#include <pspkernel.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>
#include <SDL/SDL_ttf.h>

class CSurface
{
private:

public:
   CSurface();
   ~CSurface();

   SDL_Surface *LoadSurface(const char *filepath);
   void DrawSurface(SDL_Surface *Surf_Src, SDL_Surface *Surf_Dest);
   void DrawSurface(SDL_Surface *Surf_Src, SDL_Surface *Surf_Dest, int sx, int sy, int sh, int sw,
int dx, int dy, int dh, int dw);
   void DrawLabel(SDL_Surface *Surf_SrcText, SDL_Surface *Surf_DestText, int posx, int posy);
};

#endif //CSurface_H_INCLUDED
```

[18] En el caso de tener dudas, vuelva a mirar el apartado, que trata como usar "SDL_BlitSurface" correctamente: ***4.3.5 "SDL animaciones"***

OJO: VS requiere un constructor (mismo nombre como la clase) y destructor (**~**mismo nombre como la clase) de cada clase, aunque no hay que inicializar los valores y/o liberar recurso (delete objeto).

Cuando se escribe el fichero (*.cpp), hay que incluir el fichero header de la propia clase **(#include** "nombre de clase.h**")**.

Cuando se escribe los métodos definidos en la clase, hay que añadir **SIEMPRE** el nombre de la clase en la cabecera de la función de la siguiente manera:
Valor que devuelve la función Nombre de Clase**::** Nombre de la función (Valores usando en la función)

Por ejemplos:
SDL_Surface *CSurface::LoadSurface(const char *filepath) {…}

void CMenu::Initi() {…}

El código fuente del fichero "CSurface.cpp":

```
#include "CSurface.h"

CSurface::CSurface(){}
CSurface::~CSurface(){}

SDL_Surface *CSurface::LoadSurface(const char *filepath)
{
  SDL_Surface *Surf_Temp = NULL;
  SDL_Surface *Surf_Return = NULL;

  if((Surf_Temp = IMG_Load(filepath)) == NULL)
        {
              pspDebugScreenInit();
              pspDebugScreenPrintf("No image loaded");
              return NULL;
        }
  Surf_Return = SDL_DisplayFormat(Surf_Temp);
  SDL_FreeSurface(Surf_Temp);

  return Surf_Return;
}

void CSurface::DrawSurface(SDL_Surface *Surf_Src, SDL_Surface *Surf_Dest)
{
  SDL_BlitSurface(Surf_Src, NULL, Surf_Dest, NULL);
}

void CSurface::DrawSurface(SDL_Surface *Surf_Src, SDL_Surface *Surf_Dest, int sx, int sy, int sh,
int sw, int dx, int dy, int dh, int dw)
{
  SDL_Rect Surf_RSrc;
  Surf_RSrc.x = sx;
  Surf_RSrc.y = sy;
  Surf_RSrc.h = sh;
  Surf_RSrc.w = sw;
```

```
    SDL_Rect Surf_RDest;
    Surf_RSrc.x = dx;
    Surf_RSrc.y = dy;
    Surf_RSrc.h = dh;
    Surf_RSrc.w = dw;

    SDL_BlitSurface(Surf_Src, &Surf_RSrc, Surf_Dest, &Surf_RDest);
}


void CSurface::DrawLabel(SDL_Surface *Surf_SrcText, SDL_Surface *Surf_DestText, int posx, int
posy)
{
    SDL_Rect Surf_RSrc;
    Surf_RSrc.x = posx;
    Surf_RSrc.y = posy;

    SDL_BlitSurface(Surf_SrcText, NULL, Surf_DestText, &Surf_RSrc);
    SDL_FreeSurface(Surf_SrcText);
}
```

Para imprimir texto por pantalla, se utiliza dos clases "CMenuSurface" y "CMenu". "CMenuSurface" carga la fuente, cierra la fuente y devuelve la fuente cargada para usarla en "CMenu".

Código fuente de "CMenuSurface.h":

```
#ifndef CMenuSurface_H_INCLUDED
#define CMenuSurface_H_INCLUDED

#include <pspkernel.h>
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>
#include <SDL/SDL_ttf.h>

class CMenuSurface
{
private:
            TTF_Font *fontlabel;
public:
    CMenuSurface();
    ~CMenuSurface();
    void LoadFont();
    void CloseFont();

    TTF_Font *GetFont();
};

#endif //CMenuSurface_H_INCLUDED
```

El código fuente de "CMenuSurface.cpp":

```
#include "CMenuSurface.h"

CMenuSurface::CMenuSurface() {}
CMenuSurface::~CMenuSurface() {}

void CMenuSurface::LoadFont()
{
   fontlabel = TTF_OpenFont("fonts/comic.ttf", 14);
   TTF_SetFontStyle(fontlabel, TTF_STYLE_BOLD);
}

TTF_Font *CMenuSurface::GetFont()
{
   return fontlabel;
}

void CMenuSurface::CloseFont()
{
   TTF_CloseFont(fontlabel);
}
```

En la clase de "CMenu" es importante entender el concepto de la nueva lógica del menú. Ahora se cambia sólo el color del texto imprimido en la lógica del menú. El texto se dibuja finalmente en el main del programa. La función "SDL_Surface *DrawLabel(const char* labeltext, TTF_Font *fontlabel, SDL_Color colorlabel)" crea un superficie de tipo texto, cual se usará en el main, cuando se realiza el "blitting".

Código fuente de "CMenu.h":

```
#ifndef CMenu_H_INCLUDED
#define CMenu_H_INCLUDED

#include "CMenuSurface.h"

class CMenu
{
private:
   SDL_Surface *Surf_LabelText1, *Surf_LabelText2, *Surf_LabelText3;
   SDL_Color markedtexcolor, unmarkedtextcolor;

public:
   CMenu();
   ~CMenu();

   void Init();

   SDL_Surface *DrawLabel(const char* labeltext, TTF_Font *fontlabel, SDL_Color colorlabel);
   void controlmenu(int UpDownSetting);

   SDL_Surface *Test();
   SDL_Surface *GetLabelText1();
   SDL_Surface *GetLabelText2();
   SDL_Surface *GetLabelText3();
};

#endif //CMenu_H_INCLUDED
```

El código fuente de "CMenu.cpp":

```
#include "CMenu.h"

CMenuSurface *menusurface = new CMenuSurface();

CMenu::CMenu()
{
  markedtexcolor.r = 255;    unmarkedtextcolor.r = 255;
  markedtexcolor.g = 0;      unmarkedtextcolor.g = 255;
  markedtexcolor.b = 0;      unmarkedtextcolor.b = 255;
}
CMenu::~CMenu()
{
  menusurface->CloseFont();
  delete menusurface;
}

void CMenu::Init()
{
  menusurface->LoadFont();
}

SDL_Surface *CMenu::DrawLabel(const char* labeltext, TTF_Font *fontlabel, SDL_Color colorlabel)
{
  SDL_Surface *Surf_SrcText;

  Surf_SrcText = TTF_RenderText_Blended(fontlabel, labeltext, colorlabel);

  return Surf_SrcText;
}

SDL_Surface *CMenu::Test()
{
  Surf_LabelText1 = DrawLabel("Press Start", menusurface->GetFont(), unmarkedtextcolor);
  return Surf_LabelText1;
}

void CMenu::controlmenu(int UpDownSetting)
{
  switch(UpDownSetting)
  {
        case 1:
              {
                    Surf_LabelText1 = DrawLabel("ABC", menusurface->GetFont(),
markedtexcolor);
                    Surf_LabelText2 = DrawLabel("DEF", menusurface->GetFont(),
unmarkedtextcolor);
                    Surf_LabelText3 = DrawLabel("GHI", menusurface->GetFont(),
unmarkedtextcolor);
                    break;
              }

        case 2:
              {
                    Surf_LabelText1 = DrawLabel("ABC", menusurface->GetFont(),
unmarkedtextcolor);
                    Surf_LabelText2 = DrawLabel("DEF", menusurface->GetFont(),
markedtexcolor);
                    Surf_LabelText3 = DrawLabel("GHI", menusurface->GetFont(),
unmarkedtextcolor);
                    break;
              }

        case 3:
              {
```

```
                                Surf_LabelText1 = DrawLabel("ABC", menusurface->GetFont(),
unmarkedtextcolor);
                                Surf_LabelText2 = DrawLabel("DEF", menusurface->GetFont(),
unmarkedtextcolor);
                                Surf_LabelText3 = DrawLabel("GHI", menusurface->GetFont(),
markedtexcolor);
                                break;
                        }
    }
}

SDL_Surface *CMenu::GetLabelText1() { return Surf_LabelText1; }
SDL_Surface *CMenu::GetLabelText2() { return Surf_LabelText2; }
SDL_Surface *CMenu::GetLabelText3() { return Surf_LabelText3; }
```

El "blitting" se realiza en el main con la función creada: "void DrawLabel(SDL_Surface *Surf_SrcText, SDL_Surface *Surf_DestText, int posx, int posy)" de la clase "CSurface".

El código fuente de "main.cpp":

```
#include <pspkernel.h>
#include <pspctrl.h>

#include <SDL/SDL.h>
#include <SDL/SDL_ttf.h>
#include <SDL/SDL_image.h>
#include <SDL/SDL_mixer.h>

#include "CSurface.h"
#include "CMenu.h"

#define printf pspDebugScreenPrintf

PSP_HEAP_SIZE_KB(20480);

   SDL_Surface *screen;
   SDL_Surface *temp, *bg, *intro;

   SDL_Rect spriteSoruce, spriteDestination;

   TTF_Font *fontLabel;
   SDL_Color unmarkedtext, markedtext;


int ExitCallback(int Arg1, int Arg2, void *Common)
{
    sceKernelExitGame();
    return 0;
}
int CallbackThread(SceSize Args, void *Argp)
{
    int CallbackId = sceKernelCreateCallback("Exit Callback", ExitCallback, NULL);
    sceKernelRegisterExitCallback(CallbackId);
    sceKernelSleepThreadCB();
    return 0;
}

int SetupCallbacks(void)
{
```

```
    int ThreadId = sceKernelCreateThread("update_thread",CallbackThread,0x11,0xFA0,0,0);
        if(ThreadId >= 0) sceKernelStartThread(ThreadId,0,0);
    return ThreadId;
}


extern "C" int SDL_main (int argc, char* args[]);

int main(int argc, char *args[])
{
    SetupCallbacks();
   bool OnStartGame = true;
   bool OnMainMenu = false;

   CSurface *surface = new CSurface();
   CMenu *menu = new CMenu();

   int UpDownSetting = 1;

   SceCtrlData pad, lastpad;
   sceCtrlReadBufferPositive(&lastpad, 1);

    SDL_Init(SDL_INIT_VIDEO);
    SDL_ShowCursor(0);

    screen = SDL_SetVideoMode(480,272,32,SDL_HWSURFACE|SDL_DOUBLEBUF);
   TTF_Init();

   menu->Init();

   bg = surface->LoadSurface("mainmenu.jpg");
   intro = surface->LoadSurface("intro.jpg");
   SDL_BlitSurface(intro, NULL, screen, NULL);

   surface->DrawLabel(menu->Test(), screen , 200, 200);
   SDL_Flip(screen);

   while(true)
   {
            sceCtrlReadBufferPositive(&pad, 1);

            if(pad.Buttons != lastpad.Buttons)
            {
            lastpad = pad;

                    if(pad.Buttons & PSP_CTRL_START)
                    {
                            if(OnStartGame)
                            {
                                    OnStartGame = false;
                                    OnMainMenu = true;
                                    UpDownSetting = 1;

                                    surface->DrawSurface(bg, screen);

                                    menu->controlmenu(UpDownSetting);
                                    surface->DrawLabel(menu->GetLabelText1(),
screen , 200, 50);
                                    surface->DrawLabel(menu->GetLabelText2(),
screen , 200, 100);
                                    surface->DrawLabel(menu->GetLabelText3(),
screen , 200, 150);

                                    SDL_Flip(screen);
                            }
```

```
                    }

                    if(pad.Buttons & PSP_CTRL_CIRCLE)
                    {
                                        OnStartGame = true;
                                        OnMainMenu = false;
                                        UpDownSetting = 1;
                                        surface->DrawSurface(intro, screen);
                                        surface->DrawLabel(menu->Test(), screen , 200,
200);
                                        SDL_Flip(screen);
                    }

                    if(pad.Buttons & PSP_CTRL_DOWN)
                    {
                            if(OnMainMenu)
                            {
                                    OnStartGame = false;
                                    UpDownSetting++;
                                    if (UpDownSetting >= 3) UpDownSetting = 3;

                                    surface->DrawSurface(bg, screen);

                                    menu->controlmenu(UpDownSetting);
                                    surface->DrawLabel(menu->GetLabelText1(), screen , 200,
50);
                                    surface->DrawLabel(menu->GetLabelText2(), screen , 200,
100);
                                    surface->DrawLabel(menu->GetLabelText3(), screen , 200,
150);

                                    SDL_Flip(screen);
                            }
                    }

                    if(pad.Buttons & PSP_CTRL_UP)
                    {
                            if(OnMainMenu)
                            {
                                    OnStartGame = false;
                                    UpDownSetting--;
                                    if (UpDownSetting <= 1) UpDownSetting = 1;

                                    surface->DrawSurface(bg, screen);

                                    menu->controlmenu(UpDownSetting);
                                    surface->DrawLabel(menu->GetLabelText1(), screen , 200,
50);
                                    surface->DrawLabel(menu->GetLabelText2(), screen , 200,
100);
                                    surface->DrawLabel(menu->GetLabelText3(), screen , 200,
150);

                                    SDL_Flip(screen);
                            }
                    }
            }
    }

    SDL_FreeSurface(intro);
    SDL_FreeSurface(bg);

    delete surface;
    delete menu;
    TTF_Quit();
```

```
    SDL_Quit();
     sceKernelSleepThread();

return 0;
}
```

Ahora falta el nuevo Makefile. La única cosa que hay que actualizar en el Makefile es la parte OBJS. Hay que añadir todos los objetos creados (todos los nombres de clases con "*.o") que se usará en la aplicación.

El Makefile:

```
TARGET = PSPDictionary
PSPSDK = C:/pspsdk/psp/sdk
PSPBIN= C:/pspsdk/bin

OBJS = main.o CSurface.o CMenuSurface.o CMenu.o
CFLAGS = -Wall -Wno-long-long -G0 -O2 -DJOY_$(JOY)
CFLAGS += $(shell $(PSPBIN)/sdl-config --cflags)
CXXFLAGS = $(DEFAULT_CFLAGS) -fno-exceptions -fno-rtti

LIBS =  -lSDL_mixer -lsmpeg -lstdc++ -lSDL_ttf -lfreetype -lvorbisidec -logg -lSDL_image -lpng -
ljpeg -lz -lSDL $(shell $(PSPBIN)/sdl-config --libs) -lpspirkeyb -lpsppower

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = Tobis Dictionary

include $(PSPSDK)/lib/build.mak
```

El resultado final de esta prueba es la misma que la de la prueba anterior.

4.3.8 WiFi

El código de WiFi está dado en el SDK, pero fue necesario hacer una modificación por el tema de casting y cambiar el nombre de la variable "new"[19] en el caso de C++. Dicho código fue modificado en la documentación de Anush Euredjian Chiappe y Juan Cortés Maiques en la parte de mensajes en la consola [2]. A través del telnet se puede enviar mensajes a la PSP, que esta conectada en la misma red.

El código modificado por Anush Euredjian Chiappe y Juan Cortés Maiques [2]

```
#include <pspnet_apctl.h>
#include <pspnet_resolver.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <errno.h>
#include <psputility_netmodules.h>
#include <psputility_netparam.h>
#include <pspwlan.h>

#define printf pspDebugScreenPrintf

#define MODULE_NAME "Wifi"
#define HELLO_MSG   "Bienvenido. Ya puedes escribir.\r\n"
#define SERVER_PORT 23

PSP_MODULE_INFO(MODULE_NAME, 0, 1, 1);
PSP_MAIN_THREAD_ATTR(0);

/* Exit callback */
int exit_callback(int arg1, int arg2, void *common) {
        sceKernelExitGame();
        return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp) {
        int cbid;

        cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
        sceKernelRegisterExitCallback(cbid);

        sceKernelSleepThreadCB();

        return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void) {
        int thid = 0;

        thid = sceKernelCreateThread("update_thread", CallbackThread, 0x11, 0xFA0, 0, 0);
        if(thid >= 0) {
                sceKernelStartThread(thid, 0, 0);
        }

        return thid;}
```

[19] „new" se usa para crear un objeto.

