UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

EMULADOR DE GAMEBOY PARA DISPOSITIVOS MÓVILES

Ángel Roldán Carrasco

Septiembre, 2007

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

EMULADOR DE GAMEBOY PARA DISPOSITIVOS MÓVILES

Autor: Ángel Roldán Carrasco

Director: Carlos Gonzalez Morcillo

Septiembre, 2007

© Ángel Roldán Carrasco. Se permite la copia, distribución y/o modicación de este documento bajo los términos de la licencia de documentación libre GNU, versión 1.1 o cualquier versión posterior publicada por la *Free Software Foundation*, sin secciones invariantes. Puede consultar esta licencia en http://www.gnu.org.

TRIBUNAL:

Presidente:
Vocal:
Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

RESUMEN

La emulación consiste en imitar el comportamiento de un sistema hardware o software sobre otro diferente al primero, de manera que para un observador externo el segundo sistema se comporte como el primero.

Es una técnica que está a la orden del día. Términos como máquina virtual, emulador o virtualización se oyen con frecuencia en el ámbito de la informática.

Por otro lado, el mercado de las PDA's y otros dispositivos móviles ha experimentado un importante crecimiento debido al abaratamiento de costes y el aumento de sus prestaciones, lo que ha posibilitado el desarrollo de un conjunto de frameworks alrededor de estos sistemas que permiten el desarrollo de aplicaciones de un modo rápido, portable y flexible.

Para la arquitectura PC, existen multitud de emuladores de videoconsola, siendo muchos de ellos software libre. Sin embargo, para dispositivos móviles como PDAs (Pocket PC o Palm) hay pocas alternativas y muy pocos de ellos son libres. En el proyecto GINAGE, se pretende desarrollar un emulador libre de Game Boy que sea multiplataforma para PC y varios dispositivos de tipo PDA.

A mis padres, mi familia y amigos, por apoyarme
en todo momento y tener confianza en mi.

AGRADECIMIENTOS

A Carlos por sus consejos, su tiempo y sus "empujones" periódicos en forma de e-mails.

A todos los que colaboraron en las pruebas de rendimiento: Ángel, Jose Antonio, Juan, Miguel Ángel, Carlos y Anastasio.

A Ángel y Jose Antonio por el ánimo que me han dado y por la ayuda prestada.

A todos los que hacen posible que el software libre exista y crezca día a día.

A todos vosotros, muchas gracias.

Índice de contenido

RESUMEN....5
Notación.... 13
Acrónimos.... 14
Capítulo 1. INTRODUCCIÓN.... 15
1.1 Antecedentes....15
1.2 Estructura de éste documento.... 16
Capítulo 2. OBJETIVOS DEL PROYECTO.... 18
Capítulo 3. ESTADO DEL ARTE.... 19
3.1 Conceptos previos sobre emulación.... 19
3.1.1 ¿Que es un emulador?....19
3.1.2 Tipos de emuladores.... 20
3.1.3 Los emuladores de videoconsola o arcade....22
3.1.4 Utilidades comunes de los emuladores....26
3.1.5 Aspectos legales....26
3.1.6 La obtención de ROMs.... 28
3.1.7 Emuladores conocidos de Game Boy para diferentes plataformas....32
3.2 Breve historia de las portátiles de 8 bits de Nintendo.... 47
3.3 El hardware de una videoconsola.... 49
3.3.1 Consolas de 8 y 16 bits.... 49
3.3.2 Consolas de 32 bits a las actuales.... 51
3.4 El hardware de la Game Boy.... 52
3.4.1 Introducción.... 52
3.4.2 Diferencias entre Game Boy y Game Boy Color.... 55
3.4.3 CPU....55
3.4.4 Memoria....60
3.4.5 Hardware de Video.... 66
3.4.6 Cartuchos.... 76
3.4.7 Otros periféricos (no emulados)....80
Capítulo 4. MÉTODO DE TRABAJO....83
4.1 Decisiones de diseño....83
4.1.1 Elección del sistema a emular....83
4.1.2 Elección del lenguaje y la plataforma.... 83
4.1.3 Elección del tipo de emulador....84
4.1.4 Elección del modelo de desarrollo....85
4.2 El ciclo de desarrollo en emulación....85
4.2.1 Metodología evolutiva orientada a prototipos.... 86

4.2.2 Las pruebas.. 87
4.2.3 Ventajas de la metodología evolutiva orientada a prototipos.......................... 88
4.3 Arquitectura del sistema... 88
4.3.1 Diagrama de estados genérico...92
4.3.2 La clase CPU...94
4.3.3 La clase Video...102
4.4 Fases en el desarrollo (Algunas iteraciones del modelo)......................................112
4.4.1 Lectura de metadatos de una ROM (cabecera)... 112
4.4.2 Generación del switch de procesamiento de instrucciones............................113
4.4.3 Comparativa de pintado de imágenes de Ewe y Superwaba..........................113
4.4.4 Implementación de readByte() writeByte() y reset()......................................113
4.4.5 Pruebas de rendimiento de implemetación de instrucciones y Flags.............113
4.4.6 Pruebas de accesos aleatorios a memoria..114
4.4.7 Emulación de la ROM homebrew Apocalipsis Now.......................................114
4.4.8 El soporte de Video y E/S...114
4.4.9 Soporte a más homebrew: Game Boy Demo, BigScroller y LandScape........ 114
4.4.10 La reescritura del pintado de la pantalla.. 115
4.4.11 La primera ROM comercial: “Castlevania 2 – Belmont's Revenge”...........115
4.4.12 Varias ROMs comerciales funcionando y soporte a GBC............................115
4.4.13 Mejoras en Ewe..115
4.4.14 La creación de la GUI... 116
4.4.15 Mejoras... 116
4.4.16 La versión final... 116
4.5 Consideraciones para mejorar la eficiencia... 116
4.6 Software utilizado durante el desarrollo.. 117
Capítulo 5. RESULTADOS... 118
5.1 ROMs comerciales soportadas.. 118
5.2 Comparativa de eficiencia en diferentes arquitecturas.. 121
5.2.1 Organización de las pruebas... 121
5.2.2 Diferencias entre arquitecturas...122
5.2.3 Diferencias entre máquinas virtuales... 124
5.2.4 Diferencias entre optimizaciones.. 126
5.3 Profiling.. 128
5.3.1 Consumo de memoria.. 129
5.3.2 Tiempo empleado por función... 130
5.3.3 Conclusiones.. 134
5.4 Análisis de una posible explotación comercial...135

5.4.1 Estado del mercado.......... 135
5.4.2 Costes de amortización.......... 135
Capítulo 6. CONCLUSIONES Y PROPUESTAS.......... 137
6.1 Objetivos alcanzados (consecución de los objetivos).......... 137
6.2 Propuestas de desarrollo futuras:.......... 137
Anexo A: Diagramas a color.......... 142
Anexo B: Manual de usuario.......... 144
Anexo C: Resumen de códigos de operación del LR35902.......... 148
Anexo D: Estructura del cd.......... 150

Índice de ilustraciones

Ilustración 3.1: Funcionamiento simplificado de emulador interpretador.......................... 21
Ilustración 3.2: Cálculo de flags para operación ADD de Z80 en C.................................. 24
Ilustración 3.3: Cálculo de flags para instrucción ADD de Z80 en ensamblador x86......... 25
Ilustración 3.4: Programadores de cartucho y cartuchos reescribibles................................ 31
Ilustración 3.5: Dos Game & Watch.. 47
Ilustración 3.6: Linea temporal de la Game Boy.. 49
Ilustración 3.7: Exterior de una Game Boy Color.. 53
Ilustración 3.8: Interior de una Game Boy Color.. 53
Ilustración 3.9: Detalle de la placa base de la Game Boy Color... 54
Ilustración 3.10: Formatos de instrucciones de Game Boy.. 58
Ilustración 3.11: Efecto wrap en capa Background.. 68
Ilustración 3.12: Capa Window sobre capa Background.. 69
Ilustración 3.13: Capa Sprite sobre Window y Background... 69
Ilustración 3.14: Combinación de las 3 capas para formar la imagen en pantalla................ 70
Ilustración 3.15: Organización en memoria de las paletas de Game Boy Color.................. 71
Ilustración 3.16: Organización de los dos juegos de datos de 256 tiles cada uno................ 73
Ilustración 3.17: Representación de tile en memoria... 74
Ilustración 3.18: Descripción de un mapa de tiles.. 75
Ilustración 3.19: Interior de un cartucho de Game Boy.. 76
Ilustración 3.20: Cartucho de Game Boy sin RAM.. 77
Ilustración 3.21: Apariencia externa de cada tipo de cartucho... 80
Ilustración 3.22: Accesorio Super Game Boy.. 81
Ilustración 3.23: Accesorios Game Boy Printer y Game Boy Camera................................. 81
Ilustración 3.24: Accesorio cable Gamelink.. 82
Ilustración 4.1: diagrama de etapas de metodologia evolutiva orientada a prototipos......... 87
Ilustración 4.2: Diagrama UML general de clases... 89
Ilustración 4.3: Diagrama de estados genérico... 93
Ilustración 4.4: Bucle principal del emulador.. 94
Ilustración 4.5: Diagrama de funcionamiento del método iteracion().................................. 95
Ilustración 4.6: Pseudocódigo de la función internalTiming.. 96
Ilustración 4.7: Parte del código de la función readByte().. 99
Ilustración 4.8: Pseudocódigo de la función writebyte()... 100
Ilustración 4.9: Funciones de la clase CPU... 101
Ilustración 4.10: Diagrama de funcionamiento del método pintarPantalla()....................... 102
Ilustración 4.11: Diagrama de funcionamiento del método calcularLineaComun()........... 103
Ilustración 4.12: Diagrama de funcionamiento del pintado de la capa Background........... 104

Ilustración 4.13: Diagrama de funcionamiento del pintado de la capa Window................106
Ilustración 4.14: Diagrama de funcionamiento del pintado de la capa Sprite....................108
Ilustración 4.15: Funciones de la clase Video..112
Ilustración 5.1: Comparativa de rendimiento de sistemas para SDK Java 5 de SUN........123
Ilustración 5.2: Comparativa de rendimiento del sistema 1 para diferentes SDKs............124
Ilustración 5.3: Comparativa de rendimiento del sistema 2 para diferentes SDKs............125
Ilustración 5.4: Comparativa de rendimiento del sistema 2 para diferentes HotSpot 126
Ilustración 5.5: Comparativa de rendimiento del sistema 1 para proguard........................127
Ilustración 5.6: Comparativa de tiempos del sistema 2 para proguard..............................127
Ilustración 5.7: Consumo total de memoria en GINAGE...129
Ilustración 5.8: Porcentaje de tiempo empleado por clases propias..................................130
Ilustración 5.9: Porcentajes de tiempo base para clases principales..................................130
Ilustración 5.10: Porcentajes de tiempo acumulativo para clases principales...................131
Ilustración 5.11: Porcentajes de tiempo base para funciones de la clase CPU..................131
Ilustración 5.12: Porcentajes de tiempo acumulativo para funciones de la clase CPU......132
Ilustración 5.13: Porcentajes de tiempo base para funciones de la clase Video................133
Ilustración A.1: Mapa de memoria en la Game Boy y Game Boy Color..........................142
Ilustración A.2: Efecto wrap en capa Background...142
Ilustración A.3: Representación de tile en memoria...143
Ilustración B.1: Pantalla principal de GINAGE...144
Ilustración B.2: El menú Archivo...145
Ilustración B.3: El cuadro de diálogo abrir...145
Ilustración B.4: El cuadro de dialogo para configurar teclas...146
Ilustración B.5: El menú Video..146
Ilustración B.6: El cuadro de dialogo para Editar paletas...147

Índice de tablas

Tabla 3.1: Resumen de características de emuladores de GB para PC................................ 36
Tabla 3.2: Resumen de características de emuladores de GB para algunas videoconsolas. 39
Tabla 3.3: Resumen de características de emuladores de GB para Palm.............................41
Tabla 3.4: Resumen de características de emuladores de GB para Pocket PC.................... 43
Tabla 3.5: Otros emuladores para Pocket PC...43
Tabla 3.6: Resumen de características de emuladores de GB para móvil Symbian.............45
Tabla 3.7: Resumen de características de emuladores de GB para móvil J2ME................. 46
Tabla 3.8: Resumen de Características del emulador iBoy para iPod..................................46
Tabla 3.9: Mapa de memoria en la Game Boy y Game Boy Color......................................62
Tabla 3.10: Contenido de una paleta de Game Boy.. 70
Tabla 3.11: Lecturas en espacio de memoria de MBC y sus efectos....................................78
Tabla 3.12: Escrituras en espacio de memoria de MBC y sus efectos................................. 78
Tabla 5.1: Títulos soportados por GINAGE...121
Tabla 5.2: Porcentajes de compatibilidad...121
Tabla 5.3: Sistemas usados en las pruebas... 122
Tabla 5.4: Diferencias de rendimiento entre Java 5 y Java 6... 125
Tabla 5.5: Porcentaje de mejora en sistema 1.. 125
Tabla C.1: Resumen del ISA de la Game Boy.. 149
Tabla C.2: Comparativa de opcodes entre el Z80 y el LR35902.......................................150

Notación

- Las referencias bibliográficas aparecen en la sección de Bibliografía ordenadas según el primer apellido del primer autor.

