Porfolio

Víctor Grau Moreso Mark Holland Lluís Ulzurrun de Asanza Sàez

Índice

1 Práctica 1 - Instalación del entorno

1.1 Preparación del entorno

Hemos preparado nuestro entorno de trabajo sobre Debian 7.8, versión de 32 bits[1].

1.1.1 Instalación de dependencias

Antes de poder instalar el kit de herramientas para desarrollar para la Nintendo DS es necesario preparar el sistema para poder utilizar estas herramientas. Concretamente las dependencias del toolkit son `make`, `gcc` y `tar`. El paquete `desmume` es necesario si se quieren probar los proyectos usando el emulador DeSmuME mientras que los paquetes `wine` y `unzip` son necesarios para poder emular el hardware objetivo mediante No$GBA.

Esta última posibilidad es muy interesante ya que No$GBA dispone de una versión especial orientada a facilitar en gran medida el desarrollo de software para la Nintendo DS e incluye, entre otras herramientas, un navegador de código ensamblador para poder saber qué instrucción está ejecutando la consola en cada momento y una consola de debug lo que permite imprimir mensajes desde la aplicación sin necesidad de interactuar con los controladores de las pantallas.

A pesar de no ser estrictamente necesario en nuestro caso hemos instalado algunas dependencias adicionales, entre ellas utilidades como `wget` o `g++`.

```
sudo apt-get update -qy
sudo apt-get upgrade -qy
sudo apt-get install wget tar make gcc g++ desmume wine unzip \
    libc++6 libc6 -qy
```

Figura 1: Los comandos ejecutados para instalar las dependencias en nuestro entorno de trabajo

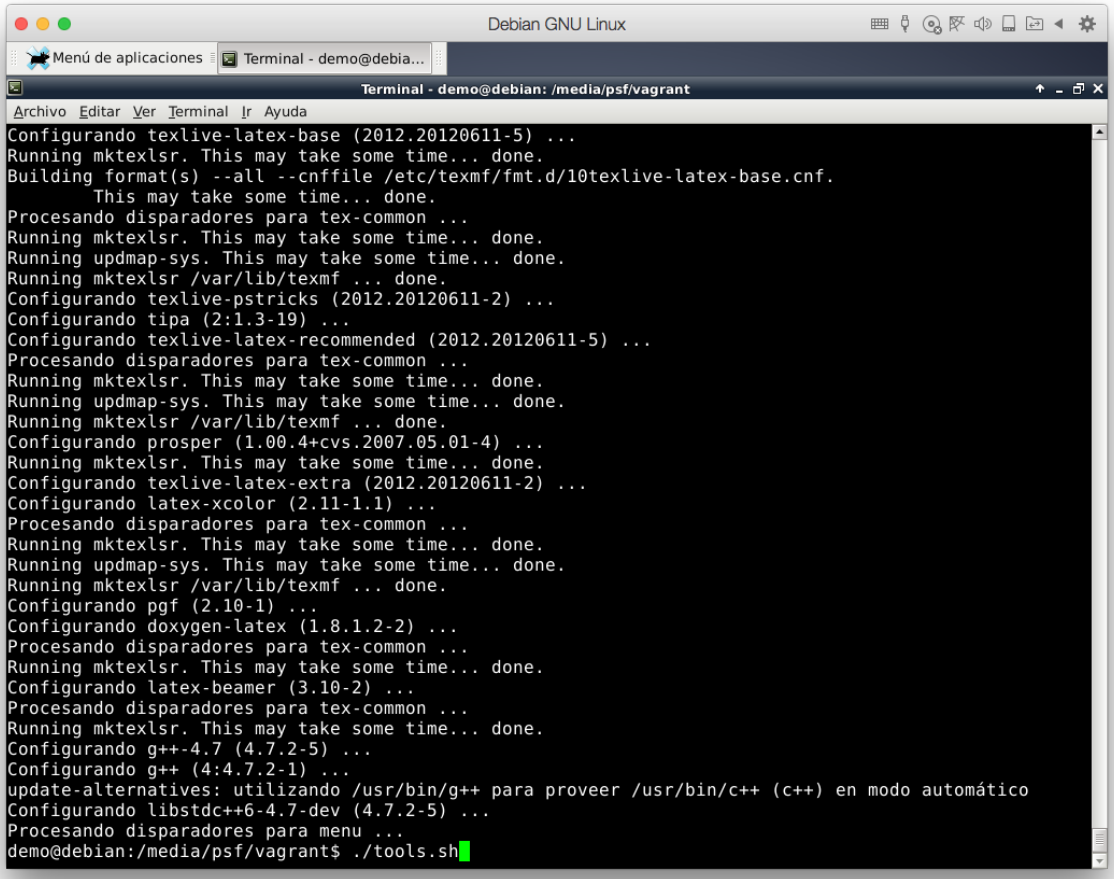
1.1.2 Instalación de las herramientas de desarrollo

Los pasos para la instalación de las herramientas de la guía de la práctica eran muy claros pero para facilitarnos la instalación de las herramientas en múltiples entornos de trabajo hemos agrupado todas las instrucciones en un único fichero para que instalar el kit de desarrollo fuese más sencillo.