```
int make_socket(uint16_t port)
{
   int sock;
   int ret;
   struct sockaddr_in name;

   sock = socket(PF_INET, SOCK_STREAM, 0);
   if(sock < 0)
   {
            return -1;
   }

   name.sin_family = AF_INET;
   name.sin_port = htons(port);
   name.sin_addr.s_addr = htonl(INADDR_ANY);
   ret = bind(sock, (struct sockaddr *) &name, sizeof(name));
   if(ret < 0)
   {
            return -1;
   }

   return sock;
}

/* Start a simple tcp echo server */
void start_server(const char *szIpAddr)
{
   int ret;
   int sock;
   int nuevo = -1;
   struct sockaddr_in client;
   size_t size;
   int readbytes;
   char data[1024];
   fd_set set;
   fd_set setsave;

   /* Create a socket for listening */
   sock = make_socket(SERVER_PORT);
   if(sock < 0)
   {
            printf("Error creando el server socket\n");
            return;
   }

   ret = listen(sock, 1);
   if(ret < 0)
   {
            printf("Error al escuchar en el socket\n");
            return;
   }

   printf("Esperando conexiones ip %s puerto %d\n", szIpAddr, SERVER_PORT);

   FD_ZERO(&set);
   FD_SET(sock, &set);
   setsave = set;

   while(1)
   {
            int i;
            set = setsave;
            if(select(FD_SETSIZE, &set, NULL, NULL, NULL) < 0)
            {
                     printf("error de seleccion\n");
```

```
                        return;
                }

                for(i = 0; i < FD_SETSIZE; i++)
                {
                        if(FD_ISSET(i, &set))
                        {
                                int val = i;

                                if(val == sock)
                                {
                                        nuevo = accept(sock, (struct sockaddr *) &client, &size);
                                        if(nuevo < 0)
                                        {
                                                printf("Error al acceptar la conexion %s\n",
strerror(errno));
                                                close(sock);
                                                return;
                                        }

                                        printf("Nueva Conexion %d desde %s:%d\n", val,
                                                        inet_ntoa(client.sin_addr),
                                                        ntohs(client.sin_port));

                                        write(nuevo, HELLO_MSG, strlen(HELLO_MSG));

                                        FD_SET(nuevo, &setsave);
                                }
                                else
                                {
                                        readbytes = read(val, data, sizeof(data));
                                        if(readbytes <= 0)
                                        {
                                                printf("Socket %d cerrado\n", val);
                                                FD_CLR(val, &setsave);
                                                close(val);
                                        }
                                        else
                                        {
                                                write(val, data, readbytes);
                                                printf("%.*s", readbytes, data);
                                        }
                                }
                        }
                }
        }

        close(sock);
}

/* Connect to an access point */
int connect_to_apctl(int config)
{
        int err;
        int stateLast = -1;

        /* Connect using the first profile */
        err = sceNetApctlConnect(config);
        if (err != 0)
        {
                printf(MODULE_NAME ": sceNetApctlConnect devuelve %08X\n", err);
                return 0;
        }

        printf(MODULE_NAME ": Conectando...\n");
```

```
	while (1)
	{
		int state;
		int err = sceNetApctlGetState(&state);
		if (err != 0)
		{
			printf("Error en el estado %i \n", state);
			sceKernelDelayThread(2000*1000);
			return 0; // connection failed
		}

		if (state > stateLast)
		{
			printf("Estado %i de 4\n", state);
			stateLast = state;
		}

		if (state == 4)
			break;  // connected with static IP

		// wait a little before polling again
		sceKernelDelayThread(200*1000); // 200 ms
	}
	printf(MODULE_NAME ": Conectado!\n");

	if(err != 0)
	{
		return 0;
	}

	return 1;
}

int net_thread(SceSize args, void *argp)
{
	int err;

	do
	{
		if((err = pspSdkInetInit()))
		{
			printf(MODULE_NAME ": Error, no se puede iniciar la red %08X\n", err);
			break;
		}

		if(connect_to_apctl(1))
		{
			// connected, get my IPADDR and run test
			union SceNetApctlInfo info;

			if (sceNetApctlGetInfo(8, &info) != 0)
				strcpy(info.ip, "IP desconocida");

			start_server(info.ip);
		}
	}
	while(0);

	return 0;
}

/* Simple thread */
int main(int argc, char **argv)
{
	SceUID thid;
```

```
    SetupCallbacks();

    pspDebugScreenInit();

     if (sceUtilityLoadNetModule(PSP_NET_MODULE_COMMON) < 0)
     {
       printf("Error, no se puede cargar PSP_NET_MODULE_COMMON\n");
       sceKernelSleepThread();
     }
     if (sceUtilityLoadNetModule(PSP_NET_MODULE_INET) < 0)
     {
       printf("Error, no se puede cargar PSP_NET_MODULE_INET\n");
       sceKernelSleepThread();
     }

    /* Create a user thread to do the real work */
    thid = sceKernelCreateThread("net_thread", net_thread, 0x18, 0x10000,
PSP_THREAD_ATTR_USER, NULL);
    if(thid < 0)
    {
              printf("Error, no se puede crear el hilo\n");
              sceKernelSleepThread();
    }

    sceKernelStartThread(thid, 0, NULL);

    sceKernelExitDeleteThread(0);

    return 0;
}
```

El cambio consiste en hacer un casting, si no el compilador muestra el siguiente mensaje de error:

```
C:\Users\Kquadrato\Desktop\Emulador jcpsp\MS0\PSP\GAME\Wifi1 - Copy>make
psp-g++ -I. -IC:/pspsdk/psp/sdk/include -O0 -G0 -Wall -g -I. -IC:/pspsdk/psp/sdk/include -O0 -G0 -
Wall -g -fno-exceptions -fno-rtti -D_PSP_FW_VERSION=
150   -c -o main.o main.cpp
main.cpp: In function 'void start_server(const char*)':
main.cpp:135: error: invalid conversion from 'size_t*' to 'socklen_t*'
main.cpp:135: error:   initializing argument 3 of 'int accept(int, sockaddr*, socklen_t*)'
make: *** [main.o] Error 1
```

La solución es cambiar los parámetros de la función "accept", mejor dicher hacer un casting a las variables necesarias de "accept". Según el mensaje de error, la función "accept" requiere un int, sockaddr y socklen_t:

```
nuevo = accept(sock, reinterpret_cast<sockaddr *>(&client) , reinterpret_cast<socklen_t
*>(&size));
```

OJO: Para comprobar si funciona este ejemplo es obstante tener un router WiFi para conectar la PSP con el ordenador, porque con el emulator no se puede comprobar. Además la PSP tiene que estar conectada con el router con el ordenador. Asimismo, para Windows 7 hay que activar la orden "telnet" para poder usar luego el cmd y abrir una conexión telnet.

El código original funciona sin problemas en C. Para C++ hay que cambiar ciertas cosas en el código (cambiar nombre de variable "new", hacer 2 castings).

4.3.9 USB

El código de USB está dado en el SDK, pero fue necesario cambiarlo un poco para poder ejecutarlo en la PSP, porque si no, la PSP lanza un mensaje de error de 80020148 (PRX type unsupported)[20]. Quiere decir, que en el código se hace referencia a una funcion del programa de PRX. El "Hombrew" activa o desactiva la conexión USB a través del botón ⊗.

El código original:

```
/*
 * PSP Software Development Kit - http://www.pspdev.org
 * -----------------------------------------------------------------------
 * Licensed under the BSD license, see LICENSE in PSPSDK root for details.
 *
 * main.c - Sample to demonstrate USB Mass Storage functionality
 *
 * Copyright (c) 2005 John Kelley <ps2dev@kelley.ca>
 *
 * Derived from code written by PspPet
 * Code flow and API usage from VSH module "sysconf_plugin.prx"
 *
 * $Id$
 */
#include <pspkernel.h>
#include <pspiofilemgr.h>
#include <pspmodulemgr.h>
#include <pspdisplay.h>
#include <pspdebug.h>
#include <pspusb.h>
#include <pspusbstor.h>
#include <pspthreadman.h>
#include <pspctrl.h>
#include <pspsdk.h>

/**
 * Define the module info section
 *
 * 2nd arg must 0x1000 so __init is executed in
 * kernelmode for our loaderInit function
 */
PSP_MODULE_INFO("USBSample", 0x1000, 1, 0);

/**
 * THREAD_ATTR_USER causes the thread that the startup
 * code (ctr0.c) starts this program in to be in usermode
 * even though the module was started in kernelmode
 */
PSP_MAIN_THREAD_ATTR(THREAD_ATTR_USER);

/* Define printf, just to make typing easier */
#define printf  pspDebugScreenPrintf

//make this global so we can check it in the exit_callback
u32 state;
```

[20] PRX tipo no soportado

```
/**
 * Function that is called from _init in kernelmode before the
 * main thread is started in usermode.
 */
__attribute__ ((constructor))
void loaderInit()
{
    pspKernelSetKernelPC();
    pspSdkInstallNoDeviceCheckPatch();
    pspDebugInstallKprintfHandler(NULL);
}

/* Exit callback */
int exit_callback(int arg1, int arg2, void *common)
{
    int retVal;

    //cleanup drivers
    if (state & PSP_USB_ACTIVATED) {
   retVal = sceUsbDeactivate(0);
   if (retVal != 0)
      printf("Error calling sceUsbDeactivate (0x%08X)\n", retVal);
    }
    retVal = sceUsbStop(PSP_USBSTOR_DRIVERNAME, 0, 0);
    if (retVal != 0)
   printf("Error stopping USB Mass Storage driver (0x%08X)\n",
         retVal);

    retVal = sceUsbStop(PSP_USBBUS_DRIVERNAME, 0, 0);
    if (retVal != 0)
   printf("Error stopping USB BUS driver (0x%08X)\n", retVal);

    sceKernelExitGame();

   return 0;
}

/* Callback thread */
int CallbackThread(SceSize args, void *argp)
{
    int cbid;

    cbid = sceKernelCreateCallback("Exit Callback", exit_callback, NULL);
    sceKernelRegisterExitCallback(cbid);
    sceKernelSleepThreadCB();

    return 0;
}

/* Sets up the callback thread and returns its thread id */
int SetupCallbacks(void)
{
    int thid = 0;

    thid = sceKernelCreateThread("update_thread", CallbackThread,
                                 0x11, 0xFA0, 0, 0);
    if (thid >= 0) {
   sceKernelStartThread(thid, 0, 0);
    }

    return thid;
}

//helper function to make things easier
```

```
int LoadStartModule(char *path)
{
    u32 loadResult;
    u32 startResult;
    int status;

    loadResult = sceKernelLoadModule(path, 0, NULL);
    if (loadResult & 0x80000000)
   return -1;
    else
   startResult =
      sceKernelStartModule(loadResult, 0, NULL, &status, NULL);

    if (loadResult != startResult)
   return -2;

    return 0;
}

int main(void)
{
    SceCtrlData pad;
    u32 oldButtons = 0;
    u32 retVal;

    state = 0;
    pspDebugScreenInit();
    pspDebugScreenClear();
    SetupCallbacks();

    //setup Pad
    sceCtrlSetSamplingCycle(0);
    sceCtrlSetSamplingMode(0);

    //print header now in case we have any errors
    printf("USB Sample v1.0 by John_K - Based off work by PSPPet\n");

    //start necessary drivers
    LoadStartModule("flash0:/kd/semawm.prx");
    LoadStartModule("flash0:/kd/usbstor.prx");
    LoadStartModule("flash0:/kd/usbstormgr.prx");
    LoadStartModule("flash0:/kd/usbstorms.prx");
    LoadStartModule("flash0:/kd/usbstorboot.prx");

    //setup USB drivers
    retVal = sceUsbStart(PSP_USBBUS_DRIVERNAME, 0, 0);
    if (retVal != 0) {
   printf("Error starting USB Bus driver (0x%08X)\n", retVal);
   sceKernelSleepThread();
    }
    retVal = sceUsbStart(PSP_USBSTOR_DRIVERNAME, 0, 0);
    if (retVal != 0) {
   printf("Error starting USB Mass Storage driver (0x%08X)\n",
         retVal);
   sceKernelSleepThread();
    }
    retVal = sceUsbstorBootSetCapacity(0x800000);
    if (retVal != 0) {
   printf
      ("Error setting capacity with USB Mass Storage driver (0x%08X)\n",
       retVal);
   sceKernelSleepThread();
    }
   retVal = sceUsbstorBootSetCapacity(0x800000);//sceUsbstorBootSetCapacity seems to be
removed from the firmware
```

```
        if (retVal != 0) {
        printf ("Error setting capacity with USB Mass Storage driver (0x%08X)\n",  retVal);
        sceKernelSleepThread();
        }

        retVal = 0;

        //if everything worked we now enter our main loop
        for (;;) {

        sceCtrlReadBufferPositive(&pad, 1);
        state = sceUsbGetState();
        pspDebugScreenSetXY(0, 0);
        printf("USB Sample v1.0 by John_K - Based off work by PSPPet\n\n");
        printf("%-14s: %s\n", "USB Driver",
              state & PSP_USB_ACTIVATED ? "activated     " :
              "deactivated");
        printf("%-14s: %s\n", "USB Cable",
              state & PSP_USB_CABLE_CONNECTED ? "connected     " :
              "disconnected");
        printf("%-14s: %s\n", "USB Connection",
              state & PSP_USB_CONNECTION_ESTABLISHED ? "established" :
              "not present");
        printf("\nPress X to establish or destroy the USB connection\n");

        if (oldButtons != pad.Buttons) {
           if (pad.Buttons & PSP_CTRL_CROSS) {
                 if (state & PSP_USB_ACTIVATED)
                    retVal = sceUsbDeactivate(0x1c8);
                 else
                    retVal = sceUsbActivate(0x1c8);
           }
        }
        oldButtons = pad.Buttons;
        sceDisplayWaitVblankStart();
        }


        //Exit program
        sceKernelExitGame();

        return 0;
    }
```

El cambio consiste en borrar o comentar la sigiente parte del código, porque el código solo funciona con la PSP de "Firmware" 1.50.