  Cuando se citan en el texto, las referencias aparecen encerradas entre corchetes. Se utiliza el siguiente convenio:
  - Cuando el autor es único, se usan las tres primeras letras del primer apellido, la primera en mayúscula y el resto en minúscula. El conjunto va seguido de las dos cifras correspondientes al año de edición. Un ejemplo puede ser [Lar99].
  - En caso de haber dos, tres o cuatro autores, se utilizan las iniciales de sus apellidos seguidas del año, como en [RS02], [FKT02] y [LSSH03].
  - Si hay más de cuatro autores, aparecen las iniciales de los tres primeros seguidas del carácter + y el año, como en [LMV^{+}04].
- Las páginas web a las que se hace referencia en el texto aparecen ordenadas en la sección Referencias web. Se hace referencia a ellas con el número correspondiente encerrado entre corchetes.
- Las ilustraciones y tablas se referéncian con Ilustración y Tabla respectivamete, seguidos del número del capítulo y el orden del elemento dentro del mismo, separados por un punto.
- Los diagramas que aparecen están destinados a aclarar el texto al que acompañan y están, habitualmente, simplificados. Pueden omitirse algunos atributos o métodos en una clase e incluso pueden omitirse clases que no son imprescindibles en el ejemplo que tratan de esclarecer.

Acrónimos

En el presente documento se hace referencia a algunos términos unas veces por su forma completa y otra como su acrónimo, utilizando ambas formas indistintamente. La siguiente tabla resume los acrónimos empleados junto a su forma completa:

Forma completa	**Acrónimo/s**
Videoconsola Game Boy	GB[1], DMG (Dot Matrix GameBoy)
Videoconsola Game Boy Color	GBC
Controlador de Bancos de Memoria	MBC (Memory Bank Controller)
Arquitectura de juego de instrucciones	ISA
Capa Background	BG
Capa Window	WIN
Capa de Sprites	SPRITE

1 Mientras no se indique lo contrario, GB hace referencia a todas las versiones de la videoconsola

Capítulo 1. INTRODUCCIÓN

1.1 Antecedentes

Tradicionalmente, los desarrollos software estaban orientados a una única plataforma. Así, las aplicaciones eran desarrolladas para una arquitectura en concreto o, como mucho, una familia reducida de computadoras. A principios/mediados de los 90, la popularización de Java posibilita la extensión del concepto de máquina virtual. Ésta tecnología permite la ejecución de una misma aplicación en muchas arquitecturas hardware diferentes (aunque con una potencia de cómputo similar).Posteriormente, la expansión del mercado de las PDAs lleva al desarrollo de plataformas unificadas de desarrollo. Así, Sun desarrolla diferentes versiones de la máquina virtual, cada una de ellas orientadas a un conjunto de máquinas con similares prestaciones, en el caso de las PDAs y teléfonos móviles, J2ME. Sin embargo estas plataformas tienen algunos inconvenientes:

- Se distribuyen bajo licencias privativas y tienen asociado un coste para su utilización (aún tras la reciente liberación de Java, la parte de J2ME sigue siendo cerrada)
- Los compiladores tienen unos costes elevados y/o están restringidos a unas pocas arquitecturas.

Afortunadamente en los últimos años se han desarrollado frameworks alternativos al omnipresente J2ME, como puedan ser Superwaba y Ewe que, sin ser compatibles con la máquina virtual de Sun, cumplen el eslogan de "write once, run everywhere". Estos entornos presentan varias ventajas frente a J2ME:

- Son libres con lo que son más fácilmente mantenibles y evolucionan de forma más rápida.
- Son gratuitos o tienen un coste muy bajo.
- Funcionan en un rango más amplio de dispositivos móviles.
- La API es más avanzada que la proporcionada por J2ME. Son más potentes.

Un defecto de las maquinas virtuales de estos entornos es que son diferentes a la J2ME de Sun, por lo que ambas tecnologías son incompatibles y no existen certificaciones de dispositivos que las soporten.

El concepto de emulación se puede definir como: *La técnica mediante la cual se imita el comportamiento de un sistema origen* (sistema emulado) *en otro* (sistema anfitrión o emulante) *de manera que éste último se comporta o parece comportarse para el usuario como lo hacía el*

primero. La tecnología de virtualización, no deja de ser sino una forma de emulación más compleja. La madurez de productos como VMWare, Xen o KVM del kernel Linux junto con la frenética carrera de la industria del hardware por integrar cada vez más núcleos en una misma CPU así como tecnologías hardware específicas para virtualización, demuestran que es una buena idea el poder emular varias máquinas en una única computadora. La máquina virtual de Java, que ha sido portada para múltiples plataformas heterogéneas, es otro ejemplo de emulador que goza de éxito actualmente.

Hay otros tipos de emulación pero actualmente la emulación de videojuegos antiguos en particular es un área muy prolífica, existiendo emuladores de videoconsola o arcade para Pcs, PDAs en incluso otras videoconsolas, permitiendo a los antiguos jugadores de éstos sistemas revivir sus viejos momentos delante del monitor o la televisión.

El mercado de emuladores para PC en concreto está muy bien surtido de grandes emuladores de todos los sistemas, y para todas las plataformas (en especial, Microsoft Windows) .Existen alternativas tanto libres como propietarias, gratuitas y de pago, maduras y con largas trayectorias que realizan una emulación eficiente y gozan de una compatibilidad casi perfecta. El mercado de los dispositivos móviles sin embargo presenta un panorama mucho más desolador con pocos emuladores, la mayoría propietarios y de pago, de calidad y compatibilidad medias y bastantes sistemas pendiente de ser emulados.

Ante esta situación, el desarrollo de un emulador de videojuegos para PC y PDA es una oportunidad excelente para adentrarse en el mundo del desarrollo de emuladores. Además, el desarrollo para PDA presenta retos inherentes a la propia plataforma debido a las menores prestaciones de este tipo de dispositivos y permite adentrarse en la tecnología puntera de los frameworks de desarrollo para dispositivos móviles. El objetivo de este proyecto será pues desarrollar un emulador de Game Boy para PDA y PC sobre la plataforma Ewe.

La elección de este hardware y plataforma presenta una serie de ventajas que se detallarán más adelante en este documento.

1.2 Estructura de éste documento

El presente documento está dividido en 7 capítulos, descritos brevemente a continuación:

- Capitulo 1: Presenta la descripción al problema tratado así como la estructura del presente documento.
- Capitulo 2: La descripción de los objetivos a alcanzar con el proyecto.
- Capitulo 3: En éste capítulo se tratan los conceptos sobre emulación, diferentes tipos de emuladores y técnicas asociadas a la emulación en general y sobre la emulación del

hardware de videoconsolas en particular. También se explica el hardware presente en las videoconsolas actuales y de anteriores generaciones, su funcionamiento, cómo se emula y se detalla en mayor medida el hardware de la Game Boy y su funcionamiento.

- Capitulo 4: El método de trabajo, incluyendo las decisiones de diseño tomadas durante el desarrollo y la arquitectura elegida son abordadas en este capitulo. También se detallan el ciclo de desarrollo en emulación junto a su funcionamiento, así como su aplicación a este proyecto en algunas etapas concretas.
- Capitulo 5: Este capítulo se centra en los resultados obtenidos tras la conclusión de GINAGE con datos sobre porcentajes estimados de juegos soportados, comparativa de eficiencia del mismo en diferentes arquitecturas software y hardware así como porcentajes de tiempo y memoria consumidos por el emulador. También se realiza un análisis sobre una posible explotación comercial.
- Capitulo 6: Las conclusiones finales, objetivos alcanzados y lineas futuras de mejoras que podrían abordarse.
- Capítulo 7: El listado de la bibliografía y las referencias web usadas durante el desarrollo del presente proyecto.

Los capítulos previamente enumerados se complementan con los siguientes anexos:

- Anexo A: Figuras a color
- Anexo B: El manual de usuario y preguntas frecuentes sobre GINAGE
- Anexo C: Resumen de los códigos de operación (opcode) de la Game Boy.
- Anexo D: La estructura y organización del cd adjunto a esta documentación.

Capítulo 2. OBJETIVOS DEL PROYECTO

El principal objetivo de éste proyecto es desarrollar un emulador de Game Boy para PC y PDA. El requisito de que sea ejecutable en PDA y PC limita las tecnologias de trabajo a las que soporten ambas plataformas, así como obliga a centrarse en la eficiencia del código (a veces reñida con las buenas prácticas de la ingeniería del software).

Se ha optado por una solución basada en Java para dispositivos móviles antes que usar C/C++ junto a alguna librería como SDL por el mayor rango de plataformas soportado por la solución Java frente a SDL al ser ésta específica para este tipo de plataformas.

Tras un estudio de la elección concreta de la plataforma [Rol06] se ha optado por Ewe como la mejor opción. Ya que Ewe actualmente no soporta dispositivos Palm, el emulador funcionará para Pocket PC y PC (además del resto de PDAs soportados por Ewe (ver [Rol06]).

El lenguaje de desarrollo será por tanto Java (con la API propia de Ewe) y se usará un PC convencional para el desarrollo de la aplicación, centrándose en la última etapa del desarrollo en una PDA. Finalmente, se pretenden conseguir al menos los siguientes hitos a lo largo del desarrollo del proyecto:

- Obtener una idea global del funcionamiento de un emulador, así cómo los conocimientos para llevar a la práctica una implementación real de un emulador de Game Boy para PC y PDA. Esto implica conocer muy de cerca el funcionamiento interno de la misma y su arquitectura.
- Obtener conocimientos de un framework de desarrollo para dispositivos móviles, similar a J2ME pero más flexible, libre y potente.
- Enfrentarse y resolver los problemas debidos a las prestaciones reducidas en una arquitectura mucho menos potente que la tradicional de PC. Se analizarán algunas técnicas para optimizar el código.
- El software generado deberá ser multiplataforma (Pcs y algunos tipos de PDA).
- El software será distribuido bajo la licencia libre GPL version 3.
- La creación de una web para alojar el proyecto.
- Se intentará soportar un alto número de ROMs
- Se estudiará la viabilidad de emular algunas características avanzadas del hardware como el soporte a ROMs de Game Boy Color y el sonido.

La emulación del sonido queda fuera del ámbito de éste proyecto por imposibilidad de las propias librerías (y de las propias PDAs).

Capítulo 3. ESTADO DEL ARTE

3.1 Conceptos previos sobre emulación

A continuación se expondrán los conceptos básicos sobre la emulación, los diferentes tipos de emulación que existen, algunos usos comunes, los aspectos legales de la misma y, por último, se presentará una relación de los emuladores de Game Boy más conocidos en la actualidad para diferentes plataformas.

La información sobre emuladores no se encuentra en bibliografía impresa. Debido a que las materias de arquitectura de computadores y procesadores de lenguajes están directamente relacionadas con el ámbito de la emulación, los libros sobre ambas son los que pueden resultar más útiles para obtener una base de conocimientos previa. No obstante, gran parte de la información consultada para realizar este proyecto fue obtenida de diversas fuentes de Internet que quedan reflejadas en el apartado de Bibliografía de este documento.

3.1.1 ¿Que es un emulador?

No es fácil encontrar una definición formal de un emulador puesto que, en la práctica, a veces se mezclan términos de forma errónea. Tampoco hay libros sobre la emulación en particular como ya se ha comentado. Según [1]: *Un emulador duplica (provee la emulación de) las funciones de un sistema en un sistema diferente de manera que el segundo sistema se comporta como (y parece ser) el primer sistema.*

Una definición más o menos informal podría ser "*la emulación consiste en la imitación del comportamiento de un ente software o hardware en otro ente distinto al que se intenta imitar, de manera que el usuario tenga una experiencia de interacción similar a la del ente original*".

Es preciso distinguir la diferencia entre un emulador y un simulador, conceptos ampliamente mezclados por error en la bibliografía. La principal diferencia radica en que un emulador se centra en imitar el comportamiento externo de la forma más fiel posible. Tiene que aparentar que el usuario está realmente delante del sistema original aunque la implementación (comportamiento interno) pueda parecerse o no a la del original.

Un simulador por contra, centra sus esfuerzos en imitar el comportamiento interno, es decir, el estado interno del sistema. Como ya se ha comentado, la bibliografía es confusa y para algunos autores de emuladores de videoconsolas o arcades, un emulador *imita el comportamiento externo e interno del sistema* mientras que el simulador únicamente *imita el comportamiento externo*. Según esta otra definición, un emulador sería, por ejemplo, un programa que imitase el

hardware de la máquina recreativa del Space Invaders, permitiendo ejecutar la ROM original, mientras que un simulador sería una versión del juego del Space Invaders para PC que usara los gráficos del original.

3.1.2 Tipos de emuladores

Se puede establecer una primera distinción entre los emuladores apoyados en hardware y los constituidos únicamente por software [1]:

- Emulador hardware: Se denominan así a los emuladores hardware/software. Son aquellos que utilizan hardware específico para acelerar la emulación. Los computadores Crusoe se basan en un hardware especifico para la ejecución de código 80x86 (ver [2]), consiguiendo un rendimiento muy superior al que conseguirían emulando únicamente las instrucciones de esa arquitectura en software. La emulación hardware tiene importantes aplicaciones en el campo del diseño de circuitos integrados. El sistema que está siendo diseñado puede probarse y depurarse antes de su producción en masa. Es típico el uso de circuitos FPGA para la emulación de prototipos en este tipo de emuladores.