Este script (**Figura 3**) descarga `devkitARM`, lo instala, prepara los archivos de configuración necesarios (realmente es tan sencillo como añadir un par de variables de entorno), descarga los ejemplos y los compila.

[1]Inicialmente nuestra intención era trabajar sobre Debian 7.8 x64 pero tras problemas con Wine y No$GBA decidimos usar la versión de 32 bits.

Figura 2: Resultados de la instalación de dependencias.



Nótese que este script no siempre funcionará de forma automática: algunos los enlaces usados son los enlaces de descarga directa temporales de `SourceForge.net`, que expiran tras cierto tiempo. Cuando expiran es necesario esperar unos segundos para descargar el fichero y es posible que el script lo que descargue sean las páginas de espera en lugar de los ficheros deseados. La solución es simple: basta con acceder al link de descarga caducado mediante un navegador web y extraer el nuevo link de descarga inmediata (aparece bajo el título “direct link”).

```
cd ~/
wget -O ~/devkitARMupdate.pl http://goo.gl/rjbq5P
chmod a+x ~/devkitARMupdate.pl
~/devkitARMupdate.pl
export DEVKITPRO=~/devkitPro
export DEVKITARM=${DEVKITPRO}/devkitARM
echo "export DEVKITPRO=~/devkitPro" >>  ~/.bashrc
echo "export DEVKITARM=${DEVKITPRO}/devkitARM" >>  ~/.bashrc
echo "export DEVKITPRO=~/devkitPro" >>  ~/.zshrc
echo "export DEVKITARM=${DEVKITPRO}/devkitARM" >>  ~/.zshrc
wget -O /tmp/examples.tar.bz2 http://goo.gl/l5I1x1
tar xjvf /tmp/examples.tar.bz2
cd ~/devkitPro/examples/nds
make
cd ~/
wget -O gba.zip http://emuparadise.me/emulators/files/user/n-west-w-1776.zip
unzip gba.zip
```

Figura 3: Los comandos ejecutados para instalar las herramientas de desarrollo y compilar los ejemplos.

1.2 Ejecución

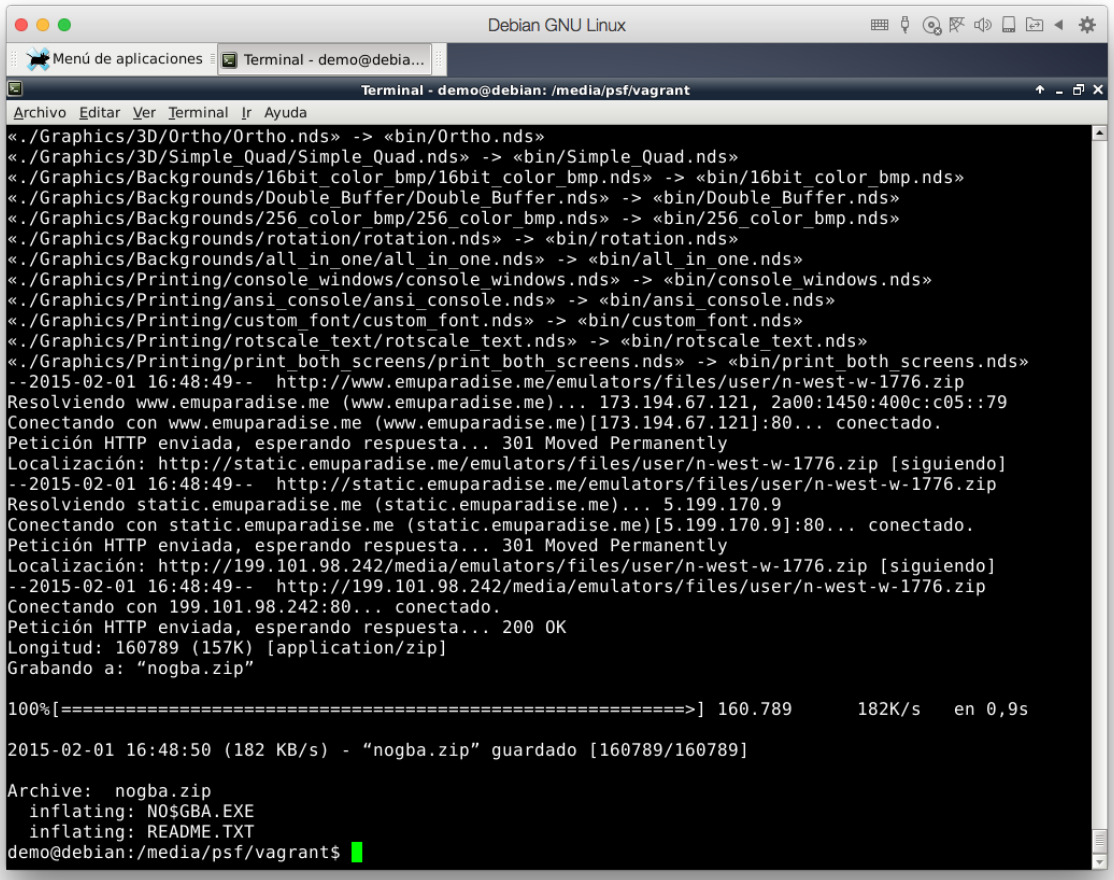
1.2.1 Compilación de los proyectos de prueba