```
//    retVal = sceUsbstorBootSetCapacity(0x800000);//sceUsbstorBootSetCapacity seems to be
removed from the firmware
//    if (retVal != 0) {
//  printf
//      ("Error setting capacity with USB Mass Storage driver (0x%08X)\n",
//       retVal);
//  sceKernelSleepThread();
//    }
```

El Makefile:

```
PSPSDK = $(shell psp-config --pspsdk-path)
PSPLIBSDIR = $(PSPSDK)/..
TARGET = usbstorage
OBJS = main.o
LIBS =-lstdc++ -lpspusb -lpspusbstor
CFLAGS = -O2 -G0 -Wall
CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti
ASFLAGS = $(CFLAGS)
BUILD_PRX = 1
EXTRA_TARGETS = EBOOT.PBP
include $(PSPSDK)/lib/build.mak
```

OJO: Para comprobar si funciona este ejemplo es obstante tener un cable USB para la PSP, porque con el emulator no se puede comprobar. Además la PSP tiene que estar conectada con el USB en el ordenador.

El código es igual en C.

5. Diseño

Este capítulo ilustra el diseño de los menús, de los teclados debido al resultado del análisis[21]. Además el diseño planificado para un fichero XML para el diccionario. Aparte del diseño, cada sub-apartado habla en pocas palabras de cómo se programa p.ej. un teclado para la PSP.

5.1 Menús

Los menús sirven para dividir información sobre varios campos / conjuntos. En este caso se ha creado un menú principal con cuatro ítems ("Load Dictionaries", "Dictionary", "Extra" y "Help")[22]. El ítem"Load Dictionaries" sirve para cambiar el diccionario actual en la apliación. Por ejemplo, antes la apliación ha traducido de español a inglés, luego el usuario quiere saber la traducción de una palabra inglesa a una palabra española. Por estos motivos se ha creado la opción "Load Dictionaries" para poder cambiar el diccionario en cualquier momento. En "Load Dictionaries" hay una restricción de 10 diccionarios para poder controlar la cantidad de información en la aplicación y también para evitar que la aplicación crezca demasiado de tamaño. "Dictionary" como indica el nombre, sirve para buscar la traducción de una palabra a otra palabra. En este ítem se muestra las opciones de "Search", "Search Random", "Search Online", "History"[23]. La opción "Search" ofrece un propio "OSK" – el teclado "inteligente" y el teclado "QWERTY". Es posible cambiar entre ambos teclados en cualquier momento. "Search Random", elige una palabra aleatoriamente y muestra directamente la traducción de dicha palabra. Para hacer el diccionario más inteligente hay que implementar una inteligencia artificial para evitar que el diccionario siempre esté buscando las mismas palabras aleatoriamente. "Search Online", como indica el nombre, debería ofrecer al usuario la posibilidad de acceder directamente al traductor de google o a un diccionario online. "History", debería mostrar las cinco últimas palabras de las últimas búsquedas. El menú "Extra" sirve para juegos de tipo mini exámenes u opciones, como por ejemplo añadir un nuevo diccionario. Como antes comentado, existe una restricción de 10 diccionarios. El usuario debería tener la posibilidad de cambiar los diccionarios o también cambiar el orden de los diccionarios según la necesidad. Y para cambiar el idioma del diccionario. Los juegos deberían usar la reproducción de audio. Por ejemplo el diccionario o la mascota dice una palabra y el usuario tiene que escribirla o decir cual de las palabras mostradas por pantalla es correcta. Debido al análisis, es obstante ofrecer esta opción, porque un diccionario no solo debe ser

[21] Se puede ver el análisis en el apéndice "Análisis y información adicionan"

[22] (Cargar Diccionarios, Diccionario, Extra, Ayuda)

[23] (Búsqueda, Búsqueda aleatoria, Búsqueda online, historia de búsquedas)

para buscar si no también para aprender nuevos vocabularios o repetirlos para no olvidarlos. El último menú sirve como un manual del diccionario. Si el usuario no se acuerda más de lo qué hace cada botón, tiene la posibilidad de volver a mirarlo en "Help".

Con todos estos detalles, fue necesario mirar cómo mostrarlos y utilizarlos. La mejor posibilidad de orientarse es mirar los ejemplos que hay, en este caso, mirar los menús de los videojuegos. Especialmente los videojuegos de tipo "RPG" usan muchos detalles en un menú, como los estados, el arma, el equipo, el nivel, etc.

De este tipo de juegos, los juegos de "Square-Enix" son muy conocidos, en concreto "Final Fantasy" y "Kingdom Hearts". En el juego "Final Fantasy VII – Advanced Children", la pantalla es divida en 2 partes, parte "activa" por la izquierda– la selección de arma, parte "passiva" por la derecha – estado del personaje. La división es visible por los dos colores usados en cada parte. Esta variante de la división es útil para el diseño del teclado "inteligente". Además el menú dispone de un mensaje de ayuda en la parte baja de la pantalla, como se muestra en la **Figura 41**

Figura 41: Capturalla de patalla del menú de "Final Fantasy VII - Advanced Children".

Como en el juego "Final Fantasy VII – Advend Children", también "Kingdom Hearts – Birth by Sleep" posee de una división de la pantalla y de un "asistente" / mensaje de ayuda. Además en el menú se muestra el tiempo total de jugar el juego. Para un diccionario, especialmente en la parte de "juegos / exámenes" estaría bien saber cuantos puntos se han sacado en un examen o cuántos segundos quedan para resolver uno. Para animar a los niños sería importante crear un sistema de "sube nivel" típico en los "RPG". Se muestra las técnicas con iconos. Las técnicas de usar magia, se muestra con un gorro de mago. Si son técnicas de fuerza, como por ejemplo de combate, se muestra con una espada. En la **Figura 42**, se puede ver por la derecha el nivel de la técnica y el tipo de la

técnica. Si se ha pasado todos los niveles sale una corona, si no queda una barra pequeña para mostrar cuantos puntos son necesarios para sacarla.

Figura 42: Caputralla de pantalla del menú de "Kingdom Hearts - Birth by Sleep".

Para facilitar el uso del diccionario para niños, se ha añadido también iconos en los ítems de menú. Además, para llamar la atención, la mascota interviene indirectamente con el usuario. La **Figura 43**, muestra el boceto del menú principal para el diccionario con la mascota "Profesor Tobí".

Figura 43: Boceto del menú principal.

Para el menú del diccionario, también interviene la mascota con el usuario. Antes fue planificado tener en la barra arriba una información adicional sobre todos los botones que se puede usar en esta parte. Para no hacer el diccionario muy complicado, no se muestra dicha información en la versión actual. En la **Figura 44** se muestra el boceto del menú del diccionario.

Figura 44: Boceto del menú del dictionario.

El cambio que se ha hecho de la versión alfa a la versión actual se muestra en **la Figura 46**. En la muestra falta la implementación de la mascota. En la **Figura 45**, se puede ver la implementación planificada en el menú.

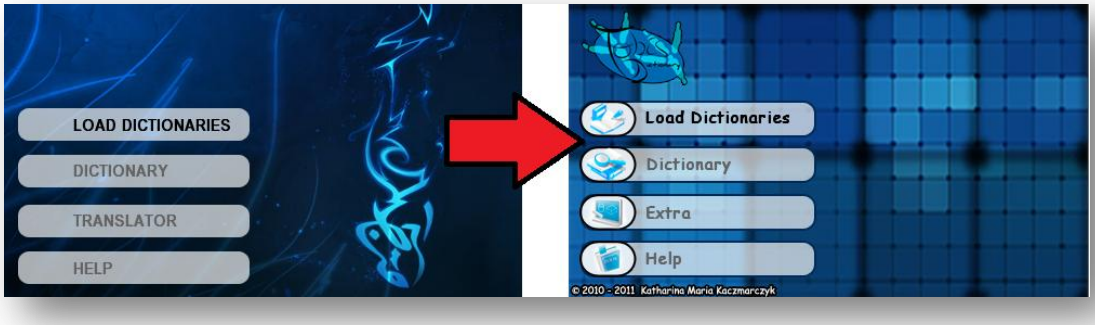


Figura 45: Cambio del menú principal.

Figura 46: Ejemplo de la implementación de la mascota en el menú.

El orden de los ítems en el menú "Dictionary" sigue siendo lo mismo. La diferencia consiste en el diseño, como se ve en la **Figura 47**.

Figura 47: Cambio del menú "Dictionary".

Como se ha cometado antes, la versión alfa tiene la restricción de un máximo de 10 diccionarios. Aparte de esto, el diccionario debería "saber" si hay un diccionario en la aplicación o no. Mejor dicho, en el caso de que si no hay ningún diccionario, el usuario debería poder cargar uno directamente a través de la conexión USB. Asimismo, la aplicación debería mostrar un icono de la bandera (como por ejemplo la bandera de Alemania y España, para saber que el diccionario es de alemán a español). En la **Figura 48** se muestra la diferencia entra la versión alfa y la versión actual sin las implementaciones adicionales.

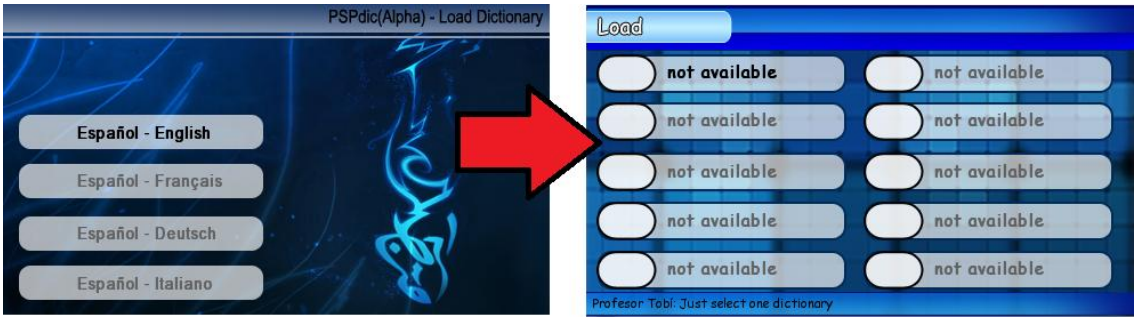


Figura 48: Cambio del menú "Load Dictionary".

Programar un menú:

Dependiendo del caso, un menú simple se puede programar como se ha mostrado en el capítulo "SDL-menú sin clases" o también "SDL menú con clases".Lo más importante consiste en la lógica del menú. Quiere decir, si se cambia el color de una capa de texto, es importante cambiar el color de la capa de texto anterior. Si no, la capa de texto anterior no va a cambiar el color.

5.2 OSK – propio teclado por pantalla

La OSK original de PSP está disponible en el SDK de MinPSPw en el fichero "psputilit_osk.h"[24]. Existen problemas con las librerías gráficas de PSPGU y SDL. También existe un teclado de SDL para la PSP que fue creado por Danzeff[25]. Para hacer el diccionario más inteligente fue necesario añadir una lista de sugerencias en la misma pantalla del teclado móvil. Con esta opción, el manejo del diccionario se ha mejorado un poco, porque el usuario tiene la posibilidad de ver directamente, si el diccionario ya ha encontrado una palabra igual o similar a la palabra que se esté buscando. Asimismo, el usuario sabe también si se ha equivocado o no. Quiere decir, el error de equivocarse se reduce aproximadamente a 99%. Como en la versión alfa, la versión actual dispone de dos teclados. El teclado "inteligente" (teclado teléfono + lista de sugerencias) y el teclado "qwerty". Dependiendo del caso, puede ser más fácil escribir la palabra en el teclado "inteligente" o en el teclado "qwerty", sin tener en cuenta las sugerencias. El diccionario debe ofrecer la posibilidad de cambiar entre ambos teclados sin problemas (móvil ↔ qwerty) en el tiempo de escritura. En el caso de PSP, el diccionario también debería ofrecer en el teclado "inteligente" un sistema de IntelliSense[26].

Debido al resultado del análisis, la pantalla de la PSP fue dividida en dos partes. La parte "activa" por la izquierda - donde se escribe la palabra. La parte pasiva por la derecha - donde se muestra las sugerencias. La parte "activa" rellena la pantalla por $\frac{1}{3}$, los $\frac{2}{3}$ son para la parte "pasiva", porque las sugrencias pueden llevar muchas letras como por ejemplo la palabra en alemán: "Betriebswirtschaftslehre"[27], que contiene 24 letras. El dato sobre la cantidad de letras en una palabra también es importante para indicar la longitud máxima de una palabra. En este punto influyen dos aspectos. Uno de ellos es la pregunta estadística - ¿cuántas veces escribe un niño una palabra de la longitud ×? o ¿Cuál es la longitud de palabras que usa un niño en su vida depeniendo del idioma?

El otro punto consiste en el diseño gráfico. Quiere decir, cuantas letras puede mostrar el diccionario por pantalla sin problemas. En este caso, depende de la letra y de la fuente que usa la aplicación. Para entenderlo mejor, una "i" es más pequeña que una "m" en el tema de representación. Por ejemplo una cadena de 5 caracteres del mismo tipo. Para "i" = "iiiii" y para "m" = "mmmmm". Para ver mejor la comparación y el resultado entre ambos caracteres, es necesario escribirlas en dos filas.

[24] En el caso de usar la OSK original de PSP, el emulador de la PSP no lo muestra por pantalla.

[25] El teclado de Denzeff se puede descargar en la página web: http://www.qj.net/psp/homebrew-applications/danzeff-oslib-modified-version-of-the-psp-virtual-keyboard-app.html

[26] IntelliSense – se refiere al sistema de IntelliSense de VS, que ofrece directamente las funciones y/o variables, las que se puede usar. En este caso, las letras, las que se puede usar en la tecla. P.ej. la tecla "abc" ofrece las letras: "abcABCáàäÁÀÄ"

[27] "Betrwirswirtschaftslehre" significa en español "economía" o "industrial"

Resultado de la comparación de "m" e "i" en una cadena de la longitud 5.

- "mmmmm"
- "iiiii"
- mimimimimi

Es claramente visible, que no es posible meter la longitud de caracteres con solo un carácter. Debido al resultado de la comparación, la longitud máxima de caracteres que una aplicación puede mostrar, depende de varios factores (tipo de carácter, fuente, palabra, idioma). En el diccionario la longitud máxima es de _____ caracteres.

En el boceto de la GUI para el teclado "inteligente", es visible la división de $\frac{1}{3}$ para los botones del teclado y su campo de IntelliSense. Los otros $\frac{2}{3}$ de la pantala sirven para el campo de texto, las sugerencias y los botones "especiales" del teclado. En la **Figura 49**, se muestra claramente.

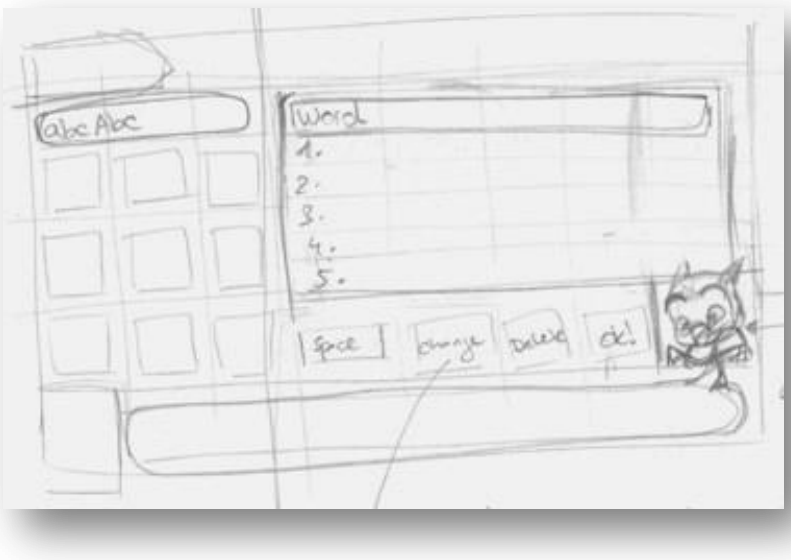


Figura 49: Boceto del teclado "inteligente".

La diferencia entre el teclado de la versión alfa y el teclado "inteligente" de la versión actual es muy elevada. En la versión alfa, el diseño no formaba una parte tan importante como en la versión actual. Especialmente en el campo de texto se nota la falta de importancia del diseño. Asimismo, no fueron hechos los análisis de diccionarios para las videoconsolas. El cambio de los botones fue importante, para que el diccionario parezca más profesional. Lo mismo pasaba con el teclado "qwerty". Como en el primero teclado, el campo de texto es demasiado elevado. Además, en la versión alfa el teclado no es "qwerty" si no "qwertz", porque el teclado alemán es "qwertz". Este fallo fue notado en el

proceso de cambiar el diseño de los dos teclados del diccionario. En la versión actual, el mensaje de ayuda cambia dependiendo del caso. Por lo contrario, la versión alfa no lo dispone. Todos estos detalles se pueden ver en la **Figura 50** y la **Figura 51** .

Figura 50: Cambio del teclado móvil de la versión alfa a la versión actual.

Figura 51: Cambio del teclado "qwerty" de la versión alfa a la versión actual.

Programación del propio teclado

Un teclado es en realidad como un menú de la forma de una matriz. Dependiendo del teclado, la matriz puede ser cuadrada o una matriz de m × n. En el caso del teclado "inteligente", la matriz es cuadrada, porque es posible saltar de los botónes "ghi", "pqrs" y ".,?!" al botón "OK!". En el caso contrario, en el teclado "qwerty", la matriz es de m× n. La m es para el número total de elementos en columna, la n sirve para el número total de elementos en fila.

Como se ve en la **Figura 51**, la última fila consiste de solo tres elementos ("Shift", "Space", "Change"), aunque las filas anteriores consisten de once elementos (p.ej. QWERTYUIOP?). En este caso, en la programación se junta los elementos de las filas anteriores en un conjunto. Quiere decir, el elemento de la columna1 y columna2

corresponden al conjuto1 (“Shift”), los elementos de la columna3 hasta la columna9 corresponden al conjunto2 (“Space”) y los elementos de la columna10 y de la columna11 corresponden al conjunto3 (“Change”). La **Figura 52** ilustra la matriz de m× n del teclado “qwerty” y también el concepto de conjuntos.

Figura 52: Ejemplo de la matriz del treclado "qwerty".

Como se ha comentado antes, la matriz del teclado “inteligente” es cuadrado (n× n). Quiere decir, que el número de elementos en la fila y en la columna son iguales. En el teclado “inteligente”, no es perceptible que la matriz sea cuadrada, porque no existe ninguna conexión visible del elemento de la columna3 al elemento de la columna4. En el tema de programación se “crea” una columna fantasma para realizar dicha conexión entre los elementos de la columna3 y el elemento de la columna4.

La **Figura 53** ilustra las conexiones invisibles de las columnas.

Figura 53: Ejemplo de la matriz del teclado "inteligente".

Para saber que botón fue apretado, es necesario tener dos contadores - para la posición de la fila y para la posición de la columna. La estructura lógica consiste de dos "switch-cases". El primer "switch-case" sirve para las filas, el segundo "switch-case" para las columnas – como una matriz.

La estructura va a ser de esta menara:

```
switch(ArribaAbajo)
{
  case 1: //fila1
  {
        switch(IzquierdaDereacha)
        {
              case 1: //fila1, columna1
              {
                    ...
                    break;
              }

              ...

              case m:
              {
                    ...
                    break;
              }

        }//cerrar switch-case de las columnas
        break;
  }//cerrar fila1

  ...

  case n: //fila n
  {
        switch(IzquierdaDereacha)
        {
              case 1:
              {
                    ...
                    break;
              }

              ...

              case m:
              {
                    ...
                    break;
              }
        }//cerrar switch-case de las columnas
        break;
  }//cerrar fila n
}//cerrar primer switch-case de las filas
```

5.3 Diccionario - Fichero XML

Esta parte del capítulo "Diseño" muestra la arquitectura de un fichero XML en el diccionario. Para saber en qué coniste el diccionario, es importante ver la estructura de un diccionario electrónico, sería una página web o un diccionario para el ordenador. Como comentado, el SDK MinPSPw dispone de TinyXML que se usa en la parte de programación. El ejemplo de TinyXML está escrito en C++. Este es uno de los motivos, porque se ha desarrollado el proyecto en C++ en vez de C. Antes de todo es obstante saber cómo utilizar TinyXML en general y luego en la PSP.

El fichero XML para el diccionario corresponde de la arquitectura de la **Figura 54**:

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <Dictionary>
4      <language type ="DE-ENG">
5          <word key="blöd">
6              <type>ADJ</type>
7              <type>fam</type>
8              <spell>[ˈblø:də]</spell>
9
10             <translation>stupid</translation>
11             <translation>fool</translation>
12             <translation>dumb</translation>
13
14             <example>das ist ja vielleicht ein blöder Kerl!</example>
15             <exampleT>he really is a nasty piece of work! </exampleT>
16         </word>
17     </language>
18
19 </Dictionary>
```

Figura 54: Ejemplo de la arquitectura del fichero XML.

Dicho fichero sirve como un sistema de información en el diccionario. Quiere decir, sin este fichero el diccioanario no funciona como un diccionario. Mejor dicho, el usuario no tiene la posibilidad de buscar la traducción de una palabra a otra palabra, porque falta la información. Sin el fichero es posible usar el diccionario y escribir palabras, pero no puede buscarlas. Además sin el fichero, no es posible crear los juegos de tipo mini examen. En la estructura del fichero XML fue obstante comprobrar qué compontenes dispone un diccionario. En concreto, la estructura corresponde del diccionario de "PONS" con el tipo de la palabra (substantivo, adjetivo,etc.), de qué tipo corresponde (p.ej. familiar), cómo se pronuncia. Cada palabra tiene varias traducciones y a veces ejemplos de cómo usar la

palabra en una frase. El fichero debería tener un máximo de 20.000 ó 30.000 palabras, porque un niño no usa en su vida muchas palabras. Otra variante de un diccioanario, un diccionario que muestra las 1000 palabras más usadas en un campo especial, como por ejemplo los diccionarios de "PONS" de la serie: "Die 1000 wichtigsten Wörter"[28]. Por estos motivos el fichero del diccionario no debería ser de gran capacidad. Quiere decir, si por ejemplo dicho ejemplo del fichero de XML contiene 1000 entradas, el tamaño es aproximadamente de 356 Kbytes. En este caso si el usuario tiene 10 diccionarios, entonces el tamaño total del sistema de información estaría aproximadamente de 356 Kbytes * 10 = 3560 Kbytes / 1024 = 3,4765625 Mbytes ≅ 3,48 Mbytes. El "Homebrew" necesita de momento ≅ 14,97 Mbytes (imágenes, sonido, ejecutable) + 3,48 Mbytes (10 diccionarios de 1000 entradas con un tamaño de 356 Kbytes) ≅ 18,45 Mbytes. Entonces quedan 100 Mbytes – 18,45 Mbytes ≅ 81,55 Mbytes para sonidos, imágenes y/u otro ficheros necesarios.

En el caso de que si el número de entradas es de 20.000 palabras de dicho ejemplo, el tamaño es aproximadamente de 7053829 Bytes = 6888.5048828125 Kbytes = 6.72705554962158203125 Mbytes ≅ 6.73 Mbytes. Si el usuario lleva 10 diccionarios, entonces el tamaño total del sistema de información sería aproximadamente de 6.73 Mbytes * 10 ≅ 67,30 Mbytes. El "Hombrew" va a ser del mismo tamaño, entonces la aplicación corresponde a una capacidad de 67,30 Mbytes + 14,97 Mbytes ≅ 82.27 Mbytes. Quedan 100 Mbytes – 82.27 Mbytes = 17.73 Mbytes para sonidos, imágenes y/u otro ficheros necesarios.

En el último caso, si el número de entradas es de 30.000 palabras, el tamaño es aproximadamente de 10574668 Bytes = 10326.82421875 Kbytes = 10.084789276123046875 Mbytes ≅ 10.08 Mbytes. Si la aplicación ofrece 10 diccionarios, entonces el tamaño total del sistema de información sería aporximadamente de 10,08 Mbytes * 10 = 100.80 Mbytes. En este caso, el sistema de información supera el tamaño máximo de una aplicación "Minis". El "Hombrew" va a ser del mismo tamaño, entonces la aplicación corresponde a una capacidad de 100.80 Mbytes + 14,97 Mbytes ≅ 115.77 Mbytes.

El mejor caso, si los ficheros del diccionario contienen 1000 palabras. La pregunta es, si en la capacidad total influyen también los diccionarios o no. Si no, sería mejor ofrecer un diccionario de 30.000 palabras para niños y/o alumnos. En el caso de que sí, la solución consiste en reducir el número máximo de diccionarios en la aplicación. Por ejemplo de máximo 10 diccionarios a máximo 5 diccionarios. Esto significa que el tamaño del sistema se reduce a la mitad, en concreto en el ejemplo 2 de un sistema de información de 67,30 Mbytes a 33,65 Mbytes. El tamaño total sería

[28] Traducción : "Las 1000 palabras más importantes"

49,01 Mbytes ≅ casi la mitad del tamaño máximo en el caso anterior (10 diccionarios). El tamaño se ha reducido de 82.27 Mbytes – 49,01 Mbytes ≅ 33,26 Mbytes, en porcentaje ≅ 40.43 %.

En la **Figura 55** se ilustra una caputra de pantalla del símbolo del sistema de Windows 7 que muestra los tamaños de los ficheros ejemplos de XML, para entender los cálculos realizados para saber la cantidad de palabras ideales del diccionario para la aplicación.

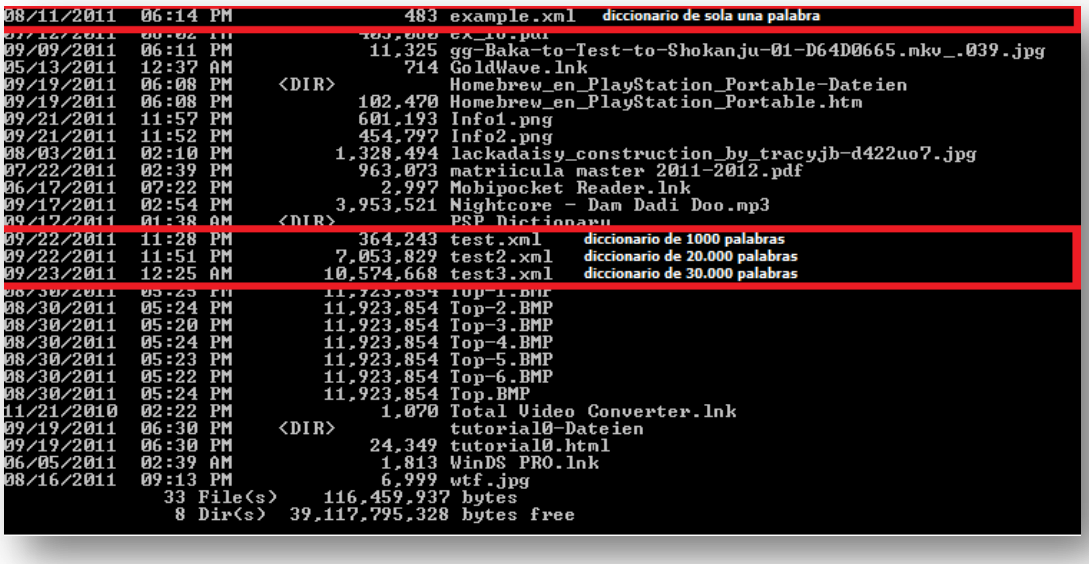


Figura 55: Captura de pantalla del símbolo del sistema de los tamaños de ficheros xml.