- Emulador software: Son los más conocidos por el gran público y también los que más variedad presentan. A diferencia de los anteriores éstos no se apoyan en más hardware que el de la máquina donde se ejecutan. Se pueden subdividir atendiendo a su ámbito de aplicación en:

  - Máquinas virtuales: Crean un entorno virtualizado (software) entre el ordenador anfitrión y su sistema operativo de manera que el usuario final puede usar software sobre esa máquina. La máquina virtual puede ser idéntica a la anfitriona, diferente (Maquina virtual hardware) o incluso no existir en la realidad (Maquina virtual de aplicación). La máquina virtual de Java es un ejemplo de este último tipo: Una máquina ficticia, con un juego de instrucciones ficticio que se creó con el objetivo de que fuera fácilmente traducible a juegos de instrucciones de máquinas reales. De esta forma se consigue independencia de la arquitectura real mediante una capa software que emula una arquitectura hardware ficticia y común. El otro tipo de máquinas virtuales permiten la emulación de arquitectura reales. Software como VMWare, Xen, Microsoft Virtual PC, VirtualBox o KVM hacen posible el uso de múltiples sistemas operativos de forma concurrente sobre un mismo PC como si de máquinas independientes se tratase. Cada sistema operativo se ejecuta sobre una máquina virtual software con unos recursos virtuales y no es consciente de la presencia de los otros.

- Emuladores Arcade/Videoconsolas: Orientadas al entretenimiento lúdico, permiten rememorar los videojuegos de antaño en la actualidad. A veces son la única forma de poder ejecutar los viejos juegos de máquinas que por su antigüedad han pasado a ser piezas de coleccionista. Emuladores como MAME [3], ZSNES [4] o SCUMMVM [5] permiten ejecutar las ROMs (juegos origi-nales de éstos sistemas) de forma fidedigna. Como los anteriores, emulan una arquitectura completa dentro de otra y se centran en aspectos como el rendimiento, la compatibilidad de ROMs y la imitación lo más fiel posible de la apariencia externa del sistema original (sensación de velocidad, sonido, colores y gráficos principalmente).

Otra taxonomía, independiente de la anterior, atiende a la forma en la que están implementados los emuladores [MF01]:

- Emuladores interpretadores: La emulación consiste en un bucle infinito que lee instrucciones de memoria una a una, pasando cada instrucción por sucesivas etapas (captura, decodificación, búsqueda de operandos y ejecución.) y realizando la actualización interna del estado que haría la máquina original (básicamente registros y memoria). Además, periódicamente se realizan tareas como atención a interrupciones, actualización de timers, consultas por polling (por ejemplo para E/S) y refresco de la pantalla, etc.

```
While (CPUenEjecucion) {
   leerInstrucción();
   decodificarInstrucción();
   ejecutar(instrucción i);
}
```

Ilustración 3.1: Funcionamiento simplificado de emulador interpretador

Este tipo de emulación es la más sencilla de realizar y depurar pues se puede implementar en lenguajes de alto nivel sin tener unos amplios conocimientos de la máquina emulante, aunque es más lenta que la que estudiaremos a continuación.

- Emuladores recompiladores: Consiste en traducir el código máquina de la máquina original al equivalente nativo de la máquina emuladora. Podría entenderse como un proceso de recompilación del código a partir de un binario para otra arquitectura. Evidentemente, esta aproximación es mucho más rápida (al igual que un programa compilado es mucho más rápido que uno interpretado) que la anterior pero presenta las siguientes dificultades:

    - Si los juegos de instrucciones de ambas máquinas son muy distintos no es trivial realizar la equivalencia de una instrucción a otra.

    - Es común que el código se modifique a sí mismo en tiempo de ejecución por lo que será imposible hacer una recompilación estática (pre ejecución) y habrá que hacer una dinámica (durante la ejecución). La recompilación dinámica se realiza por unidades sin saltos, es decir, trozos de códigos con ejecución secuencial, (por ejemplo, funciones).

En la práctica se combinan ambas estrategias para recompilar todo el código posible de forma estática y el resto de forma dinámica y por secciones.

3.1.3 Los emuladores de videoconsola o arcade

A diferencia de un ordenador, que es un hardware de propósito general, las videoconsolas son piezas hardware especializadas en el entretenimiento. Se centran principalmente en la generación de gráficos y sonido para dotar al jugador de una experiencia de juego realista. Los programas para éstos sistemas son juegos en diferentes soportes que se controlan por medio de un mando o pad de control. Las ROMs son el resultado de volcar el juego original a un formato legible por el ordenador.

Las consolas de 8 y 16 bits, arcades y algunas consolas de Nintendo (Nintendo 64 y DS) usan cartuchos para los juegos. Estos almacenan el contenido del juego en uno o más chips de memoria de sólo lectura, pudiendo estar acompañados a veces de otros chips que permiten escritura para guardado del estado de la partida, puntuaciones, personalizaciones, etc.

En los sistemas modernos, los cartuchos han sido sustituidos por soportes ópticos de cada vez mayor capacidad. El termino ISO y ROM son equivalentes en este caso, pues el contenido del juego no deja de ser una imagen en formato ISO9660 o similar.

En la actualidad, hay muy buenos emuladores de todas las generaciones de consolas pasadas. Con la excepción de la Dreamcast de SEGA, la era de los 128 bits es a duras penas emulada en los Pcs actuales. De la generación actual (Wii, Xbox360 y PS3) no hay ningún emulador

disponible. Es evidente que las videoconsolas actuales han alcanzado un grado de complejidad similar o superior al de un PC y a veces son incluso superiores en potencia a éstos como es el caso de la PS3 de Sony con su procesador Cell lo cual hace imposible emularlas con un ordenador doméstico a día de hoy.

El proceso de la emulación es muy intensivo en consumo de recursos. En los foros especializados en emuladores se comenta que la máquina emuladora ha de ser unas 10 veces más potente que la emulada para poder realizar la emulación decentemente. Para mejorar la eficiencia, con frecuencia los desarrolladores recurren a implementar las partes más críticas en ensamblador (por lo general, la CPU o core "núcleo" del emulador) o hacer emuladores recompiladores en lugar de interpretadores. Otra técnica muy usada consiste en valerse del hardware de la máquina emulante para implementar por hardware algunas partes que de otro modo habría que implementar en software. Un ejemplo común es el uso de la tarjeta de vídeo para realizar el escalado de la pantalla por hardware.

La eficiencia ganada al usar ensamblador dependerá principalmente de las similitudes entre los juegos de instrucciones de la máquina emuladora y emulante: Cuanto más parecidas sean los ISA[2] de ambas maquinas, mayor ganancia de eficiencia se puede esperar. Un buen programador de ensamblador podrá explotar las capacidades del ISA de la máquina emuladora mejor que un compilador y obtener una ganancia de rendimiento considerable. Por otro lado, todos los lenguaje de alto nivel ocultan, en mayor o menor grado, muchas de las características específicas a una arquitectura que suelen ser muy útiles para la emulación. Usando ensamblador se puede, entre otros:

- Acceder directamente a los registros de la máquina emulante:
  Algunos lenguajes como C permiten asignar variables a registros pero sin poder elegir el registro concreto de la máquina. Además y dependiendo del compilador que se trate, ésto puede perjudicar al código generado por el mismo, haciendolo menos eficiente. Asignando manualmente los registros por ensamblador, se reducen drásticamente los accesos a memoria para leerlos y escribirlos. Además el acceso al registro de flags desde ensamblador como uno más, posibilita el siguiente punto.

- Usar los flags de la CPU emulante:
  El cálculo de flags mediante lenguajes de alto nivel es muy costoso. Casi todas las instrucciones de la CPU emulada tendrán calculo de flags y en muchos casos, es más costosa la emulación de las mismas que el propio calculo asociado a la instrucción. Según [MF01] el cálculo de flags en instrucciones aritmético-lógicas (las más comunes) puede llegar a suponer hasta el 30% del tiempo de emulación de la CPU. Desgraciadamente, los flags son

2 ISA (Instruction Set Architecture): El juego de instrucciones de una máquina junto a información como tipos de datos nativos, registros, modos de direccionamiento, arquitectura de la memoria, manejo de interrupciones y excepciones y E/S externa.

una de las características más dependientes de una CPU por lo que desde ningún lenguaje de alto nivel se permite acceder a los mismos. Es posible, para resultados de 8 bits (por cuestiones de espacio en memoria) precalcular los flags en tablas. Mediante ensamblador se pueden usar los flags de la máquina emulante para el cálculo de los flags del emulador, ahorrando muchas instrucciones y acelerando increíblemente los cálculos.

- Usar la ISA directamente:
  En un lenguaje de alto nivel, la implementación de una instrucción puede consistir en varias lineas de código. Si existe en la máquina emulante una instrucción similar a la emulada, puede ser mucho más rápido implementarla usando esa instrucción y corrigiendo después los posibles "efectos colaterales" de haberla usado (como corregir algún flag en concreto).
  Por ejemplo: Implementar la instrucción para el ajuste BCD en un lenguaje de alto nivel es muy costosa o requiere del uso de tablas precalculadas. En ensamblador, es posible aprovechar la propia instrucción BCD de la arquitectura x86 para emular la del z80 pues son idénticas. Las instrucciones de tipo SWAP (intercambio de parte alta y baja en un registro) también pueden beneficiarse de la instrucción nativa equivalente.

```
/* Calculate 8-bit add carry */
#define CalcFlagC(value1, value2) tC.flagC =
((((UINT16) value1 + (UINT16) value2) & 0x100)?1:0);

/* Calculate 4-bit add carry */
#define CalcFlagAc(value1, value2) tC.flagAc =
((((value1 & 0x0f) + (value2 & 0x0f)) & 0x10)?1:0);

/* Calculate Z, P and S flags */
#define CalcFlagZPS(value) tC.flagZPS = ZPSTable[value];

/* Macro for Add instructions */
#define Add(value) {
   CalcFlagC(tC.AF.b.h, value);
   CalcFlagAc(tC.AF.b.h, value);
   tC.AF.b.h = tC.AF.b.h + value;
   CalcFlagZPS(tC.AF.b.h);
}
```

Ilustración 3.2: Cálculo de flags para operación ADD de Z80 en C

```
sahf
add al, ch
lahf
seto dl
and ah, 0fbh ; Knock out parity/overflow
shl dl, 2
or ah, dl
and ah, 0fdh ; No N!
```

Ilustración 3.3: Cálculo de flags para instrucción ADD de Z80 en ensamblador x86

Generalmente, no es necesaria la emulación exacta y completa del hardware original. Probablemente no todas las ROMs utilicen todas las capacidades del hardware, ni todas las instrucciones de la CPU y algunas características de la máquina emulada serán prescindibles en mayor o menor grado (por ejemplo, el soporte multijugador).

Algunas características admiten diferentes grados de emulación que pueden ajustarse para aumentar el rendimiento global. Un caso típico de esto es la emulación de la imagen y el sonido. Algunos frames pueden saltarse, es decir, no calcularse ni pintarse en pantalla, sin que ésto afecte de forma significativa a la experiencia de juego. Esta técnica, muy común en los emuladores actuales, es conocida como *frameskip*. De la misma forma, el sonido puede ser mono en lugar de stereo, puede bajarse la frecuencia de muestreo (downsampling) o los bits por muestra, de forma que se pierde calidad pero se gana eficiencia al hacer menos cálculos.

La eficiencia puede mejorarse:

- Ahorrando cálculos siempre que no afecten de forma significativa a la experiencia: Es el caso del frameskip y el downsampling del sonido.

- Precalculando en tablas las operaciones más complejas (por ejemplo, las trigonométricas).

- Realizando todos los cálculos que se puedan por hardware en lugar de por software.

La calidad de un emulador se mide por lo útil que es para el usuario. Así, los parámetros que determinan la calidad de un emulador son:

- Compatibilidad con ROMs: Cuanto más completa y fidedigna sea la emulación, más juegos funcionarán correctamente.

- Rango de plataformas soportado: El grado en que el emulador esta disponible para diferentes Sistemas Operativos, diferentes arquitecturas y dispositivos. En ocasiones, otro autor

diferente del original porta un emulador de una arquitectura a otra. Ésto es especialmente frecuente en los ports de emuladores para PC a videoconsolas.

- Requisitos mínimos (recursos): Los requisitos mínimos de la plataforma que hará de máquina emuladora. Cuanto menor sean los requisitos que se necesiten, mayor será el porcentaje de usuarios que podrá usarlo. En el caso de las videoconsolas más modernas, los requisitos para ejecutar sus emuladores son prohibitivos pues las videoconsolas son en cada generación más complejas y tienen más componentes a emular.

3.1.4 Utilidades comunes de los emuladores

La utilidad más obvia de un emulador es la de experimentar el juego original como si se estuviera delante de la máquina nativa. Muchas veces, la máquina emulada y sus programas ya no se fabrican, por lo que se convierten en piezas de coleccionista. y la única forma de saber cómo era esa máquina es mediante un emulador. Las consolas de nueva generación incorporan emuladores integrados para permitir la retrocompatibilidad de juegos (cuando no se puede alcanzar ésta por hardware). Así, el objetivo es volver a rentabilizar económicamente juegos antiguos.