Los proyectos de prueba pueden compilarse tanto mediante el comando `make` ejecutado en la raíz de la carpeta que contiene los programas de ejemplo como mediante el mismo comando `make` ejecutado en la carpeta de cada proyecto. La diferencia radica en que mediante el primer método se compilan todas las aplicaciones de ejemplo mientras que la segunda opción únicamente compila un proyecto concreto, dependiendo de las necesidades de cada momento convendrá usar una aproximación u otra.

1.2.2 Emulación de los proyectos de prueba

El emulador DeSmuME puede ser invocado desde la terminal mediante el comando `desmume <ruta a fichero .nds>`, sin embargo No$GBA no acepta ningún argumento, además requiere ser invocado mediante `wine` por lo que en nuestro entorno hemos dispuesto de los siguientes alias:

Figura 4: Resultados de la instalación del toolkit.



1.2.3 Ejecución de los proyectos de prueba sobre hardware real

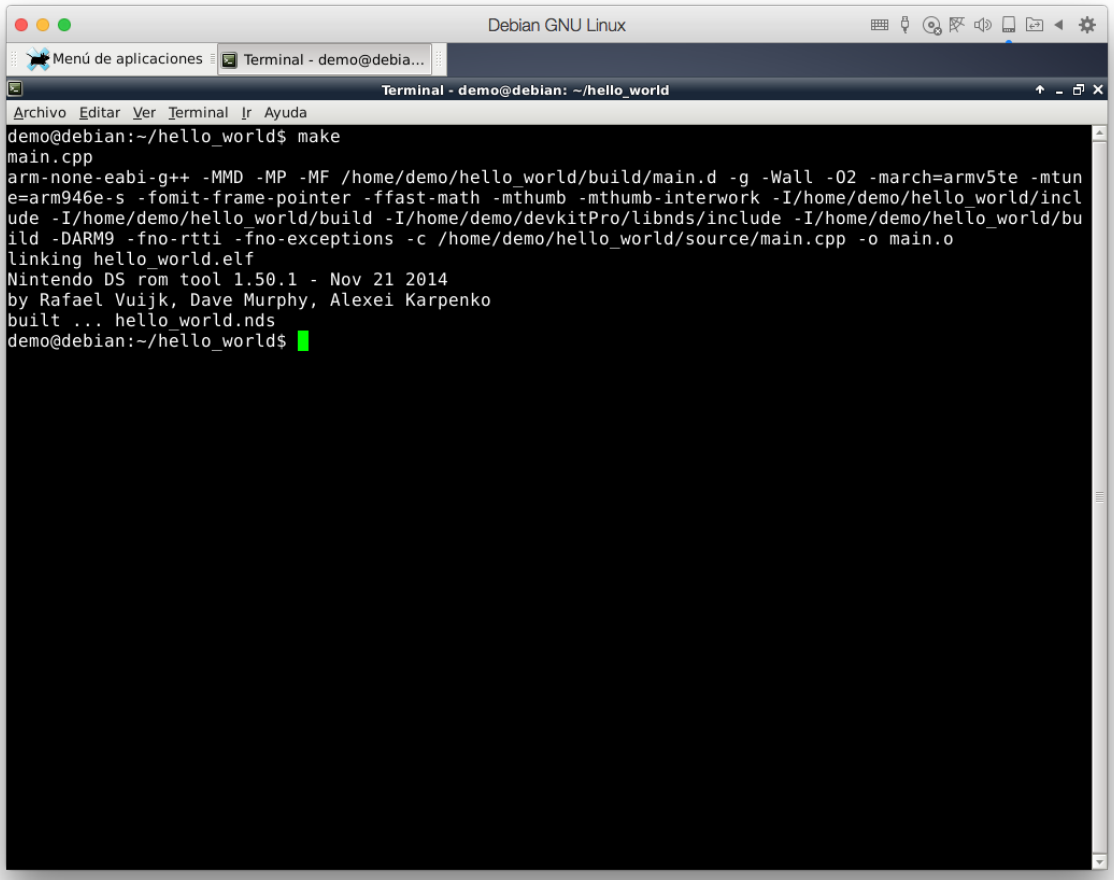
Los programas desarrollados también se pueden ejecutar sobre hardware real mediante el uso de Flashcards para evitar el firmado de código. Las aplicaciones funcionan tanto en Nintendo DS como en Nintendo 2DS o 3DS gracias a la capa de compatibilidad que incluyen las consolas modernas de Nintendo.