La **Figura 56**, muestra el cálculo del tamaño de la aplicación en el día 23 de septiembre con los datos del símbolo del sistema sobre los tamaños de cada fichero. Este número es importante para poder calcular el número de Mbytes que quedan disponibles para añadir más gráficos, sonidos u otros ficheros necesarios. El objetivo general es desarrollar una aplicación "Minis". Entonces, el tamaño máximo de la aplicación no debería superar los 100 Mbytes.

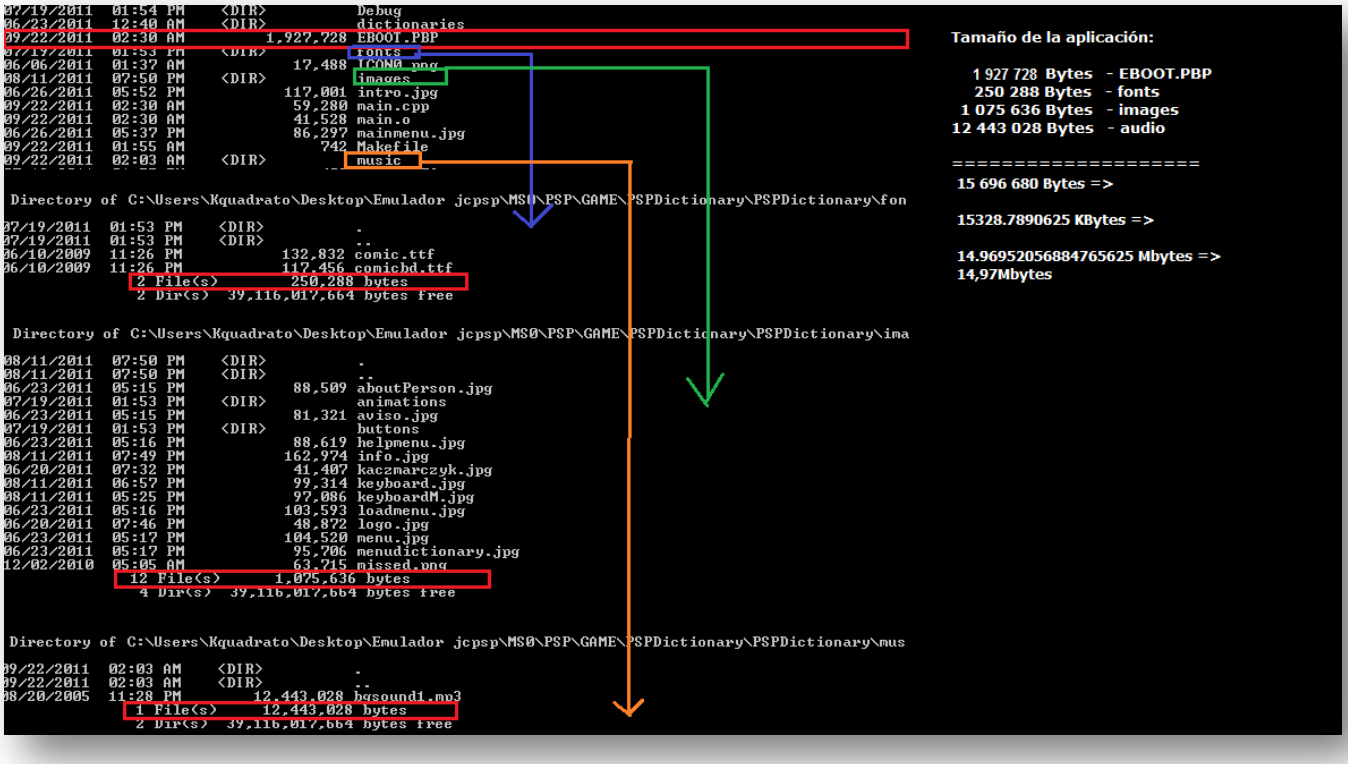


Figura 56: Calculo del tamaño de la aplicación en el día "23/09/2011".

Programación

El SDK MinPSPw ofrece TinyXML. El ejemplo dado de código fuente muestra el uso de TinyXML en Windows como se puede encontrar en muchos tutoriales por internet. Sin embargo si se quiere programar con XML en la PSP hay que hacer dos cosas.

Los includes:

```
#ifdef TIXML_USE_STL
  #include <iostream>
  #include <sstream>
  using namespace std;
#else
  #include <stdio.h>
#endif

#include <pspkernel.h>
#include <pspdebug.h>
#include <pspdisplay.h>
#include "tinyxml.h"
```

"TIXML_USE_STL" se usa para el caso de "string".

Para cargar el fichero XML y crear el árbol de DOM se ha creado la función "dump_to_stdout". Dicha función en realidad debería imprimir también el árbol de DOM. Por no saber cómo realizarlo en la PSP, se ha dejado la parte de XML. Además falta el conocimiento de "parse" un fichero XML a un "string".

Es posible compilar el código y generar el EBOOT.PBP. En la PSP se va a ver los dos mensajes de la consola ("**start Reading document**", "**start printing").

```
void dump_to_stdout(const char* pFilename)
{
  TiXmlDocument doc(pFilename);
  bool loadOkay = doc.LoadFile();
  if (!loadOkay)
  {
            printf("Failed to load file \"%s\"\n", pFilename);
  }

  else
  {

            printf("**start reading document**\n\n");
            printf("**start printing**\n\n");
            doc.Print (stdout);
          }
```

El Makefile:

```
TARGET = XMLsample

PSPSDK = C:/pspsdk/psp/sdk
PSPBIN = C:/pspsdk/bin

OBJS = main.o

CFLAGS = -O0 -G0 -Wall -g

CXXFLAGS = $(CFLAGS) -fno-exceptions -fno-rtti
ASFLAGS = $(CFLAGS)

LIBS=  -ltinyxml -lstdc++

EXTRA_TARGETS = EBOOT.PBP
PSP_EBOOT_TITLE = XML sample

include $(PSPSDK)/lib/build.mak
```

6. Programación

Este capítulo es el núcleo de toda la memoria del estudio. Contiene todos los resultados de las pruebas y los análisis juntados en un proyecto. Al principio se intentó crear una arquitectura de programa en papel con varios diagramas de clase. Por motivos de falta de experiencia de desarrollo no fue posible desarrollar el proyecto primero en papel y luego por ordenador. Con la ayuda de VS, fue posible crear los diagramas de clase sin problemas. Fue necesario instalar el "Visual Studio 2010 Visualization & Modeling Feature Pack (x86 and x64) - (English)"[29] para poder usar todas las funciones disponibles de crear diagramas directamente en VS.

6.1 Arquitectura

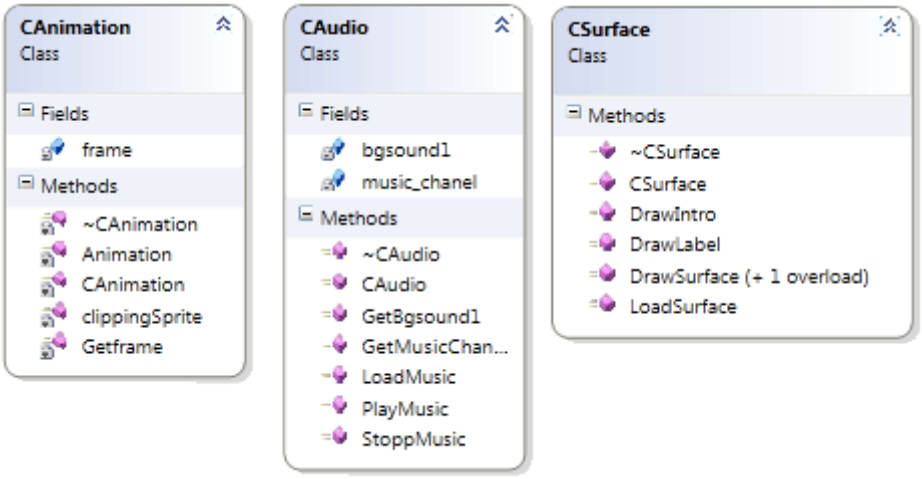
La arquitectura es el núcleo del programa e indica si un programa está bien escrito o no. En la versión alfa del diccionario, la arquitectura del programa es fatal, porque se ha copiado la mala estructura del proyecto "Metro Valencia" de Anush Euredjian Chiappe y Juan Cortés Maiques[2]. La arquitectura de "Metro Valencia" está estable en el caso de dibujar pocos elementos. En el caso contrario, el número de las líneas de código sube muchísimo. Las funciones de dibujar el menú son siempre las mismas. La única diferencia entre dichas funciones consiste en el valor de color. Cuando el menú contenga muchos ítems, entonces hubo que crear la misma cantidad de funciones como ítems. Esto provoca que la ejecución de dibujar los menús necesite mucho tiempo. Para un diccionario es inevitable que la búsqueda de la palabra sea rápida. También es importante en el caso de la PSP, la velocidad de imprimir el menú, el teclado y la palabra escrita por el teclado.

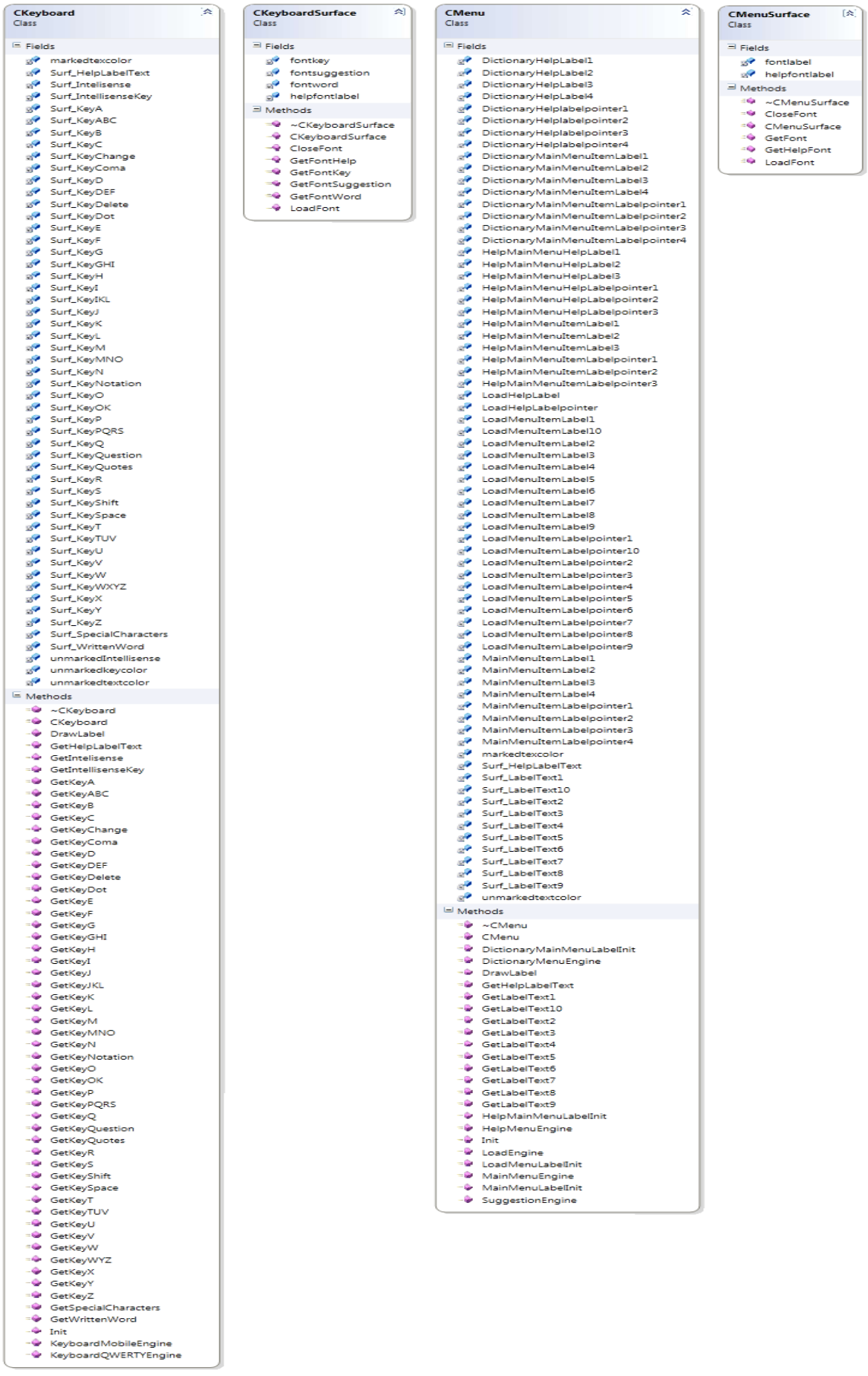
La versión actual muestra la mejor velocidad de dibujar los menús, los teclados e imprimir la palabra escrita por el teclado. Aproximadamente, la versión actual es 50% más rápida que la versión alfa por el cambio de la arquitectura del programa. Además, la versión actual no usa muchas veces la memoria como la versión alfa, porque la PSP ya tiene todas las imágenes cargada en la memoria y sabe en qué posición tiene que acceder para volver a dibujar el menú o el teclado.

[29] Disponible gratis en MSDNAA

6.2 Diagramas de Clase

El proyecto contiene varias clases. En el capítulo "SDL menú con clases" ya se ha visto las clases importantes para crear un menú. Para el teclado se ha creado más o menos las mismas clases, porque la lógica del teclado corresponde a la lógica del menú. Asimismo se ha creado 2 clasesmás. CAudio sirve para reproducir una pista de audio. CAnimation es para la implementación de animaciones en el proyecto. Por la falta de tiempo, no fue posible realizar la implementación final.