Como ya se ha citado, los emuladores también son útiles para el prototipado de hardware y para las pruebas de programas de algunas arquitecturas. Es común en el desarrollo de juegos de móvil o consola probar previamente en emuladores de la máquina para la que se desarrolló antes de volcar el programa/juego al soporte que lo admita. Las propias compañías desarrollan emuladores de alta calidad para uso interno de sus programadores con el fin de facilitar y acelerar los desarrollos.

En el mundo de la scene, los emuladores son comunes en el hackeo de ROMs, esto es, la modificación del contenido original. Se han hecho traducciones amateur de juegos que nunca llegaron a traducirse, correcciones de bugs del original, en incluso mejoras y añadidos de todo tipo: gráficas o de sonido, soporte de salvado, multijugador, soporte de trucos. Es común encontrarse en Internet diferentes versiones de una misma ROM con diferentes modificaciones. Hay gran cantidad de utilidades y programas en Internet para personalizar las ROMs y modificarlas hasta el extremo (véase [10]).

Por último, también es posible estudiar el código con fines didácticos y para comprender los fundamentos de la emulación.

3.1.5 Aspectos legales

Hay un desconocimiento generalizado sobre la legalidad actual de los emuladores en parte debido a las mentiras vertidas por algunas compañías y la falta de información. La realidad es que el desarrollo, distribución, tenencia y uso de emuladores es legal en la mayoría de países. Si el

emulador se implementa sin usar partes hardware o software bajo copyright, ni información privada de la compañía (distinta de la pública) es perfectamente legal.

- ROMs comerciales:

El uso de ROMs comerciales, sin embargo, están regidas como todo software por las leyes de copyright internacionales y por lo tanto su distribución y uso es ilegal salvo en el caso de los países que contemplan el derecho a copia de seguridad. En éste caso, el legal poseer una copia de suguridad del cartucho original, siempre que se posea éste y la copia halla sido obtenida a partir del original. La mayoría de países europeos lo contemplan, así como EEUU. Sin embargo en éste último, la nueva ley DMCA (Digital Millennium Copyright Act) considera delito la ruptura de la protección de copyright por lo que en la práctica se han ilegalizado la mayoría de las copias. En cualquier caso el uso de este derecho implica la posesión del juego original y la copia a partir de ese mismo original, por lo que las ROMs descargadas de Internet son ilegales. Existe un mito generalizado en Internet sobre que las ROMs descargadas son legales durante 24 horas, con objeto de prueba del juego, y luego han de borrarse. Esto es falso pues no existe ninguna legislación vigente en país alguno que hable de este hecho. Con la inclusión de emuladores en la nueva generación de consolas, se está empezando a perseguir con más contundencia a las páginas que ofrecen descarga de ROMs.

- Abandonware:

Se denomina así a los juegos antiguos que han terminado su rentabilidad económica (ya no se venden). Sin embargo, el concepto de abandono no existe en las leyes de copyright (sí en el de las marcas registradas) por lo que los dueños del copyright siguen teniendo los derechos de los mismos y la distribución sigue siendo ilegal (aunque raramente perseguida). Los derechos de autor, actualmente, expiran a las 75 años en EEUU y a las 50 en Europa desde la fecha de publicación de la obra. No obstante, con cada reedición de la misma son renovados. Ésta es la principal razón de que se remastericen juegos antiguos (además de por su éxito así asegurado). También es posible poder distribuir la ROM con una autorización expresa de los propietarios legales, que pueden ser una empresa o los autores originales o sus herederos en caso de personas físicas. Éste es el caso de algunas aventuras gráficas para ScummVM como *Beneath a Steel Sky* o *Flight of the Amazon Queen*.

- ROMs amateur y homebrew:

Los desarrollos amateur (homebrew) no tienen los problemas legales de las ROMs comerciales. Existen páginas dedicadas a recopilar este tipo de juegos que normalmente pueden distribuirse sin problemas legales.

En resumen, es ilegal distribuir software bajo copyright. Esto incluye ademas de las ROMs, la propia BIOS del sistema, necesaria para el correcto funcionamiento de muchos de los emuladores de la última (quinta) generación. Mientras un emulador no incluya ninguno de los anteriores en su descarga, es perfectamente legal.

3.1.6 La obtención de ROMs

Ya se ha visto que los soportes preferidos por las videoconsolas y arcades ha sido tradicionalmente el de los chips de ROM y más recientemente los discos ópticos. Para poder usar las ROMs, los emuladores necesitan que éstas sean ficheros binarios por lo que hay que realizar un proceso previo de conversión o volcado (dump en ingles) del contenido original del juego.

3.1.6.1 Contenido de los cartuchos

Los cartuchos de las videoconsolas suelen llevar uno o más chips ROM con el contenido del juego, esto es, una probable cabecera con metadatos sobre la propia ROM, el código máquina (instrucciones de la CPU de la videoconsola) y los datos (gráficos y sonido). Éstos pueden estar comprimidos para ahorrar espacio, normalmente en formatos propios. Normalmente no existe una separación real entre código y datos por lo que éstos se almacenan de forma consecutiva o en regiones del cartucho arbitrarias. El cartucho lleva tantos chips como necesite para almacenar toda la información. En cualquier caso cada fabricante de hardware tiene su propio formato interno de ROM, con o sin cabecera, con diferentes tamaños y tipos, etc.

Las máquinas arcade son hardware dedicado, mucho más potente que el de sus hermanas de sobremesa. Consisten en placas a las que está acoplada el juego y éste consta de múltiples chips ROM en los que, a diferencia de los de videoconsola, sí suelen estar separados los datos del código en diferentes chips.

Es común el uso de chips específicos tanto en arcades como videoconsolas junto con las propias ROMs y la pequeña memoria escribible. Estos chips suelen ser de dos tipos principalmente:

- Controladores de bancos de memoria: Se encargan de gestionar los accesos a los diferentes bancos de memoria ROM. Serán explicados más adelante en este documento, cuando se detalle el hardware de la Game Boy.

- Chips que apoyan a la CPU principal: Liberan del procesamiento de ciertas tareas específicas a la CPU principal o permiten el uso de ciertos efectos que sin ellos seria muy costosos hacerlos por la CPU. El ejemplo más típico son los chip SuperFX1 y SuperFX2 que permitían, incorporados a un cartucho de Super Nintendo, el procesamiento de polígonos de forma eficiente, revolucionando en su día el mercado de las videoconsolas con los primeros

gráficos en 3D. Las placas arcade también pueden llevar chips de apoyo, por ejemplo, un DSP para la producción de sonido de alta calidad.

3.1.6.2 Contenido de los soportes ópticos

En sistemas más recientes se ha abandonado el uso del cartucho por soportes ópticos de, cada vez, mayor capacidad. Las razones de este progresivo abandono han sido varias:

- Mayores necesidades de espacio: Conforme la calidad gráfica de los juegos ha ido mejorando, la necesidad de espacio ha ido creciendo de forma exponencial. El espacio necesario para almacenar una imagen se multiplica por 4 cuando la resolución se dobla. Antes con un cartucho del orden de MB había espacio más que de sobra. En la actualidad, el uso de texturas de alta resolución están obligando al uso del dvd e incluso hd-dvd o blueray.

- Abaratamiento de coste de unidades lectoras y soportes ópticos: En la era de los 80 y principios de los 90, el coste de una unidad lectora de cd y el propio cd eran mayores que el de un cartucho de pocos MB. Por otro lado, no tenía sentido esa cantidad de espacio para las posibilidades de la máquina. Además, las compañías más reticentes como Nintendo, apostaron por el cartucho durante un tiempo más que las otras, pues al ser más complicado de copiar disminuye la copia ilegal.

Los soportes ópticos almacenan una gran cantidad de datos en comparación al propio código del juego. Generalmente se usan formatos estándar y con compresión con poca pérdida para el almacenamiento de las texturas. El sonido suele almacenarse en calidad de cd como pistas de audio, en la generación actual, en 5.1. También es común la inclusión de escenas cinemáticas (FMV de Full Motion Video en ingles). El formato lógico de estos discos no suele ser estándar, y se requiere de una unidad grabadora especial para grabarlos. En la actualidad, las consolas utilizan tecnología de cifrado de la información del disco para evitar su posible volcado.

3.1.6.3 El volcado

El volcado (dumpeado en ingles) de juegos de cartucho se realiza por lo general mediante el uso de dispositivos dedicados de fabricación artesanal. Estos dispositivos son específicos para cada tipo de cartuchos y para las consolas más extendidas pueden encontrarse a la venta principalmente por Internet. En ocasiones el volcado requiere conocimientos de electrónica y soldadura. Esto es especialmente cierto con las placas de maquinas arcade, en la que el volcado es casi específico de cada juego.

En el caso de la GB y GBC, existen programadores de cartuchos que permiten hacer un

backup del contenido de la ROM de un cartucho, leer y escribir su RAM o escribir en la propia ROM en los cartuchos con chip de ROM reemplazado (haciéndolos escribibles). Los datos son traspasados a un PC mediante puerto RS-232, paralelo o USB dependiendo del modelo. Puede encontrarse más información sobre los flasheadores y cartuchos flash para GB y GBC en [6].

Ilustración 3.4: Programadores de cartucho y cartuchos reescribibles
Primera fila: programador casero de cartuchos (GB Flasher). Segunda Fila: Programadores comerciales. Tercera Fila: De izquierda a derecha: Cartucho casero programable a partir de uno original, cartucho programable de Nintendo para desarrolladores, dos cartuchos programables comerciales

La obtención de imágenes de soportes ópticos es bastante más sencilla. Por lo general, tras descubrirse algúna puerta trasera que permita leer el contenido o tras un filtrado de las especificaciones del sistema de archivos, suelen aparecer por Internet programas que lo permiten, bien utilizando el propio lector de la consola para extraer (ripear) la información o bien mediante el uso de drivers modificados para algunos modelos de lectores y un PC convencional.

Tras el volcado de la ROM o la imagen, puede ser necesario un proceso previo de adecuación de la misma. Para facilitar su procesamiento por parte de los emuladores, las ROMs suelen modificarse (simplificarse) para que las regiones de código y datos sean consecutivas. También es común, en las ROMs que no llevan algún tipo de cabecera, la inclusión de una que aporte metadatos sobre la estructura, tamaño y tipo de ROM entre otros. Como los chips de memoria tienen capacidades que aumentan en potencia de dos, es frecuente que muchos juegos no aprovechen el 100% de la capacidad de los chips. En estos casos el fabricante rellena con 0 o 1 el resto de posiciones de memoria hasta el final (bits basura) por lo que es posible eliminar este trozo y hacer la ROM más pequeña.

En la actualidad, los fabricantes intentan ponérselo difícil a los hackers usando soportes ópticos no estándares o encriptando el contenido. Nada que ver con las primeras videoconsolas y ordenadores donde las ROMs se almacenaban en cinta magnética, siendo extremadamente sencillas de copiar. Simplemente se necesitaba un casete reproductor de audio cuya salida estuviera conectada a la entrada de una tarjeta de sonido. Posteriormente, el fichero de audio resultante se convierte en una ROM binaria mediante un programa específico.

3.1.7 Emuladores conocidos de Game Boy para diferentes plataformas

La Game Boy ha sido emulada en multitud de plataformas. Desde arquitecturas hardware PC (y PowerPC) para multitud de sistemas operativos (Windows, GNU/Linux y otras variantes Unix, MS-DOS, Macintosh, MacOX, BeOS...), pasando por los clásicos computadores Amiga, hasta los actuales dispositivos móviles tipo PDA (Pocket PC y PalmOS) y algunos teléfonos móviles (Symbian principalmente). Por supuesto, también hay emuladores para consolas de sobremesa (PSX, Nintendo 64, Dreamcast, Gamecube, Xbox...) y para portátiles (Game Boy Advance, Nintendo DS, GP32, GP2X, N-Gage y PSP). Ante esta variedad de plataformas y sistemas operativos se tratará de poner orden mediante una taxonomía de los mismos, agrupándolos por diferentes características comunes y parámetros y mostrando algunos ejemplos de los emuladores más exitosos.

3.1.7.1 Emuladores de Game Boy para PC y Power PC

La arquitectura PC está sobradamente surtida de emuladores de todos los sistemas y específicamente de GB y GBC. Los mejores emuladores, se encuentran en PC y concretamente para

sistemas operativos Windows. No es de extrañar pues los desarrolladores de emuladores buscan llegar al mayor público posible.

Suelen estar escritos en C y/o C++, pues son dos lenguajes muy eficientes y con compiladores para multitud de arquitecturas y sistemas operativos. Aunque algunos desarrolladores desarrollan su emulador en C/C++ puro, otros se apoyan en librerías gráficas, de sonido y de soporte de dispositivos de E/S como SDL o DirectX. También se suele utilizar OpenGL y en mucha menor medida, librerías para desarrollo de videojuegos genéricas como Allegro. Para mejorar la eficiencia, algunos emuladores implementan las partes del código más críticas en ensamblador. Otros, como los escritos en Java, optan por la portabilidad frente a la eficiencia y tienen la ventaja de poder ejecutarse en todas las plataformas donde esté portada la MV. El resto de los lenguajes de programación son marginalmente utilizados: a veces se trata de emuladores con bugs y poca compatibilidad, otras llegan a ser una prueba de concepto y muy pocos alcanzan la madurez y calidad de sus hermanos escritos en los lenguajes mayoritarios.