1.3 Automatización de la instalación del entorno de desarrollo

Nuestro grupo trabaja sobre diversos sistemas operativos, como Mac OS X o Windows. Esta diversidad de software hace que sea realmente difícil aislar ciertos problemas dependientes del entorno de modo que resulta conveniente utilizar una máquina virtual a fin de evitar estos problemas de compatibilidad.

Recurrimos a virtualizadores como VirtualBox o Parallels Desktop manejados mediante Vagrant para agilizar la puesta a punto del entorno, sin embargo en esta ocasión apostar por Vagrant

Figura 5: Resultados de la compilación del proyecto `Hello World`.



no fue una buena decisión ya que nos encontramos con problemas a la hora de conseguir habilitar el entorno de escritorio por lo que finalmente instalamos Debian en la máquina virtual manualmente y aprovechamos el script que teníamos preparado para *provisionar* la máquina para instalar todo el software necesario.

```
alias nocashgba="wine ~/NOcashGBA.exe"
alias nocashgbadebug="wine ~/NOcashGBAdebug.exe"
```

Figura 6: Alias para ejecutar No$GBA más cómodamente.

Figura 7: `Hello World` ejecutado en `DeSmuME`.

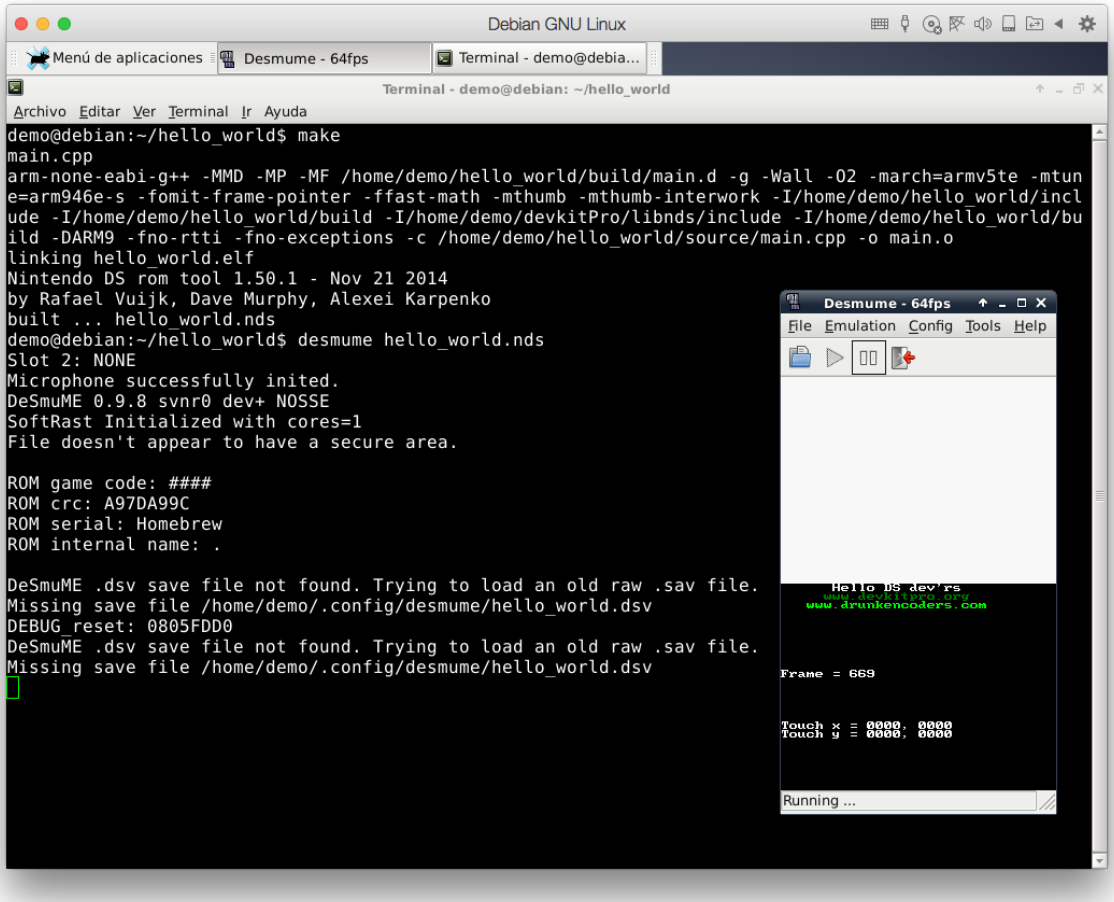


Figura 8: `Hello World` ejecutado en `No$GBA`.