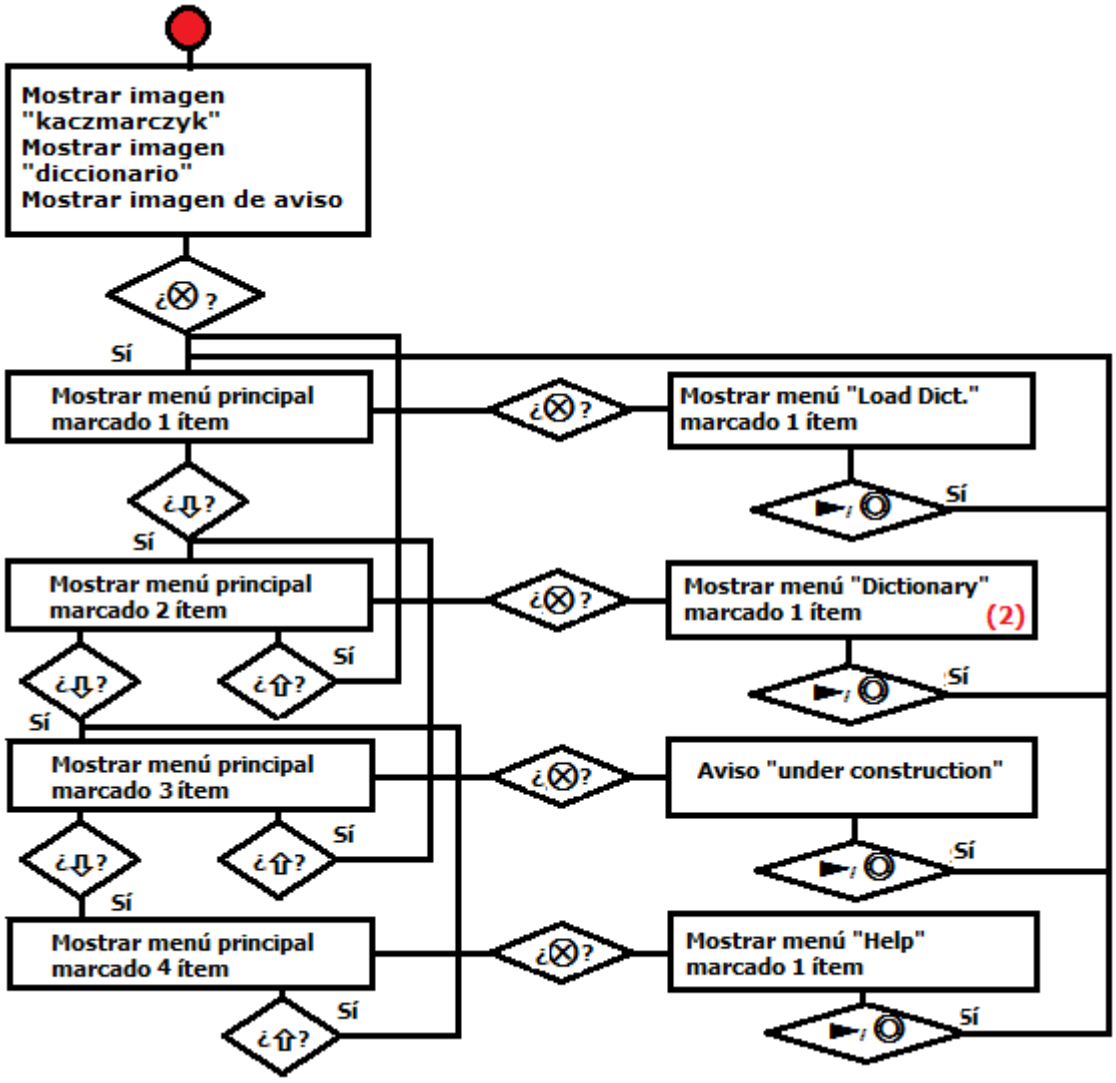


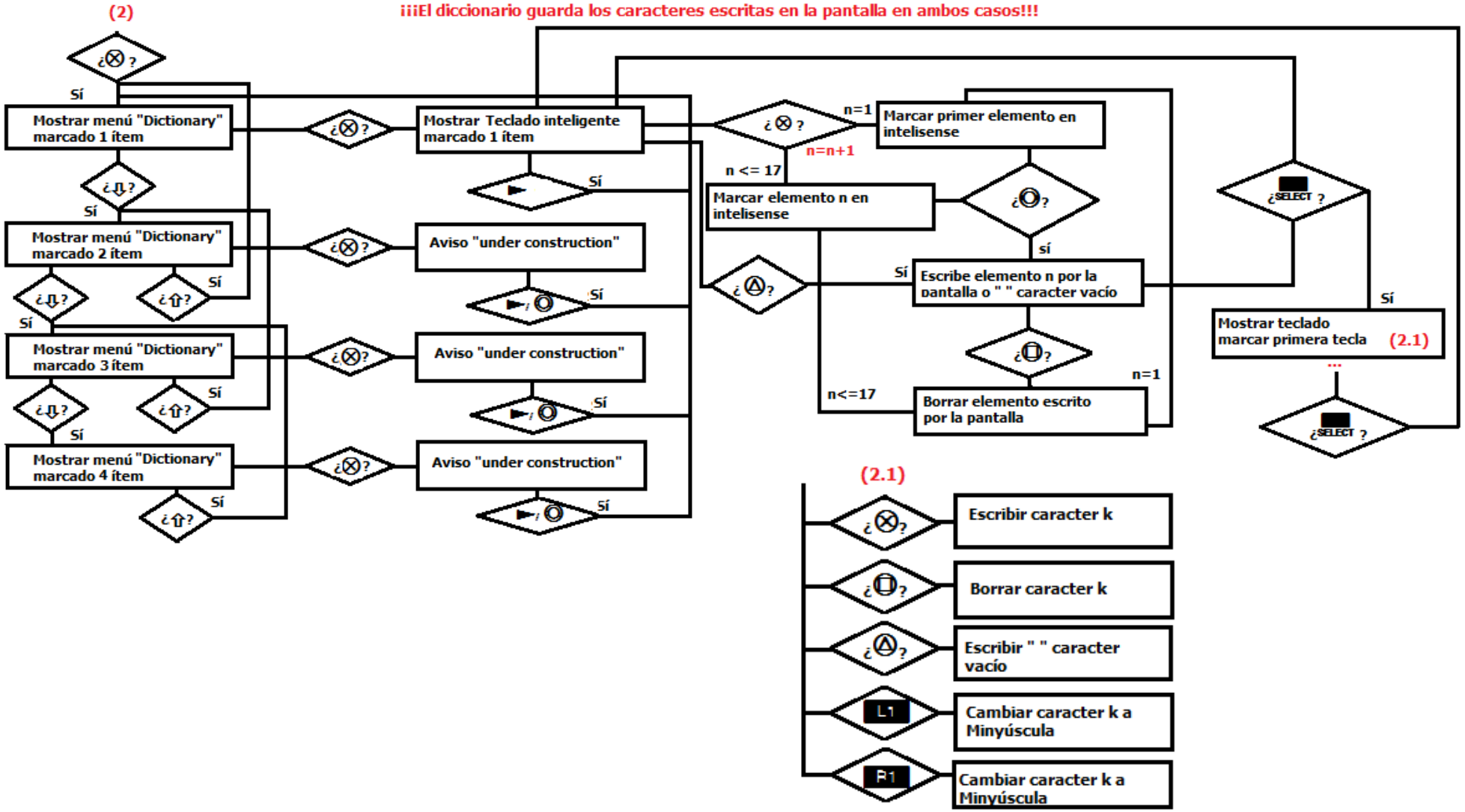

CKeyboard
Class
Fields
markedtexcolor
Surf_HelpLabelText
Surf_Intellisense
Surf_IntellisenseKey
Surf_KeyA
Surf_KeyABC
Surf_KeyB
Surf_KeyC
Surf_KeyChange
Surf_KeyComa
Surf_KeyD
Surf_KeyDEF
Surf_KeyDelete
Surf_KeyDot
Surf_KeyE
Surf_KeyF
Surf_KeyG
Surf_KeyGHI
Surf_KeyH
Surf_KeyI
Surf_KeyIKL
Surf_KeyJ
Surf_KeyK
Surf_KeyL
Surf_KeyM
Surf_KeyMNO
Surf_KeyN
Surf_KeyNotation
Surf_KeyO
Surf_KeyOK
Surf_KeyP
Surf_KeyPQRS
Surf_KeyQ
Surf_KeyQuestion
Surf_KeyQuotes
Surf_KeyR
Surf_KeyS
Surf_KeyShift
Surf_KeySpace
Surf_KeyT
Surf_KeyTUV
Surf_KeyU
Surf_KeyV
Surf_KeyW
Surf_KeyWXYZ
Surf_KeyX
Surf_KeyY
Surf_KeyZ
Surf_SpecialCharacters
Surf_WrittenWord
unmarkedIntellisense
unmarkedkeycolor
unmarkedtextcolor
Methods
~CKeyboard
CKeyboard
DrawLabel
GetHelpLabelText
GetIntelisense
GetIntellisenseKey
GetKeyA
GetKeyABC
GetKeyB
GetKeyC
GetKeyChange
GetKeyComa
GetKeyD
GetKeyDEF
GetKeyDelete
GetKeyDot
GetKeyE
GetKeyF
GetKeyG
GetKeyGHI
GetKeyH
GetKeyI
GetKeyJ
GetKeyJKL
GetKeyK
GetKeyL
GetKeyM
GetKeyMNO
GetKeyN
GetKeyNotation
GetKeyO
GetKeyOK
GetKeyP
GetKeyPQRS
GetKeyQ
GetKeyQuestion
GetKeyQuotes
GetKeyR
GetKeyS
GetKeyShift
GetKeySpace
GetKeyT
GetKeyTUV
GetKeyU
GetKeyV
GetKeyW
GetKeyWYZ
GetKeyX
GetKeyY
GetKeyZ
GetSpecialCharacters
GetWrittenWord
Init
KeyboardMobileEngine
KeyboardQWERTYEngine
CKeyboardSurface
Class
Fields
fontkey
fontsuggestion
fontword
helpfontlabel
Methods
~CKeyboardSurface
CKeyboardSurface
CloseFont
GetFontHelp
GetFontKey
GetFontSuggestion
GetFontWord
LoadFont
CMenu
Class
Fields
DictionaryHelpLabel1
DictionaryHelpLabel2
DictionaryHelpLabel3
DictionaryHelpLabel4
DictionaryHelplabelpointer1
DictionaryHelplabelpointer2
DictionaryHelplabelpointer3
DictionaryHelplabelpointer4
DictionaryMainMenuItemLabel1
DictionaryMainMenuItemLabel2
DictionaryMainMenuItemLabel3
DictionaryMainMenuItemLabel4
DictionaryMainMenuItemLabelpointer1
DictionaryMainMenuItemLabelpointer2
DictionaryMainMenuItemLabelpointer3
DictionaryMainMenuItemLabelpointer4
HelpMainMenuHelpLabel1
HelpMainMenuHelpLabel2
HelpMainMenuHelpLabel3
HelpMainMenuHelpLabelpointer1
HelpMainMenuHelpLabelpointer2
HelpMainMenuHelpLabelpointer3
HelpMainMenuItemLabel1
HelpMainMenuItemLabel2
HelpMainMenuItemLabel3
HelpMainMenuItemLabelpointer1
HelpMainMenuItemLabelpointer2
HelpMainMenuItemLabelpointer3
LoadHelpLabel
LoadHelpLabelpointer
LoadMenuItemLabel1
LoadMenuItemLabel10
LoadMenuItemLabel2
LoadMenuItemLabel3
LoadMenuItemLabel4
LoadMenuItemLabel5
LoadMenuItemLabel6
LoadMenuItemLabel7
LoadMenuItemLabel8
LoadMenuItemLabel9
LoadMenuItemLabelpointer1
LoadMenuItemLabelpointer10
LoadMenuItemLabelpointer2
LoadMenuItemLabelpointer3
LoadMenuItemLabelpointer4
LoadMenuItemLabelpointer5
LoadMenuItemLabelpointer6
LoadMenuItemLabelpointer7
LoadMenuItemLabelpointer8
LoadMenuItemLabelpointer9
MainMenuItemLabel1
MainMenuItemLabel2
MainMenuItemLabel3
MainMenuItemLabel4
MainMenuItemLabelpointer1
MainMenuItemLabelpointer2
MainMenuItemLabelpointer3
MainMenuItemLabelpointer4
markedtexcolor
Surf_HelpLabelText
Surf_LabelText1
Surf_LabelText10
Surf_LabelText2
Surf_LabelText3
Surf_LabelText4
Surf_LabelText5
Surf_LabelText6
Surf_LabelText7
Surf_LabelText8
Surf_LabelText9
unmarkedtextcolor
Methods
~CMenu
CMenu
DictionaryMainMenuLabelInit
DictionaryMenuEngine
DrawLabel
GetHelpLabelText
GetLabelText1
GetLabelText10
GetLabelText2
GetLabelText3
GetLabelText4
GetLabelText5
GetLabelText6
GetLabelText7
GetLabelText8
GetLabelText9
HelpMainMenuLabelInit
HelpMenuEngine
Init
LoadEngine
LoadMenuLabelInit
MainMenuEngine
MainMenuLabelInit
SuggestionEngine
CMenuSurface
Class
Fields
fontlabel
helpfontlabel
Methods
~CMenuSurface
CloseFont
CMenuSurface
GetFont
GetHelpFont
LoadFont

6.3 Diagramas de Flujo

En el primer diagrama de flujo se muestra el manejo de botones en el menú principal. Antes de entrar en el menú principal, la PSP requiere la aceptación del aviso. En el menú principal se puede moverse con los botones arriba y abajo para seleccionar luego con ⊗ el ítem para entrar en unos de los funciones (“Load Dictionaries”, “Dictionary”, “Extra”, “Help”). Para volver en el menú principal, se debe apretar ◎ o START .

En el segundo diagrama de flujo, se analiza el manejo del apartado “Dictionary”. En esta parte influye también el manejo del teclado. Para no hacer el diagrama de flujo demasiado grande se muestra solo el manejo de las teclas en el primer caso en ambos teclados.