En definitiva, podemos determinar que la inmensa mayoría de los emuladores de PC están escritos en C o C++ (a veces apoyados en librerías) y algunos en Java.

A continuación se citan algunos de los más extendidos y sus características:

- KIGB: Actualmente considerado el mejor emulador de GB y GBC existente, tiene una compatibilidad casi perfecta.

- VisualBoy Advance: Popular emulador principalmente de GBA pero con buen soporte a GB y GBC.

- NO$GMB: Uno de los más antiguos y de los que menos recursos requieren (funciona en un 386 a 33Mhz). El autor del mismo es también el creador y principal aportador de información del documento pandocs, usado frecuentemente durante el desarrollo de GINAGE.

- JavaBoy: uno de los mejores emuladores de GB hechos en java. Su autor, Neil Millstone, es también autor del port de ScummVM para Nintendo DS. Se ha recurrido en alguna ocasión a él durante el desarrollo de GINAGE cuando la documentación era contradictoria o inexistente y durante la corrección de bugs mediante el depurador.

  Existen muchos otros emuladores muy buenos y conocidos como BGB,TGB Dual (GPL). Mención aparte merece el emulador gnuboy. Aunque ha sido superado en compatibilidad por otros emuladores, fue uno de los primeros emuladores GPL cuando toda la scene estaba

dominada por emuladores con licencias propietarias. Escrito en C y con un diseño modular y centrado en la portabilidad, ha sido portado a diferentes plataformas incluidas videoconsolas, PDAs, teléfonos móviles, etc.

General				
Nombre	KIGB	VisualBoy Advance	NO$GMB	JavaBoy
Licencia	Freeware	GPL	Shareware (juegos de GBC necesitan key)	GPL
Lenguaje	ANSI C + Allegro	C ++ y Ensamblador + SDL o DirectX (2 versiones)	Ensamblador	Java
Plataformas	Windows, GNU/Linux, MS-DOS, MacOSX	Windows, GNU/Linux, BSD, Unix POSIX, MacOSX, BeOS	Windows, MS-DOS	Todas las soportadas por una MV java compatible.
Compatibilidad	La más alta	Muy Alta	Muy Alta	Muy Alta
Web	http://kigb.emuunlim.com	http://vba.ngemu.com	http://www.work.de/nocash/gmb.htm	http://www.millstone.demon.co.uk/download/javaboy/
Caracteristas específicas soportadas				
MBCs	1,2,3,5,7,HuC1,HuC3,GB Camera	1,2,3,5,7,Huc1,Huc3	1,2,3,5,7,Huc1,Huc3	1,2,3,5
GBC	Si	Si	Si	Si
SGB	Si	Si	Si	Si
Tamaño de pantalla	1x 2x 3x y pantalla completa	1x 2x 3x 4x y pantalla completa a diferentes resoluciones	-	1x 2x 3x 4x
Filtros gráficos	Super 2xSaI, Super Eagle, scanline y bilinear	2xSaI, Super 2x Sai, Super Eagle, pixelate y motion blur	-	No
Sonido	Si	Si	Si	Si
Depurador integrado	No	Si	Si (de calidad profesional)	Si
Trucos	Si (GameGenie y GameShark)	Si (GameGenie y GameShark)	Si (GameGenie y GameShark)	No
Otras características				

Game Link (multiplayer)	En el mismo PC, y en red TCP/IP: local o Internet (muy lento)	No	Si (en la misma máquina)	En red TCP/IP: local o Internet. Muy pocos juegos funcionan.
Puerto Infrarojos	No	No	No	No
Paletas personalizadas	Si (juegos GB)	Si (juegos GB)	Si (juegos GB)	No (varios juegos predefinidos)
Otros	Roms comprimidas en Zip o gZip. Efectos gráficos experimientales, Game Boy printer, frameskip y velocidad ajustable, 4 jugadores en SGB, capturas de pantalla	Roms comprimidas en Zip o gZip, frameskip y velocidad ajustable, capturas de pantalla	frameskip y velocidad ajustable, capturas de pantalla	Roms comprimidas en Zip o gZip, frameskip y velocidad ajustable

Tabla 3.1: Resumen de características de emuladores de GB para PC

Los filtros gráficos presentes en algunos de estos emuladores, permiten mejorar la calidad de la imagen percibida. Debido a que la imagen del hardware original tienen una resolución inferior a la de los Pcs actuales, se requiere de filtros de escalado para aumentar la resolución. Los filtros clásicos de interpolación como bilinear y bicúbico interpolan el color de los pixels vecinos e introducen cierto emborronamiento en la imagen, estropeando los bordes de las lineas nítidos. Por el contrario la interpolación al vecino más cercano, aunque conserva los bordes, hace que las diagonales y lineas curvas se vean pixeladas. El mejor algoritmo para escalado ha de ser adaptativo:

- Hacer interpolación en zonas de la imagen con tonos continuos .
- Preservar la nitidez en lineas rectas (no diagonales o curvas) p.e. aplicando la interpolación al vecino más cercano.
- Suavizar las diagonales y curvas con anti-aliasing.

Los algoritmos más comunes en los emuladores de PC son:

- Eagle, supereagle, 2 x SaI y super 2 x SaI: Algoritmos de interpolación espacial (aplicado a un único frame) adaptativos. Doblan la resolución vertical y horizontal de la imagen y los nuevos pixels son generados detectando patrones como lineas y bordes e interpolando píxeles adicionales usando anti-aliasing

- Bilinear: Es un algoritmo clásico de escalado por interpolación espacial. El valor de un pixel en la imagen escalada se obtiene como una interpolación de sus píxeles vecinos, esto es, con una ecuación que da diferentes pesos a cada pixel vecino en su contribución para general el nuevo color. Como ya se ha dicho, tiene el problema de la pérdida de definición en lineas y curvas.

- Pixelate: Algoritmo estético que muestra una pixelación exagerada de los gráficos, dando una sensación de videojuego clásico.

- Scanline: Algoritmo estético aplicado a las lineas horizontales que proporciona la sensación de estar ante una pantalla de TV.

- Motion Blur: Realiza una interpolación temporal (aplicado a varios frames) generando una imagen intermedia entre un frame y el/los anterior(es)/posterior(es). El anti-aliasing temporal se basa en generar un frame intermedio a partir de típicamente otros dos consecutivos. Ésto permite mejorar la sensación de fluidez en los movimientos a base de emborronar ligeramente algunos detalles.

3.1.7.2 Emuladores de Game Boy para videoconsola

Aunque el número de emuladores de GB para PC es enorme (varias decenas) aquí el panorama cambia. Es habitual encontrarse entre uno y tres por plataforma. Todos los emuladores para consola están hechos en C y/o ensamblador nativo debido a su mayor velocidad y a que las librerías homebrew para cada consola están hechas en éste lenguaje. La mayoría de ellos son ports de las versiones de PC y en las plataformas menos potentes presentan importantes restricciones: Menor compatibilidad, peor calidad gráfica o de sonido, lentitud, baja calidad o ausencia de escalado, etc. Esto es debido principalmente a que el port no está optimizado para aprovechar el hardware de la máquina. Sin embargo también existen buenos ports y emuladores nativos creados de cero, con las partes críticas en ensamblador y que realizan todos los cálculos posibles mediante hardware. Resulta especialmente útil en este sentido el hardware de tiles presente en algunas consolas de poca o media capacidad. Las consolas con una scene más rica actualmente son la Dreamcast, GP32, GP2X, PSP, Xbox y DS. A continuación se citan algunos de los más conocidos para algunas plataformas:

- Dreamcast: Es la consola donde el fenómeno de desarrollo de emuladores explotó masivamente. Posee 2 emuladores de GB. El mejor emulador disponible es Dcgnuboy. Se trata de uno de los múltiples ports de gnuboy, el emulador más portado de GB con diferencia.

- GP32: Consola portátil dedicada principalmente al homebrew. Actualmente cuenta con 3 emuladores de GB. El mejor emulador disponible es fGB32 (port de gnuboy).

- XboX: La primera consola de Microsoft dispone de 2 emuladores de GB específicos (entre ellos un port de gnuboy) además de varios ports de VisualBoy Advance. Uno de éstos ports, Xboy Advance, es el mejor emulador disponible.

- Nintendo DS: La última portátil de Nintendo. A la scene de ésta consola hay que sumarle la de la GBA. 3 emuladores (con port de gnuboy incluido) para DS y 6 (más variantes) para GBA. Emuladores realmente utilizables son 4, de los cuales, Lameboy y GoombaColor (GBA) son los más avanzados.

General				
Plataforma	Dreamcast	GP32	XboX	Nintendo DS
Nombre	DCgnuboy	fGB32	XboyAdvance	Lameboy
Licencia	GPL	GPL	GPL	Freeware
Lenguaje	C++ y ensamblador SH4	C y ensamblador ARM9	C + ensamblador + ¿DirectX?	C ¿y ensamblador?
Compatibilidad	Alta	Alta	Muy Alta	Media-Alta
Web	http://www.pqrs.org/~tekezo/dreamcast/software/dcgnuboy	http://www.emulnation.info/retrodev/forum/viewtopic.php?t=75	http://xport.xbox-scene.com/xboyadvance.php	http://lameboy.ath.cx/
Caracteristas específicas soportadas				
MBCs	1,2,3,5,7,HuC1,HuC3	1,2,3,5,7,Huc1,Huc3	1,2,3,5,7,Huc1,Huc3	1,2,3,5
GBC	Si	Si	Si	Si
SGB	No	No	Si	Si
Tamaño de pantalla	Varios niveles de ajuste a TV	Varios niveles de ajuste a pantalla	Varios tamaño y ajuste de posición	1x o pantalla completa (bilinear)
Filtros gráficos	No	Si	No	No
Sonido	Si	Si	Si	Si
Trucos	No	No	Si (GameGenie y GameShark)	No
Paletas personalizadas	No	No	Si (juegos GB)	No
Otros	Frameskip	Roms comprimidas en Zip o gZip, frameskip, capturas de pantalla	Roms comprimidas en Zip o gZip, frameskip, capturas de pantalla	Roms comprimidas en Zip o gZip, frameskip

Tabla 3.2: Resumen de características de emuladores de GB para algunas videoconsolas

3.1.7.3 Emuladores de Game Boy para Dispositivos móviles

Dispositivos Palm:

El hardware y software de los Palm ha evolucionado mucho. Los primeros modelos de PalmOS (versiones 1-3.3) funcionaban en procesadores Motorola m68k (generalmente el Freescale DragonBall) y permitían resoluciones de 160x160 en monocromo. Palm 3.5 introdujo el soporte a color y ya con la serie 5 se dio soporte a procesadores ARM, y los sistemas Palm experimentaron un salto de potencia y resolución considerable. A diferencia de los emuladores para videoconsolas, no se dispone de un hardware de tiles en que apoyarse para hacer la emulación por lo que ésta se complica aún más si cabe. Podemos encontrar emuladores compatibles con los Palm antiguos (con CPUs 68000) y con los nuevos (basados en ARM). Entre los primeros destacan Liberty y Phoinix y entre los segundos Mario. o Little John Palm (éste ultimo emula varias máquinas además de la GB).

General				
Nombre	Liberty	Phoinix	Little John	Mario
Licencia	Shareware (trial de 30 dias + pago)	GPL	GPL	GPL
Lenguaje	Ensamblador m68K	Ensamblador m68K	Ensamblador ARM	C++ y Ensamblador ARM
Compatibilidad	Media	Media	Alta	Media-Alta
Web	http://www.gambitstudios.com	http://phoinix.sourceforge.net	http://little-john.net	http://sourceforge.net/projects/mario/
Caracteristas específicas soportadas				
Version Palm	3.0 o superior	3.0 o superior	5.0 o superior	5.1 o superior
GBC	No	No	Si	No
SGB	No	No	No	No
Tamaño de pantalla	1x	1x	varios, zoom	1x
Filtros gráficos	No	No	No	No
Sonido	Si (muy malo)	No	Si	-
Trucos	No	No	No	No
Paletas personalizadas	Si (En Palm en color)	No	No	No
Otros	Frameskip	Frameskip, capturas de pantalla	Frameskip, ROMs comprimidas en Zip	Frameskip, ROMs comprimidas en Zip

Tabla 3.3: Resumen de características de emuladores de GB para Palm

Dispositivos Pocket PC:
Los modelos más antiguos de pocket PC usan una CPU MIPS, SH3 o Xscale y sistema operativo windows CE o Pocket PC. Los más modernos usan procesadores ARM y Windows Mobile (2003 en adelante). Por lo general son bastante más potentes que los modelos Palm y al funcionar con Windows tienen muchos emuladores portados directamente de PC. Los más conocidos de Game Boy son:

- MorphGear: En realidad es un framework construido a base de plugins. Cada plugin aporta la emulación de uno o varios sistemas. El framework es gratuito y con una licencia open source propia. Los plugins dependen de diferentes desarrolladores a los del framework y tienen diferentes licencias. En concreto, el de GB/GBC es shareware con version trial. Su desarrollador es Marat Fayzullin, uno de los personajes más destacados en la scene de la emulación y el plugin es una versión de su emulador Virtual Game Boy (VGB).