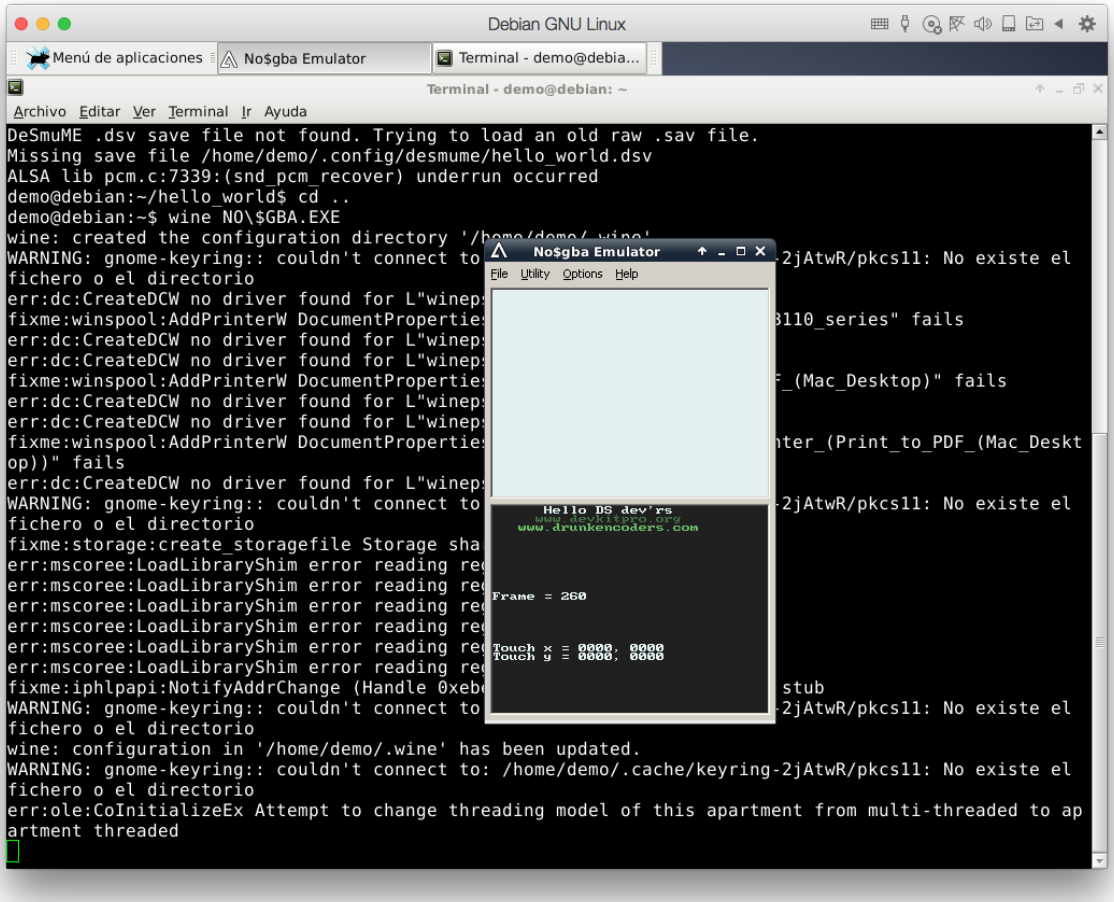


Figura 9: `Hello World` ejecutado en `No$GBA` debugger.