(2)
¡¡¡El diccionario guarda los caracteres escritas en la pantalla en ambos casos!!!
¿⊗?
Sí
Mostrar menú "Dictionary" marcado 1 ítem
Mostrar menú "Dictionary" marcado 2 ítem
Mostrar menú "Dictionary" marcado 3 ítem
Mostrar menú "Dictionary" marcado 4 ítem
Mostrar Teclado inteligente marcado 1 ítem
Aviso "under construction"
Aviso "under construction"
Aviso "under construction"
n=1
n=n+1
n <= 17
Marcar primer elemento en intelisense
Marcar elemento n en intelisense
Escribe elemento n por la pantalla o " " caracter vacío
Borrar elemento escrito por la pantalla
n<=17
¿SELECT ?
Mostrar teclado marcar primera tecla (2.1)
(2.1)
Escribir caracter k
Borrar caracter k
Escribir " " caracter vacío
Cambiar caracter k a Minyúscula
Cambiar caracter k a Minyúscula
K = tecla actual del teclado QWERTY

124

6.4 Código

Como se puede ver en los diagramas de flujo, el proyecto ya posee de muchas funciones. En esta parte no se va a mostrar todo el código, porque el proyecto tiene aproximadamente 8000 líneas de código. Se ilustra unos trozos de código para enseñar cómo se ha programada unas cosas como por ejemplo escribir por pantalla, borrar un carácter o cambiar el carácter de minúscula a mayúscula o atraves.

Versión alfa:

En la versión alfa fue posible buscar dentro de un fichero *.txt una palabra y muestra la traducción por pantalla. Para poder hacer esto en la PSP es necesario cargar un fichero *.txt en la memoria de PSP.

Definir global:

```
char fpath[200];
char read_buffer[128*1024];
char write_buffer[128*1024];

const char *fileEspEng = "/dic/Espanol/en";
const char *fileEspFr = "/dic/Espanol/fr";
const char *fileEspIt = "/dic/Espanol/it";
const char *fileEspGer = "/dic/Espanol/ger";
```

La función de leer un fichero en la PSP, necesario para un diccionario:

```
void readfile(char *file)
{
    int fd; //fileid
    int offset=0;
    int bytesread=0;
    int byteswrote =0;

    int i;

    sprintf(fpath, "%s.txt", file);

    // Open the file for reading only
    if(!(fd = sceIoOpen(fpath, PSP_O_RDONLY, 0777))) {/*ERROR*/}
    else {
        bytesread = sceIoRead(fd, read_buffer, sizeof(read_buffer));
        }
    sceIoClose(fd);
}
```

La función de escribir en un fichero en la PSP, necesario para guardar la búsqueda. Se guarda la búsqueda solo sí, se ha realizado una llamada al menú "result":

```
void createSavefile(char text[])
{
   int fd; //fileid
   int offset=0;
   int byteswrote=0;

   int i;

   sprintf(fpath, "%s.txt", "/dic/History");

   // Open the file for writing only and append it to the end of the file, create the file if it doesnt
exist.
   if(!(fd = sceIoOpen(fpath, PSP_O_WRONLY|PSP_O_APPEND|PSP_O_CREAT, 0777))) {/*Error
*/}
   else {
         text[strlen(text)]= ';';
         sprintf(write_buffer, text);
         byteswrote = sceIoWrite(fd, write_buffer, strlen(text));
         text[strlen(text)-1]= NULL;
      }
   sceIoClose(fd);
}
```

La función de búsqueda en la versión alfa solo funciona con ficheros *.txt "especial". Quiere decir, que el fichero *.txt tiene que tener la siguiente estructura para poder buscar en él:

```
coche;Auto;madre;Mutter;padre;Vater;hermano;Bruder;hermana;Schwester;guay;cool;jueves;Donn
erstag;viernes;Freitag;miércoles;Mittwoch;suspender;durchfallen;aprobar;bestehen;ejemplo;Beispie
l;
```

La función de búsqueda:

```
void searchDic()
{
   char *raux = &read_buffer; //exampletext
   char *saux = &text3; //searchtext
   char *taux = &tResD; //translation

   char *pEntry = strstr(raux,saux);

   if (pEntry == NULL){tResD[0]='0';}
   else {
      int size = strlen(saux);
      pEntry = pEntry + size + 1;
      char *pTransending = strchr(pEntry,';');
      int tsize = pTransending - pEntry;
      char *pSuccess = strncpy(taux,pEntry,tsize);

      tResD[tsize]= '\0';
   }
}
```

Para cambiar el diccionario es obstante borrar el "buffer" antes de cargar el nuevo fichero. La función para borrar el buffer es el siguiente:

```
void deleteBuffer()
{
   int I;
   for(i=0;i<=strlen(read_buffer);i++) {read_buffer[i] = NULL;}
}
```

Para cambiar el diccionario se debe hacer lo siguiente:

- Borrar el buffer (void deleteBuffer())
- Leer fichero (void readfile(char *file))
- Dibujar el menú + update (drawMenú() + SDL_Flip())

Versión actual:

Para poder trabajar con la función "TTF_Render" de SDL es necesario crear un vector de caracteres. En C++ hay dos maneras de crear un vector de caracteres. La primera versión viene de C, la segunda de C++.

La primera variante es más fácil de entender y aprender. En este caso se muestra la función sprintf, el cual se ha ultilisado para realizar el sistem de Intelisense. Es necesario tener un array de tamaño fijo y un vector de char.

```
sprintf(intelisense,"%s%c", intelisense, jkl[IndexCharacterIntelisense]);
```

La segunda variante consiste en varios pasos, primero crear las variables:

```
#include <string>
#include <string.h>
using namespace std;

  string HelpMainMenuItemLabel1; char *HelpMainMenuItemLabelpointer1;
  string HelpMainMenuItemLabel2; char *HelpMainMenuItemLabelpointer2;
  string HelpMainMenuItemLabel3; char *HelpMainMenuItemLabelpointer3;

  string HelpMainMenuHelpLabel1; char *HelpMainMenuHelpLabelpointer1;
  string HelpMainMenuHelpLabel2; char *HelpMainMenuHelpLabelpointer2;
  string HelpMainMenuHelpLabel3; char *HelpMainMenuHelpLabelpointer3;
```

Luego asignar las variables y crear el vector de char. Después de crear el vector char hay que copiar la información del "string" al vector char. Este ejemplo el caso de ítems del menú "Help"

```
//Menuitemlabels
HelpMainMenuItemLabel1 = "Info";
HelpMainMenuItemLabel2 = "Control";
HelpMainMenuItemLabel3 = "About";

HelpMainMenuItemLabelpointer1 = new char[HelpMainMenuItemLabel1.size()+1];
HelpMainMenuItemLabelpointer2 = new char[HelpMainMenuItemLabel2.size()+1];
HelpMainMenuItemLabelpointer3 = new char[HelpMainMenuItemLabel3.size()+1];

strcpy(HelpMainMenuItemLabelpointer1, HelpMainMenuItemLabel1.c_str());
strcpy(HelpMainMenuItemLabelpointer2, HelpMainMenuItemLabel2.c_str());
strcpy(HelpMainMenuItemLabelpointer3, HelpMainMenuItemLabel3.c_str());

//Helptext
HelpMainMenuHelpLabel1 = "Profesor Tobí: Some hints about the application";
HelpMainMenuHelpLabel2 = "Profesor Tobí: About the control settings.";
HelpMainMenuHelpLabel3 = "Profesor Tobí: About us, me and my creator.";

HelpMainMenuHelpLabelpointer1 = new char[HelpMainMenuHelpLabel1.size()+1];
HelpMainMenuHelpLabelpointer2 = new char[HelpMainMenuHelpLabel2.size()+1];
HelpMainMenuHelpLabelpointer3 = new char[HelpMainMenuHelpLabel3.size()+1];

strcpy(HelpMainMenuHelpLabelpointer1, HelpMainMenuHelpLabel1.c_str());
strcpy(HelpMainMenuHelpLabelpointer2, HelpMainMenuHelpLabel2.c_str());
strcpy(HelpMainMenuHelpLabelpointer3, HelpMainMenuHelpLabel3.c_str());
```

Es muy importante borrar luego los vectores creados con "delete" para librerar espacio:

```
delete HelpMainMenuItemLabelpointer1; delete HelpMainMenuHelpLabelpointer1;
delete HelpMainMenuItemLabelpointer2; delete HelpMainMenuHelpLabelpointer2;
delete HelpMainMenuItemLabelpointer3; delete HelpMainMenuHelpLabelpointer3;
```

El sistema de intelisense fue programada de la siguiente manera: dos switch-cases, para cada fila y columna. Como se ha comentado, es necesario tener un vector de char y un array con el tamaño fijo. Para realizar el primer caso (abc) en el teclado "inteligente" el código es lo siguiente:

Antes se ha escrito todos arrays necesarios como variables globales:

```
char abc[] = {'a','b','c','A','B','C','á','à','â','ä','æ','Á','À','Â','Ä','Æ','ç','Ç'};
char def[] = {'d','e','f','D','E','F','é','è','ê','ë','É','È','Ê','Ë'};
char ghi[] = {'g','h','i','G','H','I','í','ì','î','ï','Í','Ì','Î','Ï'};
char jkl[] = {'j','k','l','J','K','L'};
char mno[] = {'m','n','o','M','N','O','ñ','Ñ','ó','ò','ô','ö','Ó','Ò','Ô','Ö'};
char pqrs[] = {'p','q','r','s','P','Q','R','S','ß'};
char tuv[] = {'t','u','v','T','U','V','ú','ù','û','ü','Ú','Ù','Û','Ü'};
char wxyz[] = {'w','x','y','z','W','X','Y','Z'};

char characters[] = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',
                     'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'};

char pmark[] = {'¿','?','¡','!',',','.',':',';', '»','«', ' '};
```

Estructura general del sistema de intelisense:

```
switch(UpDownSetting)
{
  case 1:
  {
  switch(LeftRightSetting)
          {
          case 1: //abc
                  {
                  if(IndexCharacterIntelisense < 17)
                          {
                          IndexCharacterIntelisense++;
                          sprintf(intelisense,"%s%c", intelisense,
abc[IndexCharacterIntelisense]);
                          DrawKeyboardMobile();
                          }

                  else
                          {
                          DeleteIntelisenseArray();
                          DrawKeyboardMobile();
                          }
                  break;
                  }
           break;
          }
  }
```

Ahora es importante borrar el Array de intelisense si el usuario ha apretado más que elemtneos hay en el array. En el primero caso (abc) se contiene 17 elementos. Si se sale de estos, se debe borrar el array.

```
void DeleteIntelisenseArray()
{
  IndexCharacterIntelisense=-1;  //return to the first element

   for(int i =0;i<=strlen(intelisense);i++) intelisense[i] = NULL;
}
```

Ambos teclados poseen de borrar el último carácter escrito. De nuevo se trabaja con un array y se borrar de la siguiente manera:

```
void DeleteCurrentCharacter()
{
  int pos=strlen(writtenword);
  writtenword[pos-1]=NULL;
}
```

Para realizar la baja es importante mostrarlo también por pantalla y poner el valor inicila de unas variables, si no, la PSP empieza rallarse con las variables y salta de un menú al otro menú.

```
if(pad.Buttons & PSP_CTRL_SQUARE)
{
        if(OnKeyboardMobileMenu)
        {
                ShiftTime = false;
                IndexCharacter=0;
                DeleteCurrentCharacter();
                DrawKeyboardMobile();
        }

        else if(OnKeyboardQWERTYMenu)
        {
                ShiftTime = false;
                IndexCharacter=0;
                DeleteCurrentCharacter();
                DrawKeyboardQWERTY();
        }
}
```

Para cambiar de minyúscula a mayúscula o atraves, las dos siguientes fuciones fueron programado con la lógica de arrays el cual se ha mostrado antes. La primera de ellas es para el caso minyúscula a mayúscula, la otra para mayúscula a minyúscula. Esta lógica funciona solo con el array "characters".

```
void ShiftCharactersFromLowerToUpper(int i)
{
   IndexCharacter=i+26;
    int pos=strlen(writtenword);
    writtenword[pos-1]=characters[IndexCharacter];
}

void ShiftCharactersFromUpperToLower(int i)
{
   IndexCharacter=i-26;
    int pos=strlen(writtenword);
    writtenword[pos-1]=characters[IndexCharacter];
}
```

Para hacerlo de verdad, es importante comprobar si se ha cambiado o no. Para este caso se ha creado un bool "ShiftTime". Si es falso, quiere decir que no se ha cambiado, si es cierto, se ha cambiado.

Caso1: de minyúscula a mayúscula:

```
if(ShiftTime == false)
{
   ShiftTime=true;
   ShiftCharactersFromLowerToUpper(IndexCharacter);
   DrawKeyboardQWERTY();
}
```

Caso2: de mayúscula a minyúscula:

```
if(ShiftTime == true)
{
   ShiftTime=false;
   ShiftCharactersFromUpperToLower(IndexCharacter);
   DrawKeyboardQWERTY();
}
```

7. Herramienta de Autor

De momento es posible entrar en todos los ítems del menú. Pero todavía faltan unos detalles para complementar la funcionalidad del diccionario. En dichos casos, la PSP muestra una imagen de aviso "under construction"[30]. Ambos teclados funcionan, se puede escribir una palabra o un texto. Es posible cambiar entre ambos teclados en cualquier momento.

Futuro

En el capítulo "Diseño" ya se ha hablado un poco sobre el tema futuro. En esta parte se ilustra de nuevo unas implementaciones para la aplicación en el futuro. Una de estas consiste en la mejor interacción entre mascota y usuario. La mascota debería saludar al usuario, avisarle sobre los últimos resultados del examen y dar consejos de cómo mejorar los exámenes. También debería hablar, quiere decir reproducir una pista de audio para enseñar cómo se pronuncia una palabra. Deberían existir mini juegos de tipo test para dar la oportunidad de aprendizaje rápido de vocabularios. Aparte de los exámenes, la aplicación debería ofrecer un resumen o mejor dicho una estadística del estudio como todas las AA de la NDS. Es obstante realizar más análisis para saber si es necesario crear varios perfiles o no. Asimismo, se debería pensar sobre la opción de combate entre los jugadores en uno de los mini juegos a través de la conexión WiFi. Para realizar los mini juegos es necesario trabajar con un sistema de información, en este caso XML. El diccionario debería buscar la palabra y/o buscar aleatoriamente una palabra en el fichero XML. Como se ha planificado, debe crear una lista de historia de búsqueda de las 5 últimas búsquedas realizadas. En las búsquedas la aplicación debería acordarse en qué idioma la búsqueda fue realizada. Quiere decir, si el usuario ha buscado la palabra alemana "Wörterbuch" en inglés, en español u otro idioma. Las implementaciones requieren una inteligencia del software para evitar por ejemplo que en los minis juegos ocurran siempre las mismas preguntas, para realizar las sugerencias. Con ayuda de un fichero de XML, se podría realizar una GUI "inteligente". Dar la posibilidad de cambiar el idioma de la interfaz. La aplicación de momento está en inglés, pero la aplicación debería ser para niños de todo el mundo. Entonces, sería mejor ofrecer la opción de cambiar el idioma de la interfaz. Se podría hacerlo antes de arrancar la aplicación o en la opción "Extra" como un sub-elemento "Options". Para los idiomas asiáticos es obstante analizar las teclas del teclado para poder escribir también en chino, japonés u otro idioma asiático. En este caso, también se debe cambiar el formato del fichero XML para el diccionario.