- SmartGear: Un emulador multisistema, entre ellos GB y GBC. Junto con MorphGear son los emuladores más usados por usuarios de pocket PC.

- Varios ports de gnuboy: Se han realizado múltiples ports y forks de ports de gnuboy. Los más famosos son gnuboy CE (así como sus variantes y forks para diferentes arquitecturas) y PocketGnuboy. Desafortunadamente, el desarrollo de estos emuladores está abandonado y algunas de las páginas ya no existen.

General		
Nombre	MorphGear + VGB plugin	SmartGear
Licencia	Framework: Open source propia plugin: shareware (trial) + versión de pago	Shareware (trial) + versión de pago
Lenguaje	C + Ensamblador ARM	C + Ensamblador ARM
Compatibilidad	Muy Alta	Muy Alta
Web	http://www.spicypixel.com/xwiki/bin/view/MorphGear/WebHome	http://www.bitbanksoftware.com/SmartGear.html
Caracteristas específicas soportadas		
Version Windows	Windows Mobile, Pocket PC 2000-2002 y Handheld PC	Windows Mobile 2003 y posteriores
GBC	Si	Si
SGB	Si	Si
Tamaño de pantalla	1x	1x
Filtros gráficos	No	No
Sonido	Si	Si
Trucos	Si (GameGenie y GameShark)	-
Paletas personalizadas	No (paleta verde o gris)	-
Otros	Frameskip, graba audio en MIDI, ROMs comprimidas en Zip	Frameskip, ROMs comprimidas en Zip

Tabla 3.4: Resumen de características de emuladores de GB para Pocket PC

Otros emuladores:		
Nombre	*Descripción*	*Web*
gnuboyCE	Ports de gnuboy	http://www.geocities.com/gnuboyce/
gnuboyBE		http://www.geocities.com/dwall/gnuboybe.html
PocketGnuboy		http://www.freewareppc.com/utilities/pocketgnuboy.shtml
PalmGB	Emulador de GB y GBC descontinuado	http://www.revolution.cx/PalmGB.htm
YameCE	Emula varias plataformas, entre ellas GB y GBC. Descontinuado.	http://www.mobilefan.net/Pocket-PC.nsf/Download-Free/Simulation-Game-YameCE-V0.38g-for-WinCE3.0-to-PPC2003

Tabla 3.5: Otros emuladores para Pocket PC

Teléfonos móviles: Gracias al auge de los teléfonos con sistema operativo Symbian, el aumento de prestaciones de los mismos y, especialmente, el auge de J2ME, en la actualidad hay varios emuladores de Game Boy para celulares. Algunos de los juegos clásicos que se pueden descargar previo pago, incorporan su propio emulador embebido. A continuación se

citarán los más importantes:

- Dispositivos Symbian (Smartphones):

  - VGB: El antes citado como emulador para Pocket PC, también está disponible para los smartphone Symbian. La mayoría de los móviles Symbian son de Nokia (consolas Ngage, serie 60 y superiores).

  - Gamephone Advance: A pesar de su nombre, no es capaz de reproducir juegos de GBA. Sin embargo es un buen emulador de GB y GBC.

- Teléfonos J2ME: La mayoría usan el perfil MIDP 2.0. Éste es el perfil para teléfonos móviles y es el menos potente de todos (subconjunto mínimo de J2SE). Existen pocos emuladores independientes para J2ME, esto es, sin contar los embebidos en las páginas de descargas de juegos.

  - MeBoy: Posiblemente el mejor emulador disponible en J2ME. Es un port de JavaBoy optimizado.

  - Mario: Además de la ya vista versión nativa para Palm, posee una escrita en J2ME.

  - MJavaBoy: Otro port de JavaBoy menos avanzado que MeBoy.

Emuladores para dispositivos Symbian				
General				
Nombre	VGB	SuperGoBoy	vBoy	Gamephone Advance
Licencia	shareware (trial) + versión de pago	shareware (trial) + versión de pago	shareware (trial) + versión de pago	shareware (trial) + versión de pago
Lenguaje	C + Ensamblador ARM	¿Ensamblador ARM?	¿Ensamblador ARM?	¿Ensamblador ARM?
Compatibilidad	Muy Alta	Muy Alta	Muy Alta	Muy Alta
Web	http://fms.komkon.org/VGB	http://www.wildpalm.co.uk/SuperGoBoy12.html	http://vampent.com/vboy.htm	http://www.allaboutsymbian.com/software/item/GamePhone_Advance.php
Caracteristas específicas soportadas				
Sistemas soportados	Nokia serie 60 con Symbian OS 9	Nokia serie 60, 80 y 90	Nokia serie 60	Nokia serie 60
GBC	Si	Si	Si	Si
SGB	Si	No	Si	No
Tamaño de pantalla	1x y ajuste a la pantalla	1x	1x y pantalla completa	1x y pantalla completa
Filtros gráficos	No	No	No	No
Sonido	Si	Si	Si	Si
Trucos	Si (GameGenie y GameShark)	No	No	No
Paletas personalizadas	No (paleta verde o gris)	No (paleta verde o gris)	No	No
Otros	Frameskip, graba audio en MIDU, roms comprimidas en Zip	Graba audio en MIDI, roms comprimidas en Zip o Rar, envío de ROMs por IR o Bluetooth	graba audio en WAV, capturas de pantalla en BMP, roms comprimidas en Zip	Roms comprimidas en Zip

Tabla 3.6: Resumen de características de emuladores de GB para móvil Symbian

Emuladores para dispositivos J2ME			
General			
Nombre	MeBoy	Mario	MJavaBoy
Licencia	GPL	GPL	GPL
Compatibilidad	Alta	Media-Alta	Alta
Web	http://arktos.se/meboy/index.php	http://sourceforge.net/projects/mario	http://mjavaboy.latinowebs.com
Caracteristas específicas soportadas			
Sistemas soportados	MIDP 2.0	MIDP 2.0	MIDP 1.0 y 2.0
GBC	Si	No	Si
SGB	No	No	No
Tamaño de pantalla	1x y rotaciones de 90º	1x	1x
Filtros gráficos	No	No	No
Sonido	No	No	No
Trucos	No	No	No
Paletas personalizadas	No	No	No
Otros	Frameskip	Frameskip, roms comprimidas Zip	Frameskip

Tabla 3.7: Resumen de características de emuladores de GB para móvil J2ME

3.1.7.4 Emuladores de Game Boy para otros dispositivos móviles

iBoy: Emulador para los iPod de Apple preparado para ejecutarse sobre ipodlinux. Se trata de otro port de gnuBoy que utiliza técnicas de recompilación dinámica.

General	
Nombre	iBoy
Licencia	GPL
Compatibilidad	Media
Web	http://sourceforge.net/projects/iboy
Caracteristas específicas soportadas	
Plataformas soportadas	iPod 3G-5G, iPod Mini, iPod Nano
GBC	Si (juegos con color si la pantalla lo admite)
SGB	No
Tamaño de pantalla	1x y escalado en pantallas pequeñas
Salvado de partidas	Si
Filtros gráficos	No
Sonido	Si
Trucos	No
Paletas personalizadas	No
Otros	Frameskip

Tabla 3.8: Resumen de Características del emulador iBoy para iPod

3.2 Breve historia de las portátiles de 8 bits de Nintendo

Nintendo es una de las empresas más antiguas de la actualidad. Fundada en 1889, comenzó fabricando juegos de cartas. Desde el lanzamiento de su primera videoconsola, la Nintendo TV Game 6, se ha establecido en el mercado como una de las compañías más importantes respecto a innovación y manteniendo una base de usuarios fieles a sus sistemas.

La primera videoconsola portátil de Nintendo fue la Game & Watch, lanzada en 1980. Conocidas popularmente como “maquinitas”, estos sistemas llevaban una única ROM incorporada la mayoría de las veces incluyendo dos juegos (GAME A y GAME B), siendo el juego B una version similar del A pero más difícil. Mostraba unos gráficos simplistas en un LCD además de poseer funciones de reloj y alarma. En éstas consolas se introdujo el PAD de control (la popular “cruceta” de todos los mandos de consola modernos). Debido a su éxito, fueron lanzadas cerca de sesenta máquinas diferentes.

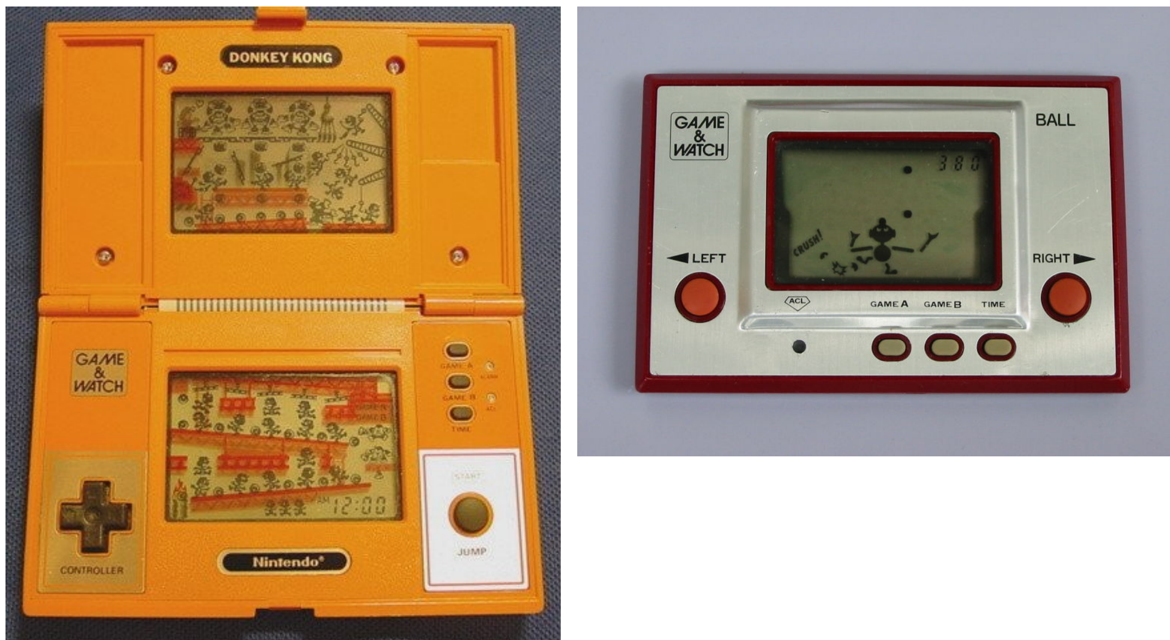


Ilustración 3.5: Dos Game & Watch

A la izquierda Donkey Kong, un clasico a doble pantalla. A la derecha Ball, el primer juego de Game & Watch

La primera versión de la Game Boy data de 1989. Con una pantalla monocroma con 4 tonos de verde de 2.6 pulgadas, sonido monoaural (stereo con cascos) surgió como sucesora de la Game & Watch y fue todo un éxito. A diferencia de su predecesora, la Game Boy permitía el uso de múltiples juegos en la misma consola, gracias al uso de cartuchos intercambiables. Logró imponerse a sus competidoras contemporáneas, la GameGear de SEGA y la Atari Lynx, vendiendo la increíble cifra de 70 millones de consolas en todo el mundo. Ésto fue debido principalmente a:

- Mayor duración de baterías: Con 4 pilas AA la duración podía extenderse hasta 35 horas gracia al uso de un LCD no retroiluminado.
- Menor precio y tamaño: Era realmente pequeña y ligera para su época: 148mm x 90mm x 32mm y 394gr (pilas incluidas).
- Mayor catálogo de juegos: Nintendo tenia acuerdo con múltiples licenciatarios (Third-partys) de NES y SNES por lo que el desarrollo de juegos para ésta plataforma estaba asegurado.

En 1996 Nintendo saca al mercado un rediseño de Game Boy, la Game Boy Pocket:

- Mas pequeña y ligera: 124mm x 76mm x 23mm y 148gr (pilas incluidas).
- Pantalla más nítida: blanco y negro y sin el molesto efecto de pantalla borrosa del anterior modelo)
- Menos baterías: dos pilas AAA con hasta 10 horas de duración.

La Game Boy Color surge en 1998 con un rediseño hardware importante. Con una pantalla a color y retrocompatible con el catálogo de sus hermanas anteriores. De nuevo la historia se repite: inferior tecnológicamente a sus contemporáneas (Neo Geo Pocket de SNK y Wonderswan de Bandai) consigue vender 49 millones de unidades.

- Dimensiones y peso: 133mm x 75mm x 27.4mm y 138gr (pilas incluidas)
- Colores: 32768, máximo 56 en pantalla

Ilustración 3.6: Linea temporal de la Game Boy

De izquierda a derecha: Game Boy, Game Boy Pocket y Game Boy Color

Hay otras portátiles posteriores a la Game Boy Color como la Game Boy Advance y la reciente Nintendo DS, mucho más potentes que ésta. Se puede encontrar más información sobre la historia de las portátiles de Nintendo en [7] y sobre la historia de los videojuegos en general en [8].