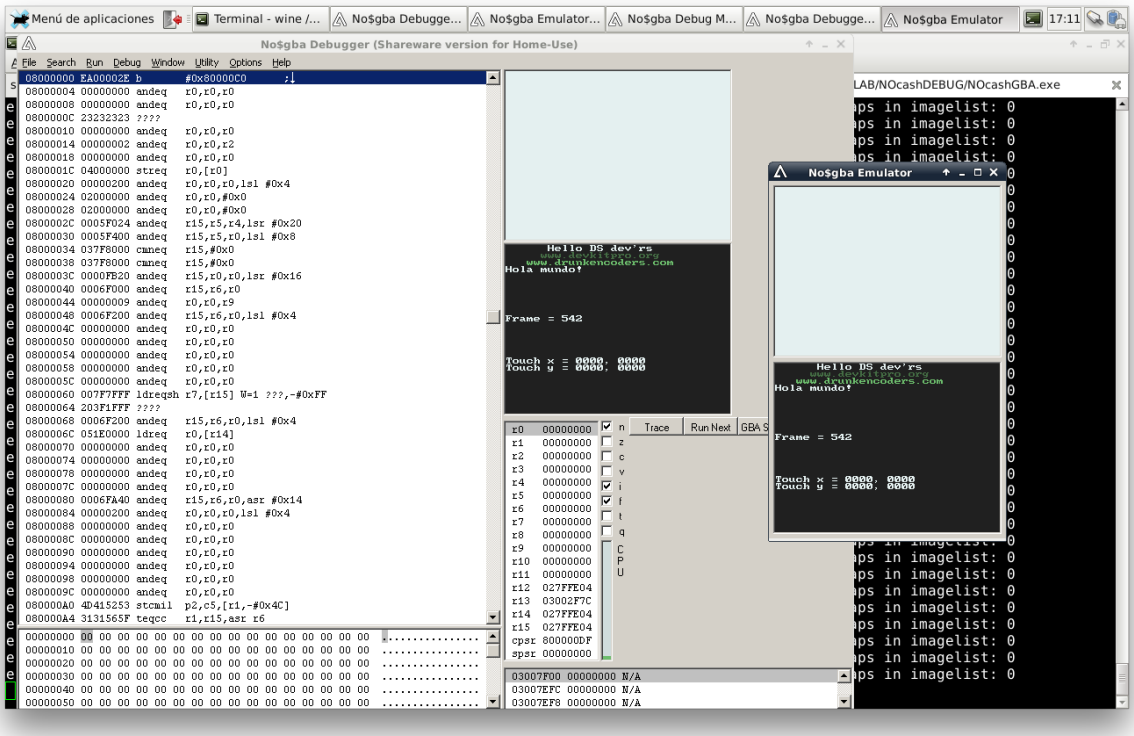
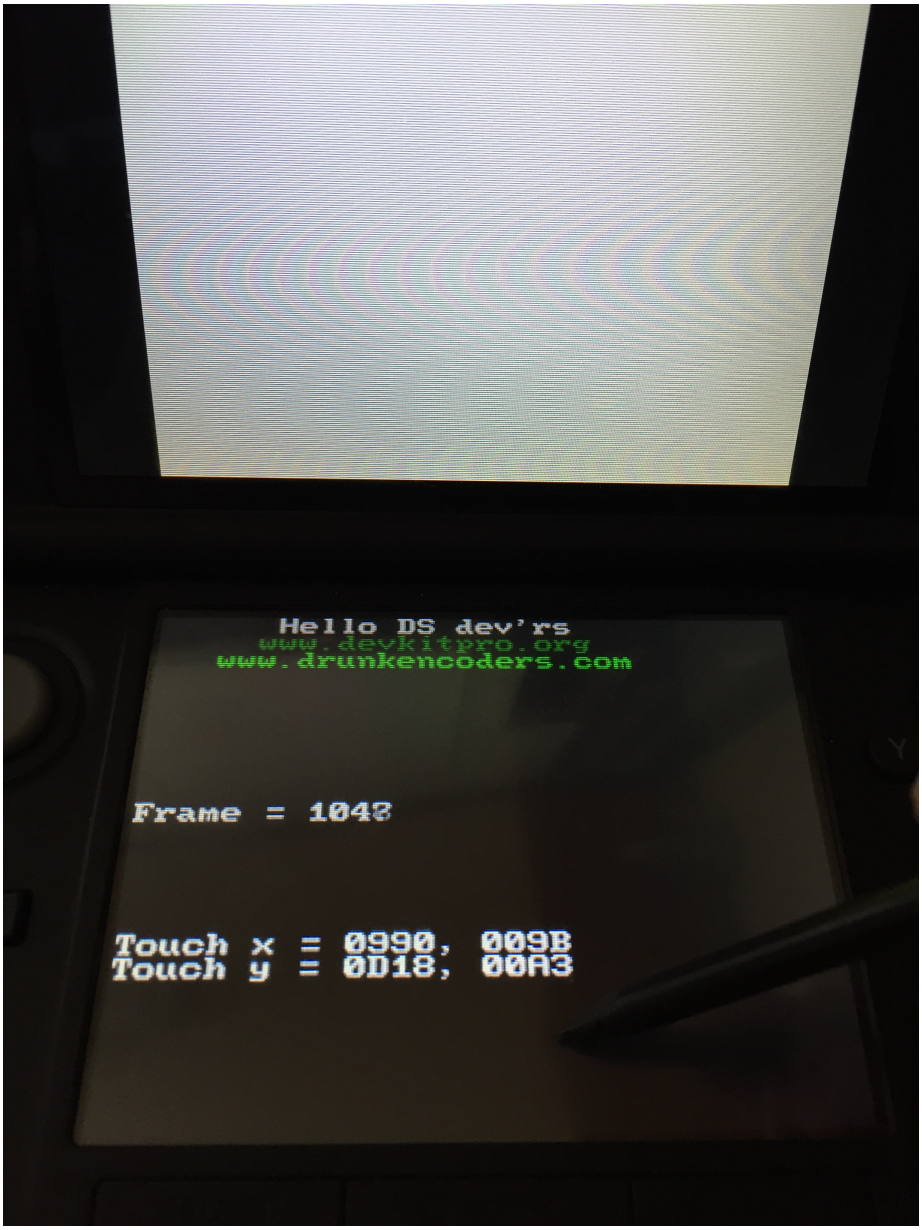