[30] bajo construcción

El tema de conexión WiFi forma una parte importante en la calificación de una aplicación o videojuego. En este caso, el diccionario debería ofrecer la opción de descarga a través de un servidor por ejemplo PSN. El diccionario debe soportar un cierto tipo de XML. Esta opción mejora el uso del diccionario. El usuario no tiene que descargarse una aplicación para traducir del idioma x al idioma y, si no solo necesita el fichero XML. Para mejorar el uso de WiFi, el diccionario debería dar la posibilidad de conectarse al traductor de Google u otros diccionarios online (wordreference, leo, etc.). La opción de WiFi es importante para la PSP Go y las otras generaciones de PSP (1000, 2000, 3000) excepto la nueva PSP (E-1000), el cual no tiene ninguna conexión de WiFi para reducir el coste de producción.

La PSP ofrece una cámara para hacer fotos o para jugar los nuevos juegos del estilo virtualidad aumentada: "EyePet"[31] y "Invizimals PSP"[32]. Se debería usar la técnica de la virtualidad aumentada para traducir una palabra grabada con la cámara directamente en la PSP. Así el usuario no tiene que escribir la palabra con el teclado si no solo grabarla con la cámara. Dicha tecnología se debería usar también para los mini juegos para el diccionario. Por ejemplo - "¿qué significa la palabra inglesa "dance"?". Entonces el usuario debería buscar con la cámara en la habitación la palabra. Durante la búsqueda aparecen en la PSP varias otras palabras. Si el usuario encuentra la palabra correcta, debería apretar ⊗.

Por las nuevas PSP (E-1000, Vita) es necesario de realizar versiones extras. La PSP E-1000 no soporta la conexión WiFi, entonces se debe dar una nueva opción para substituir "Dictionary Online". Debería ser un nuevo mini juego o cualquier otra opción sin usar WiFi. Para la sucesora de PSP, la PSP Vita es importante acceder a los menús a través de la pantalla táctil con un "Stylus" como en la NDS y N3DS. También se podría hacer funciones extra como hacer cosquillas a la mascota, acariciar a la mascota, etc.

Todavía no se sabe nada en el tema de programar en la PSP Vita. La programación en la PSP E-1000 es igual como en la PSP-2000 o PSP-3000.

[31] Video tráiler del juego "EyePet PSP": http://www.gamespot.com/psp/puzzle/eyepet/video/6273719/eyepet-psp-gamescom-2010-trailer

[32] Video tráiler del juego "Invizimals PSP": http://www.youtube.com/watch?v=6LUeLJR0uk8

8. Críticas constructivas

Todavía faltan unas cosas de programar para acabar el diccionario, aunque el diccionario fue programado en la metodología de XP. Por la información mala y la arquitectura mala, el proyecto se ha hecho dos veces de nuevo. Primero fue la versión alfa, luego la versión 2 y por último la versión actual. El resultado de la versión actual está bastante bien hecha. Funciona rápida en el tema de dibujar los menús y/o los teclados como los caracteres tecleados. Ofrece unas opciones nuevas como por ejemplo cambiar en cualquier momento entre ambos teclados o volver en cada momento al menú principal. Especialmente con el cambio de la arquitectura del programa estoy muy orgullosa, porque fue obstante sacar todos los detalles de SDL en particular de cómo imprimir un texto por pantalla. Suena muy fácil, pero en el caso de SDL puede costar muchos esfuerzos, porque la función "TTF_RenderText" requiere primero un vector de caracteres. Luego, dicha función no dispone de saltos de líneas. Por estos motivos, no es recomendable programar una aplicación de muchos elementos de texto como un diccionario o un teclado. Para reproducir audio con SDL existe un problema de "ruido" en el audio. Quiere decir, si la PSP está reproduciendo una pista de audio un rato, el audio empieza a tener ruidos raros de medio segundo o un segundo. Asimismo la PSP siempre está cargando el audio, lo que puede causar el ruido en el audio. En el caso de que intervienen varios elementos en el mismo tiempo, el audio empieza a tener más ruidos. Por ejemplo en la versión actual se ha implementado tener audio. Pero por el "ruido", la reproducción de audio fue eliminada. Además en la comunidad de programación de PSP la mayoría de usuarios no usan SDL, porque programan con OpenGl o con la librería gráfica de PSP. Aparte de SDL, el análisis sobre los diccionarios tomaba mucho tiempo. Comprobar las diferencias de cada uno, pensar en la funcionalidad de diccionario y pensar en el usuario. Cómo mejorar el uso de la PSP. Un ordenador dispone de un teclado y ratón, la NDS dispone del "Stylus" y una pantalla táctil como los botones. En el caso de la PSP solo existen los botones y el OSK original. No oficial existe un teclado para la PSP, pero para poder usarlo es necesario tener una PSP con CF y el teclado solo funciona con el "Homebre" – "TyDoPad" [5]. El OSK original no está mal. Pero para un jugador puede costar mucho escribir con él, porque las teclas más usadas en la mayoría de videojuegos son ⊗, ○, △, □. Para escribir un carácter en el teclado OSK teléfono, es necesario seleccionar el carácter con ⊗ y luego escribirlo con R1. Especialmente para niños es difícil usar muchas veces el botón R1 en vez de los botones más usados. Debido a estos detalles, el propio OSK en la versión actual facilita escribir, porque usa los botones más usados y ofrece la posibilidad en el teclado "QWERTY" cambiar una letra de minúscula a mayúscula (R1) o a través (L1). El OSK original se usa en la PSP solo para ver páginas web en la PSP. Yo personalmente no uso casi nunca el WiFi en la PSP, porque la mayoría de páginas webs no soporta la PSP. Quiere

decir, la página web es demasiada grande para la pantalla de la PSP. En muchas ocasiones hay que realizar el "scroll down" para ver los contenidos interesantes.

En la **Figura 57** se muestra el OSK original de la PSP.

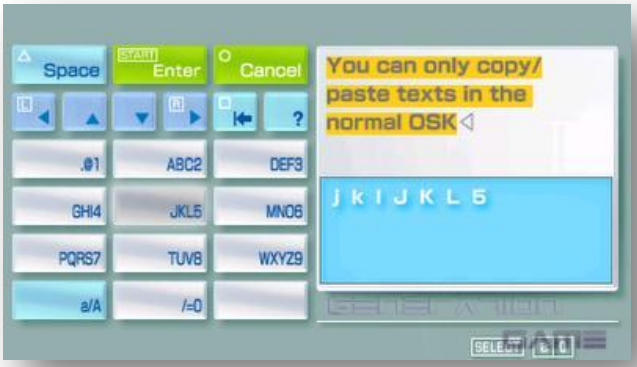


Figura 57: Original OSK de la PSP.

El desarrollo de una aplicación "Minis" se debería hacer en un equipo pequeño, porque son demasiadas tareas para "un equipo" de únicamente una persona. Lo ideal sería un equipo de 3 personas, el diseñador (y también animador) y dos programadores. Uno de los dos programadores se encarga de la arquitectura y la inteligencia artificial del programa y el otro de la implementación de todos los elementos de la GUI. Como comentado, el proyecto fue realizado de la metodología de XP. Cuando se ha tenido los primeros resultados del análisis y los diseños, la programación comenzó directamente. Después de la programación siempre se realizaba unas pruebas para saber si todo funciona bien o no. En particular la arquitectura del programa fue comprobado (menús, teclados) para ver la diferencia entre el resultado de tiempo de ejecución anterior y el nuevo resultado. En este caso, el resultado consiste en la mejor representación de menús y de los teclados. Quiere decir, la PSP dibuja los elementos más rápido que antes. También por todos los problemas fue necesario realizar todo lo más rápido posible. Por ejemplo, los teclados fueron programado en menos de 2 días – diseñar, implementar el dibujo y los botones, dar la posibilidad de cambiar entre ambos teclados y ofrecer opciones extras (teclado "QWERTY" – escribir en mayúscula o minúscula). El problema de CB causaba un retraso de varios meses, aunque las primeras pruebas fueron programadas sin IDE para que el proyecto no se quedase en un bucle infinito del desarrollo. Por las respuestas rápidas[33] en el foro de programación de la PSP [3] fue posible resolver muchos problemas y avanzar mucho en el desarrollo en un cierto tiempo.

[33] Después de escribir en el foro, después de unas horas o el día siguiente ya había una respuesta para resolver el problema.

El resultado final está bastante bien, aunque faltan ciertas funciones como buscar en un fichero XML, conectar al traductor de Google o disponer de mini juegos. Es menester decir el cambio de la arquitectura y la mejor de la GUI. En la versión actual la arquitectura es bastante estable. El número total de líneas de código se ha reducido aproximadamente a un $\frac{1}{3}$ de la versión alfa y el tiempo de dibujar todos los elementos cuesta menos aproximadamente 50% menos de tiempo. En la primera iteración la PSP es un poco lenta como en las iteraciones siguientes, porque tiene que cargar todos los elementos por la primera vez en su memoria. La aplicación hace luego referencia a la memoria, por esto la PSP dibuja los elementos más rápidos en las iteraciones restantes. Quiere decir, si el usuario usa por primera vez ambos teclados, es la primera iteración de dibujar los elementos. Luego, si el usuario vuelve al teclado, la PSP realiza la segunda iteración.

Con este proyecto se da la posibilidad de aprender sobre el tema de videojuegos, porque el diccionario es indirectamente un videojuego para la PSP. Los análisis necesitan mucho tiempo por la grande selección en cada campo (juegos, diccionarios, aplicaciones de aprendizaje, teclados), también por los requisitos de una aplicación hoy en día. No se puede comprobar un juego antiguo con los videojuegos actuales por el tema de programación y del diseño. Es menester que existan diferencias entre los juegos de una videoconsola y de una videoconsola portátil. Quiere decir, que la aplicación debería ser "inteligente". Debido a la falta de tiempo y a ningún conocimiento de inteligencia artificial, no fue posible acabar el diccionario en tiempo. La inteligencia es importante para que la aplicación no haga siempre lo mismo, como por ejemplo preguntar siempre las mismas preguntas en el examen. Esto lo indica directamente la grande importancia de la programación.

Aparte de estos detalles, es obstante aprender a dirigir a un equipo. En este proyecto se ha enviado la información necesaria para colorear los bocetos del búho y realizar la animación a Sabrina Takeuchi. Colorear una imagen cuesta mucho tiempo aunque el dibujo no es muy complicado. Especialmente crear una animación requiere mucho tiempo y conocimiento en dibujar. Como se ha comentado antes, fue importante contactar con una persona más en el desarrollo del proyecto, en concreto a Sabrina Takeuchi como animadora del búho.

Con este proyecto he aprendido mucho en el desarrollo de videojuegos. Con este conocimiento quiero seguir con el proyecto en un equipo de dos personas. Yo como programadora y Sabrina Takeuchi como animadora y diseñadora.

9. Apéndice

Análisis e información adicional: Contiene todos los resultados de los análisis sobre el diseño. Asimismo el código cómo sacar el OSK original en la PSP y la información adicional para aprender cómo poner sonido en su "Homebrew" en el XMB.

Bibliografía

[1] Información sobre la Sony PlayStation Portable "Minis". http://www.zath.co.uk/sony-psp-mini-games-announced/ (página web está en inglés), llamada 2010-09-19

[2] Anush Euredjian Chiappe y Juan Cortés Maiques; Guía de desarrollo de aplicaciones para PSP - Especificaciones e Instalación, UPV 2010
Anush Euredjian Chiappe y Juan Cortés Maiques; Guía de desarrollo de aplicaciones para PSP – El lenguaje C, UPV 2010

[3] PSP-Programming Tutorials. http://www.psp-programming.com/forums/index.php?action=globalAnnouncements;id=1, llamada 2010-12-22
PSP-Programming Foro. http://www.psp-programming.com/forums/index.php/board,2.0.html, llamada 2011-09-19

[4] Preguntas frecuentes sobre la diferencia entre la PSP E-100 y la PSP 3000. http://de.playstation.com/psp/support/general/detail/linked398344/item398322/PSP-E1000-Antworten-auf-deine-Fragen/ (página web está en alemán), llamada 2011-09-19

[5] Teclado para las PSPs con CF. http://www.qj.net/psp/homebrew-applications/the-psp-keyboard-has-arrived-tydopad-v001-beta.html, llamada 2011-09-19

[6] El lexicón en alemán con los términos usados en la comunidad de programing PlayStation (PSP, PS2, PS3)
http://www.pspking.de/forum/misc.php?action=lexikon, llamada 2011-09-23

[7] PSP-PRX-Ejemplo: http://forum.pspfreak.de/psp-programmierung/85047-c-prx-datei-starten.html (página web está en alemán), llamada 2011-09-23

[8] Tutoriales de programación PRX - http://forums.dashhacks.com/f141/prx-tutorial-and-template-t180216/, http://forums.ps2dev.org/viewtopic.php?t=4269, llamada 2011-09-23

[9] Enlace de amazon sobre el precio de la PSP 3000. http://www.amazon.es/PSP-SLIM-BLACK-SERIE-3000/dp/B001OI2LRC/ref=sr_1_1?ie=UTF8&qid=1316961809&sr=8-1, llamda 2011-09-23

Declaración jurada

Yo afirmo que este estudio mío lo realicé totalmente por mi cuenta sin apoyo ilegal. Marqué todos los enfoques usados que no son míos con citas directas o indirectas. Estoy segura en que cada aseguración mala tiene consecuencias jurídicas.

Todas las imágenes utilizadas en este estudio, que no fueron creadas por mí son con derechos de autor. Yo solo he realizado las capturas de pantalla y ninguna modificación.

Valencia, 30 de Septiembre de 2011 Katharina Maria Kaczmarczyk