3.3 El hardware de una videoconsola

Se pueden distinguir claramente una evolución en el tipo de hardware incorporado por las videoconsolas según se han sucedido las generaciones en los últimos años.

3.3.1 Consolas de 8 y 16 bits

En las primeras videoconsolas, los desarrolladores programaban los juegos accediendo directamente al hardware de la máquina. Muy pocos eran los juegos programados en un lenguaje de más alto nivel que ensamblador y el uso de librerías o kits de programación era prácticamente nulo. Cada desarrolladora se creaba sus pequeñas librerías para sus juegos y las usaba para múltiples desarrollos. La dificultad de depuración de este tipo de juegos es obvia.

Respecto al hardware, el procesador central usado era uno de tipo RISC estandard para la época o, como mucho, adaptado por el fabricante con pequeñas modificaciones. Así, la Génesis (o Mega Drive como se la conoce en Europa) utiliza un 68000. Otras como la Game Boy, GameGear o Master System utilizan un Z80 (básicamente un 8080 con añadidos). En general, eran procesadores

poco potentes y baratos, pues la reducción de costes era primordial para permitir al fabricante vender la videoconsola con un precio más barato que la competencia. Además la CPU estaba liberada de la mayoría de las tareas gráficas gracias al VDP (Video Digital Processor).

Un VDP es el hardware gráfico precursor al de las videoconsolas y Pcs actuales. Cada fabricante diseñaba su propio hardware específico a veces con potencia superior a la de la propia CPU de la máquina. En los años 80 y principio de los 90, el coste del MB de memoria era muy elevado, por lo que estos sistemas no podían permitirse el lujo de poseer mucha si querían ser económicamente competitivos. Por otro lado, debían aprovechar la poca memoria de la que disponían de forma eficiente, mostrando los mejores gráficos posibles y evitando tener a la CPU y los buses constantemente ocupados en transferencias de datos de memoria principal a memoria de video. La solución a todos éstos problemas consiste en el uso de los *tiles* y *sprites*.

Un "tile" es una agrupación de pixeles con forma de paralelogramo (8x8, 8x16, 32x32, etc). Un sprite es un tile con coordenadas (puede ubicarse en pantalla en cualquier posición a diferencia de un tile que solo puede ubicarse en posiciones especificas, esto es, múltiplos de sus dimensiones). La imagen en pantalla se construye a base de múltiples tiles como si de pequeños baldosines se tratara, colocados en múltiples capas. Las capas se pintan en orden de prioridad, tapando las superiores el contenido de las inferiores como si de múltiples acetatos se tratase. Encima de los tiles, se sitúan los sprites. El VDP suele implementar algunos efectos gráficos como el volteo de tiles/sprites (girar sobre el eje vertical u horizontal los sprites), escalado por hardware y, en el caso de las arquitecturas más potentes, diferentes grados de transparencia y alpha blending. Se trata de descargar lo máximo posible de trabajo gráfico a la CPU principal. El uso de tiles y sprites aporta las siguientes ventajas:

- Ahorro de memoria:
  Al componer la imagen a base de "trozos" **que se pueden repetir**, se necesita menos memoria para almacenar **sólo cada trozo "diferente"**. Visto de otro modo, con la misma cantidad de memoria podemos obtener unos gráficos más vistosos. La imagen en pantalla vendrá dada por un mapa de tiles en el que se indicará el tile exacto (el trozo gráfico) que corresponde a cada región de la pantalla. Ésta es la principal razón de que en los juegos de 8 y 16 bits los fondo de las pantallas sean repetitivos (especialmente acusado en los juegos de scroll lateral): Están construidos a base de repetir pequeños fragmentos. Cuanto mayor sea el numero de tiles presentes en pantalla, los tiles diferentes cargados en memoria y la calidad gráfica y disposición de estos últimos, menos se notará este efecto.

- Ahorro de ancho de banda:
  Al estar compuesta la imagen a base de tiles, se necesitan transmitir menos datos a la memoria de video pues muchos de los mismos se reutilizan. Los buses están ocupados menos tiempo y transmiten menos información. Es frecuente la inclusión de DMAs que liberan a la CPU de estas transferencias.

El hardware gráfico por tanto consiste en un motor más o menos potente de tiles/sprites sin capacidad 3D alguna.

El hardware de sonido o PSG (Programable sound generator) consiste en un chip que puede mezclar diferentes canales, cada uno con un tono variable en frecuencia y amplitud (volumen). En ocasiones puede haber algún canal FM que genere armónicos a partir de la combinación de múltiples ondas, como en un instrumento de música real. También es frecuente el uso de generadores de ruido, usados para efectos de sonido tales como explosiones o disparos. En general, la calidad de sonido obtenida por este tipo de chips es muy inferior a la obtenida por las tarjetas de sonido actuales pero perfectamente aceptables para la época.

3.3.2 Consolas de 32 bits a las actuales

En la actualidad, el hardware actual de la consolas tiende a converger cada vez más con el de los Pcs. Los juegos actuales explotan las capacidades 3D de las tarjetas gráficas que incorporan las consolas. Así, en la Xbox hay una tarjeta aceleradora 3D de Nvidia, en GameCube una de ATI, etc. Hoy en día es común el uso de lenguajes de alto nivel y tecnologías similares a las de PC en los desarrollos: C++, OpenGL, Direct3D, etc. Se usan APIs de programación y kits de desarrollo comunes a varias desarrolladoras (proporcionadas por el fabricante de la máquina) y se reutilizan motores 3D para múltiples juegos. Es frecuente la venta de licencias de uso de librerías y motores de una compañía a otra. En la reciente generación, las CPUs y tarjetas de video tienen una gran capacidad de cálculo, con múltiples cores y procesadores trabajando en paralelo.

Muchas de las consolas de la actual generación y alguna de las anteriores son retrocompatibles con las de anteriores generaciones mediante el uso del propio hardware de la consola original o mediante emuladores software. Evidentemente el uso del hardware original produce mejores resultados que cualquier emulador software, con una compatibilidad del 100%. El uso de emuladores software, si bien puede presentar una compatibilidad menor o ser más lento, hace más económica la consola y permite realizar efectos por software que el hardware de la consola que se emula no hacía, por ejemplo, antialiasing y filtros de texturas. Algunas consolas con hardware de anteriores generaciones son:

- Primeras remesas de PS3: Las primeras remesas de PS3 japonesas incluían el hardware de

PS2 y PSX, presentando una compatibilidad total con ellas.

- PS2: Incluye el hardware de PSX
- DS: Incluye el hardware de GBA (aunque no el de la propia Game Boy)
- Wii: Incluye el hardware de GameCube (mediante un divisor de frecuencia hace que la consola arranque en modo GC o Wii).

Algunas consolas que realizan emulación por software:

- Xbox360: Emulador para cada juego de Xbox (personalizado) y juegos de consolas antiguas para comprar en el bazar. Éstos juegos suelen estar mejorados respecto a los originales por lo que no se pueden considerar emuladores en si sino más bien nuevas versiones de los mismos.
- Wii: La consola virtual permite descargar juegos de anteriores generaciones con emuladores software embebidos en ellos.
- Remesas actuales de PS3: Realizan la emulación de PS2 y PSX mediante un emulador software embebido en el firmware de la consola. También permiten descargar juegos de anteriores generaciones (el firmware incorporar emuladores de varias plataformas).

3.4 El hardware de la Game Boy

3.4.1 Introducción

La Game Boy es una videoconsola excelente para introducirse en el desarrollo de emuladores. Sin dejar de ser un hardware simple, contiene todos los elementos presentes en otras arquitecturas más complejas de 8 y 16 bits. Es aspecto externo e interno de la Game Boy es el siguiente:

Ilustración 3.7: Exterior de una Game Boy Color

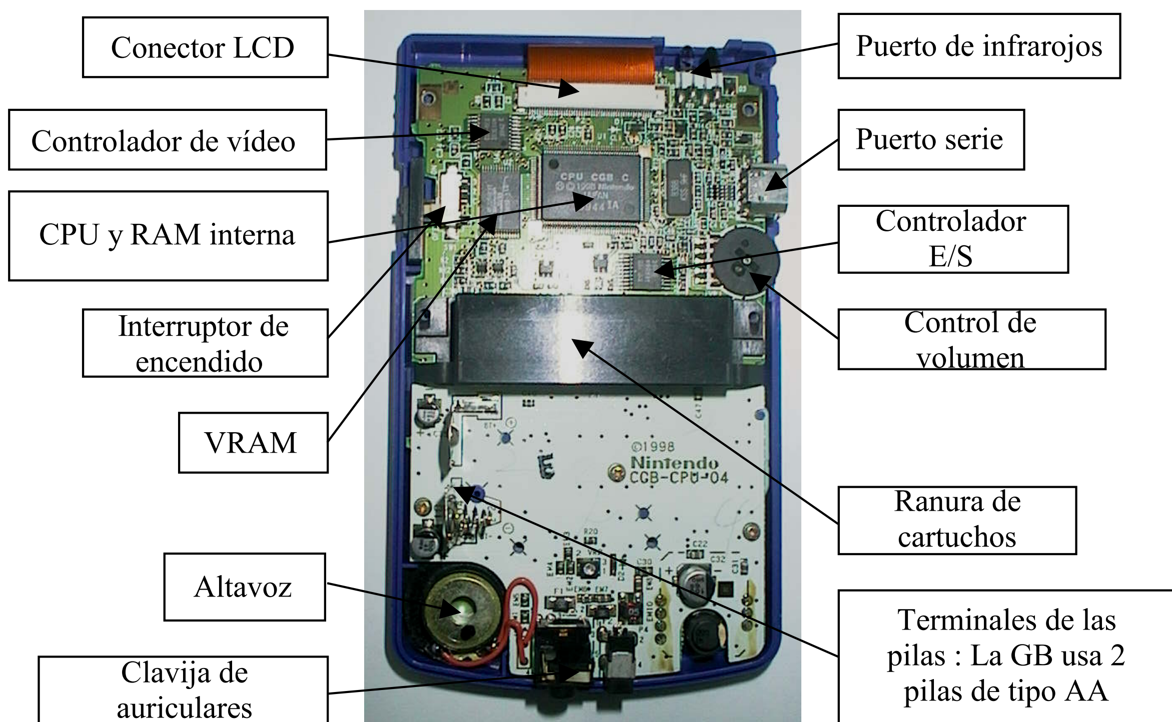


Ilustración 3.8: Interior de una Game Boy Color

De todos esos componentes, la parte de la CPU realmente interesante para la emulación es la siguiente:

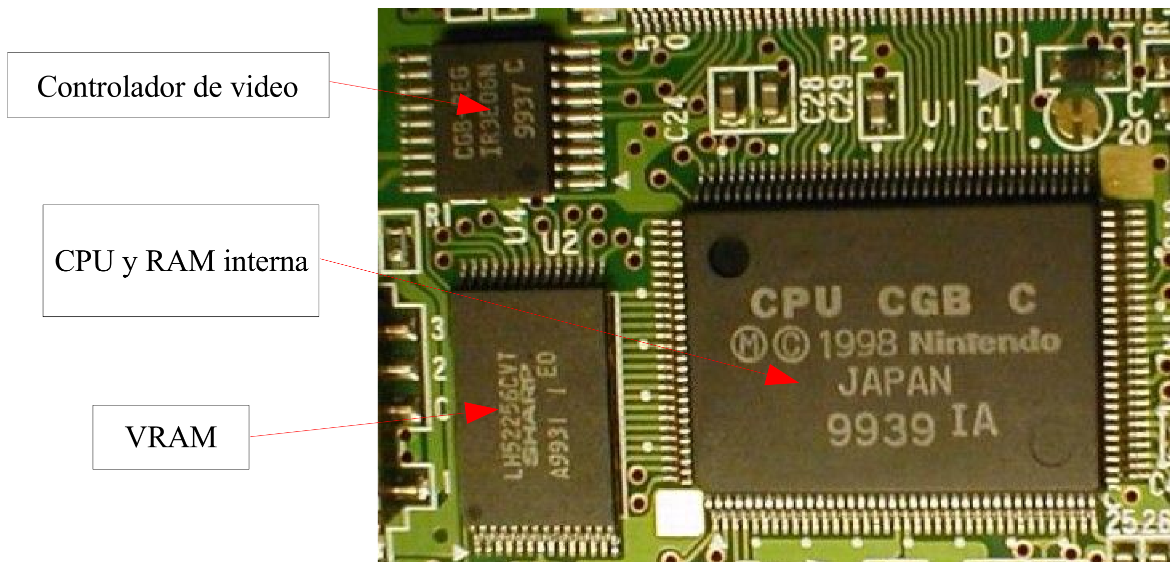


Ilustración 3.9: Detalle de la placa base de la Game Boy Color

- La CPU y memoria RAM integrada: El chip más grande de la fotografía contiene la CPU principal junto con la RAM de la videoconsola. Es común la inclusión en un mismo chip integrado de más de un componente por las siguientes razones:

    - Mejor comunicación: Cuanto más próximos físicamente se encuentren 2 componentes, más rápido se comunicaran entre si (o visto de otro modo, el ancho de banda entre ellos será mayor)

    - Abaratamiento de costes: La tecnología de integración avanza, se abarata y permite integrar más transistores por pulgada por lo que al fabricante le resulta más barato fabricar varios componentes integrados en un mismo encapsulado. En el caso de estos chips, de una velocidad ridícula para nuestros días (unos pocos megaherzios), el problema de disipación de calor es casi inexistente.