Figura 10: `Hello World` ejecutado en una Nintendo 3DS.



```
# -*- mode: ruby -*-
# vi: set ft=ruby :
VAGRANTFILE_API_VERSION = "2"
Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.vm.box = "puphpet/debian75-x64"
  config.ssh.forward_x11 = true
  config.vm.synced_folder "../LAB", "/LAB"
  config.vm.provider "virtualbox" do |v|
    v.name = "AEV"
    v.memory = 3096
    v.cpus = 2
    v.gui = true
  end
  config.vm.provider "parallels" do |v|
    v.name = "AEV"
    v.update_guest_tools = true
    v.memory = 3096
    v.cpus = 2
    v.customize ["set", :id, "--on-window-close", "keep-running"]
  end
  config.vm.provision "shell", path: "dependencies.sh", privileged: false
  config.vm.provision "shell", path: "personal.sh", privileged: false
  config.vm.provision "shell", path: "tools.sh", privileged: false
end
```

Figura 11: Fichero `Vagrantfile` de definición de la máquina virtual. Los archivos referidos en este fichero se corresponden con los fragmentos detallados en los apartados previos. El fichero `personal.sh` únicamente instala `vim` y `zsh`.

2 Práctica 2 - Acceso al Hardware

2.1 PersonalData

Hemos imprimido todos los valores que nos ofrece el struct de `PersonalData`. Se puede observar en la **Figura 12** que hemos escogido el color verde para los nombres y el gris para los valores. Hemos observado que en el emulador No$GBA, no están definidos algunos de los valores como el *nombre* y el *mensaje personal* pero en el emulador DeSmuMe sí que existen valores para todos.

Figura 12: Captura del programa PersonalData.

2.2 Paso del tiempo

2.2.1 stopwatch

A continuación están las respuestas de los ejercicios planteados en el boletin.

1. **¿Cual es la definición (prototipo) de esta función?**

```
void timerStart( int timer, ClockDivider divider, u16 ticks, VoidFn callback )
```

2. **Busque también la definición de los argumentos segundo y tercero.**

   `ClockDivider_1024` equivale a dividir el reloj por 1024 ($\approx$ 32.7284 kHz) y el 0 como parámetro para el número de ticks antes de desbordar hace que cada tick de nuestro timer equivalga a un tick de reloj.

2.2.2 timercallback

1. **¿Qué significa la función que se le pasa como tercer parámetro?**

   Calcula el valor correcto para el desbordamiento de ticks para una frecuencia dada.

2. **¿Cual es la definición (prototpipo) de la función, el cuarto parámetro, que puede ser llamada por el timer?**

```
void timerCallBack()
```

3. **¿Cómo se le pasa/recoge información a/desde esa función?**

   Con la variable global `play`.

2.2.3 RealTimeClock

1. **¿Qué dos funciones se utilizan para averiguar la hora y la fecha?**

   Primero llama a `time` para que devuelve un struct con el Unix Time, después utiliza `gmtime` para tratar este `time_t` struct y devolver otro struct con el tiempo en un formato más manejable.

   Sus prototipos son los siguientes:

```
time_t time(time_t *t)
struct tm *gmtime(const time_t *timer)
```

2. **¿Cómo inicializa este ejemplo el uso de las dos pantallas de la consola?**

   Con las líneas #103 y #106:

```
        lcdMainOnTop(); // Del fichero system.h
videoSetMode(MODE_0_3D); // Del fichero video.h
```

2.3 Gestión de entrada

2.3.1 keyboard/keyboard_async

Proporciona un teclado con una consola donde aparece lo que se teclea.

Figura 13: keyboard_async

2.3.2 keyboard/keyboard_stdin

Lee del teclado como entrada estándar con un típico programa de hola $(inserta_nombre).

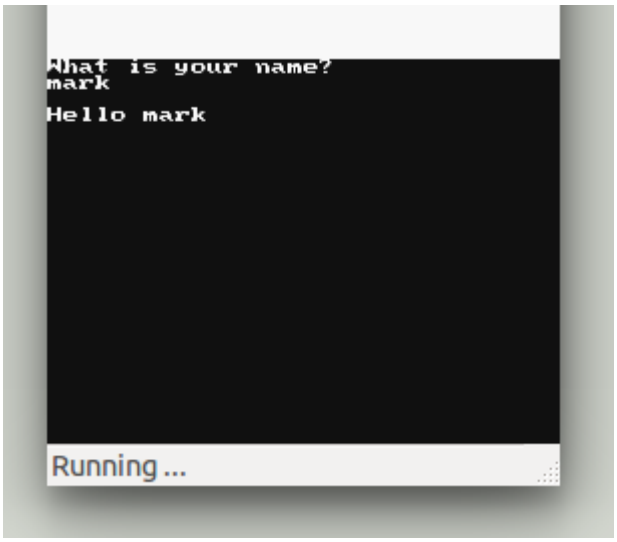


Figura 14: keyboard_stdin

2.3.3 Touch_Pad/touch_area/

Indica en pantalla qué punto de los ejes x/y de la pantalla está tocando el usuario y con qué cantidad de fuerza.

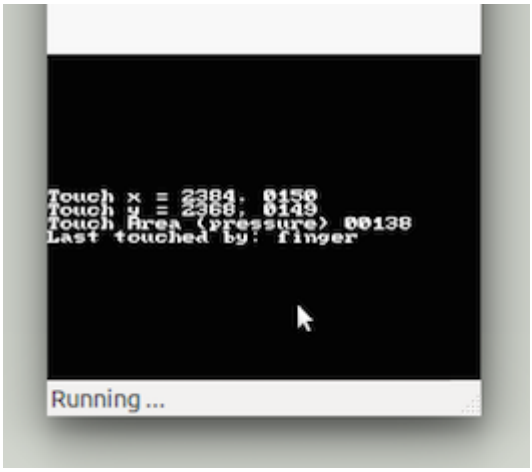


Figura 15: touch_area

2.3.4 Touch_Pad/touch_look/

Proporciona en la pantalla de arriba un mundo 3D donde podemos "mirar" utilizando la pantalla de abajo deslizando el dedo.

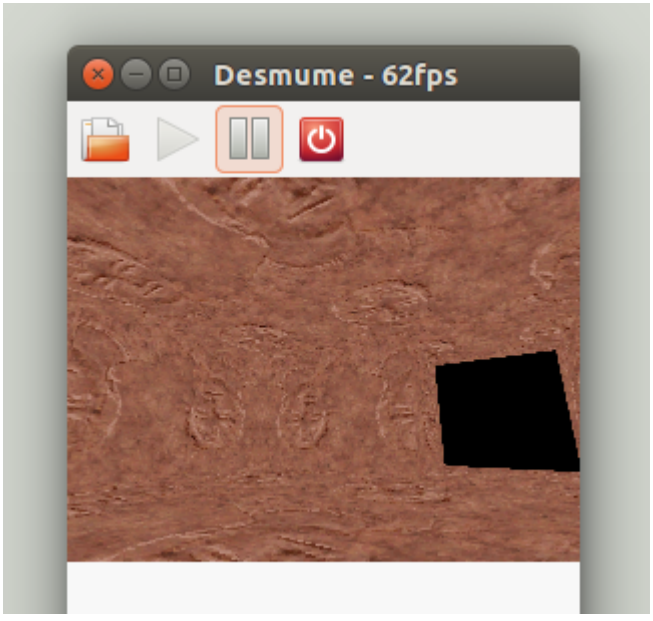


Figura 16: touch_look

2.3.5 Touch_Pad/touch_test/

Parecido al *test_area* pero proporcionando más información como que botones están siendo pulsados.

Figura 17: touch_test

2.4 Trabajo autónomo

Mediante los ejemplos indicados no hubo gran problema para realizar el ejercicio, lo único que ha requerido refrescar un poco ha sido para mostrar contenido en la pantalla de arriba manteniendo el menú en la de abajo.

A continuación dejamos capturas del programa mostrando el valor de `name` y lo mismo tras darle a la función de `lcdSwap`.

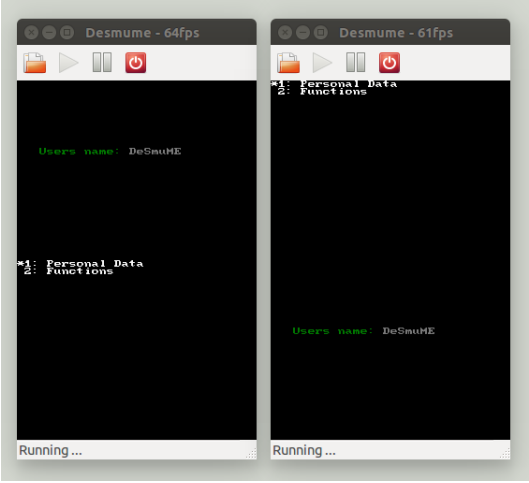


Figura 18: Programa con menú