- El controlador de Video (LCD controller): Situado en la esquina superior izquierda, es el responsable del cálculo de la imagen que se va a mostrar por pantalla así como del propio pintado, liberando a la CPU de cualquier tarea de este tipo.

- La RAM de video: Situada en la esquina inferior derecha, es la responsable de almacenar la información gráfica usada por el controlador de video. Los datos escritos en o desde ésta memoria generalmente son realizados por técnicas de DMA para liberar a la CPU de éstas tareas.

Información detallada del hardware de la Game Boy y su funcionamiento puede consultarse

en [FFRK01].

3.4.2 Diferencias entre Game Boy y Game Boy Color

Las diferencias entre ambos sistemas no son meramente externas (el uso de pantalla a color) sino que también afecta al hardware interno. Éstas diferencias serán explicadas en los siguientes apartados en mayor profundidad. Brevemente los cambios son:

- La CPU de la Game Boy Color puede funcionar a dos velocidades: single (4.2 Mhz) o double (8.4 Mhz).
- El hardware de video fue mejorado. La consola posee el doble de memoria de Video, efectos gráficos de tiles (además de los de sprites ya presentes en Game Boy), un máximo de 56 colores en pantalla (de una paleta de 32768 colores) y 4 veces más memoria RAM interna.
- Dos modos nuevos de transferencia DMA.

3.4.3 CPU

La CPU de la Game Boy es un LR35902 (un Z80 modificado fabricado por Sharp) a 4.2 Mhz con el bus de datos de 8 bits y uno de direcciones de 16, lo que le permite acceder a 2^{16} = 65536 posiciones de memoria (bytes). Con la Game Boy Color se incorporó el modo doble velocidad (8.4 Mhz) al ya existente, por lo que la CPU podía cambiar entre dos velocidades durante su funcionamiento (consumiendo mayor batería en el modo doble por supuesto).

3.4.3.1 Diferencias y similitudes con el Z80

El LR35902 es una versión recortada (simplificada) del Z80, creada expresamente para la Game Boy y con los siguientes cambios:

- Menor número de instrucciones: Las instrucciones del tipo DD, ED y FD han sido eliminadas aunque las CB se mantienen. También han sido eliminadas las que usan los registros IX e IY, las de intercambio (hacen uso del doble juego de registros), las de E/S y los saltos condicionales, llamadas o retornos que usan el flag de paridad/desbordamiento o el de signo.

- Cambio de acceso a los puertos: Al haberse eliminado las instrucciones IN/OUT, los puertos están mapeados en memoria y se acceden con simples instrucciones LD. Es decir, la E/S ha cambiado a espacio de memoria (E/S mapeada en memoria).

- Menor número de registros: Los registro IX e IY junto con el segundo juego de registros completo han sido eliminados. La bibliografía no dice nada sobre los registros R e I presentes en el Z80 por lo es probable que la Game Boy carezca de ellos.

- Menor número de flags: El flag de signo y el de paridad/desbordamiento han sido eliminados.

- Ciclos de instrucciones simplificados: Todas las instrucciones duran un mínimo de 4 ciclos y siempre son duraciones múltiplos de 4.

Muchas instrucciones han desaparecido, se han añadido unas pocas nuevas y algunas se han remapeado a otro opcode diferente. Se puede encontrar un resumen más detallado del juego de instrucciones en el anexo B. Se puede encontrar una recopilación de documentos sobre el Z80 en la sección de descargas de [9].

3.4.3.2 Registros y Flags

Los registros del LR35902 se pueden resumir en la siguiente tabla:

16bits	*Hight*	*Low*	*Nombre*
AF	A	-	Acumulador/Flags
BC	B	C	
DE	D	E	Propósito General
HL	H	L	
SP	-	-	Stack Pointer
PC	-	-	Program Counter

NOTA: *Los registros señalados con guión "-" indican que no pueden ser direccionados (accedidos) por el programador, esto es, son transparentes a la programación*. A continuación se detallan los registros más importantes:

- Acumulador (A): Es el registro de 8 bits más usado en las operaciones aritmético-lógicas. Suele ser el registro destino de bastantes operaciones aritméticas y de carga aunque el resto de registros son usados también en menor medida con este fin.

- Propósito General (B,C,D,E,H,L): Usados como almacenamiento resultado/fuente operandos por algunas instrucciones. Estos registros de 8 bits pueden agruparse de 2 en 2 para ser direccionados como registros de 16 bits (BC,DE y HL).

- Stack Pointer (SP): Registro de 16 bits que apunta a posición de memoria de la cima de la pila. Usado por operaciones de tipo push/pop. En la Game Boy, la pila crece hacia abajo (al añadir un elemento, se decrementa SP y al eliminarlo se incrementa).

- Program Counter (PC): Registro de 16 bits que apunta a la posición de memoria de la siguiente instrucción. Normalmente incrementado de forma secuencial salvo en las instrucciones de salto, llamada (CALL), push y pop, retorno de subrutina (RET) y similares.

El registro F no es accesible directamente por el programador. Contiene los flags que afectan al comportamiento de algunas instrucciones como las de salto condicional, suma BCD, etc. Cada operación modifica de manera diferente los flags: algunas no los modifican, otras los ponen siempre a 1, a 0 o a un valor dependiente de la operación. Los flags existentes y su significado son:

Bit	*Nombre*	*Activado (Set)*	*Borrado (Clear)*	*Explicación*
7	zf	Z	NZ	Zero Flag
6	n	-	-	Add/Sub Flag (BCD)
5	h	-	-	Half Carry Flag (BCD)
4	cy	C	NC	Carry Flag
3-0	-	-	-	No usados (siempre a 0)

- Zero Flag: 1 si el resultado de la última operación fue 0. Usado en saltos condicionales.

- Carry Flag (C or Cy): 1 si el resultado produce overflow o underflow, esto es, el resultado de una suma es mayor que 0xFF(8 bits) o que 0xFFFF(16 bits), o resultado de una resta es menor que 0x0. En las operaciones de rotación/desplazamiento se considera un bit más al que se mueve el bit desplazado/rotado. En general es un noveno o diecisieteavo bit usado para detectar overflow/underflow. Es usado en saltos condicionales e instrucciones como ADC,SBC,RL,RLA,etc.

- Half Carry Flag (H): Similar al anterior, es usado para detectar si hubo acarreo (1) o no (0) en el bit *(n/2) - 1* del registro, siendo *n* el tamaño del registro (8 o 16)[3].

- Add/Sub Flag (N): 1 si la anterior operación fue una resta, 0 si fue un suma. Este flag y el anterior se usan para aplicar una corrección a la suma/resta de dos números BCD para que el

3 El término ingles *nibble* hace referencia a los conjuntos de bits [0,(n/2)-1] y [n/2,n-1] en un registro de *n* bits. Así, todo registro está formado por 2 *nibbles* (p.e. un registro de 8 bits estará formado por los *nibbles* [0,3] y [4,7]). El flag H denota precisamente si se ha producido un acarreo del bit más significativo del primer *nibble* (menos significativo) al bit menos significativo del segundo *nibble* (más significativo).

resultado siga siendo otro número BCD.

Para conocer como afecta exactamente cada instrucción a los flags del LR35902 se recomienda la consulta [FFRK01].

3.4.3.3 Juego de instrucciones

La CPU soporta 512 instrucciones. La estructura de una instrucción de GB es similar a la del Z80:

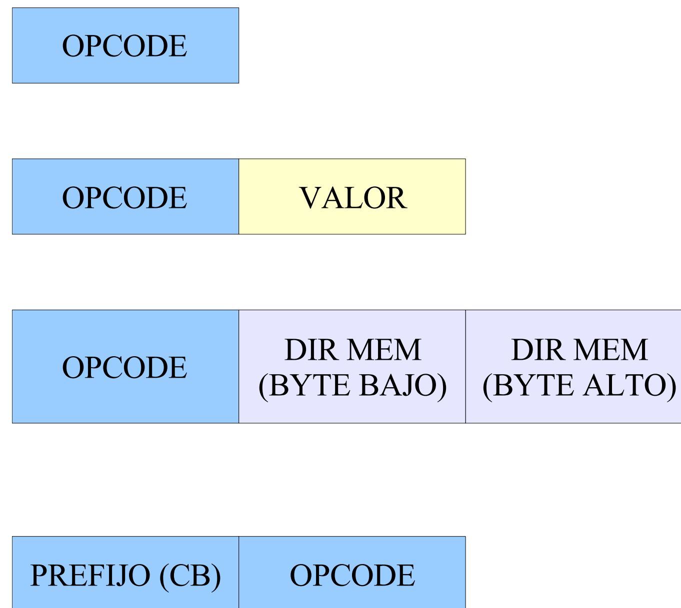


Ilustración 3.10: Formatos de instrucciones de Game Boy

Los 3 primeros tipos de instrucciones corresponden al juego de instrucciones original. Su opcode ocupa un byte y pueden tener 1 valor literal de un byte (con o sin signo) o una dirección de memoria (2 bytes). En total la longitud puede oscilar entre 1 y 3 bytes.

El 4º tipo de instrucción son las llamadas CB opcodes. Su opcode está compuesto por 2 bytes, siendo el primero de ellos siempre el valor hexadecimal 0xCB. Su longitud siempre es de 2 bytes.

Los modos de direccionamiento soportados son:

- Direccionamiento de código: Instrucciones de salto y llamadas
  - Absoluto: Dirección de 16 bits embebida en la instrucción
  - relativo a PC: offset de 8 bits embebido en la instrucción
  - indirecto a registro: Instrucción indica registro que apunta a posición de memoria con

offset de 8 bits.

- Direccionamiento de datos:
  - de registro: Algunos bits del opcode indican los registros origen y destino de la operación.
  - de bits: Algunos bits del opcode indican los bits de operando afectados por la operación.
  - inmediato: Operando de 8 bits embebido en la instrucción
  - indirecto a registro: Instrucción indica registro que apunta a posición de memoria con operando de 8 bits.

Varios de los direccionamientos de datos pueden darse simultáneamente en la misma instrucción. Para más información ver [FFRK01].

3.4.3.4 Interrupciones

Las interrupciones en la Game Boy son siempre enmascarables. Se pueden enmascarar individualmente reseteando bits en el registro IE o globalmente mediante el uso de las instrucciones EI y DI. Se producen automáticamente por el hardware aunque el programador también puede provocarlas manualmente habilitando en el registro de petición (IF) el bit correspondiente. Si se produce una petición de interrupción mientras ésta está deshabilitada, se quedará pendiente hasta que vuelva a habilitarse ese tipo de interrupción. Cuando se atiende una interrupción, se deshabilitan las demás (por defecto, no se permiten interrupciones anidadas). Ante múltiples peticiones de interrupción, se atiende primero la más prioritaria. Atender una interrupción consiste en saltar a cierta posición fija de memoria que contiene la rutina para atender a la misma. A continuación se muestran las posibles fuentes de peticiones de interrupción, por orden de mayor a menor prioridad:

- V-Blank: Se produce por cada refresco completo de la pantalla, ésto es, 60 veces por segundo.

- LCD Stat: Puede producirse por múltiples causas como son:

  - LYC = LY: El registro LY contiene la linea que se está pintando actualmente. Si su valor coincide con LYC y está habilitada la producción de interrupción LCD por esta causa en el registro STAT, se producirá una petición de interrupción LCD. El uso de esta causa para la producción de interrupción es frecuente para el uso de algunos efectos gráficos que requieren producirse antes de terminar de pintarse el frame. Por ejemplo, algunos juegos en color cambian los colores de las paletas en mitad del frame para permitir pintar más de 56 colores en una misma pantalla.

- OAM (modo 2): El pintado de casa linea en el LCD pasa por diferentes estados del mismo como después se verá. Esta causa hace que se produzca la petición de interrupción cuando el LCD entra en el modo 2 para cada linea.

- V-Blank (modo 1): Se produce la petición despues de pintar cada frame, como la propia interrupción V-Blank. Si ambas interrupciones están habilitadas se atenderá primero la interrupción V-Blank (más prioritaria) y despues la interrupción LCD Stat por causa V-Blank. Ambas son interrupciones diferentes con direcciones de la rutina de salto para atenderlas diferentes.

- H-Blank (modo 0): Idem que la causa OAM pero para el modo 1.

- Timer: La GB posee un timer que funciona a 4 posibles frecuencias. Cuando éste se desborda se produce una petición de interrupción.

- Puerto serie: La GB posee un puerto serie con un protocolo similar al RS-232 pero en crudo, ésto es, sin bits de start o stop. Cuando se completa una transferencia de un byte se produce la petición de interrupción.

- Joypad: Se produce cuando se presiona una tecla. Ésta interrupción no parece funcionar en la GBC o hardware posteriores pues es poco útil para los programadores debido a la imposibilidad de distinguir una pulsación de tecla de una pulsación del PAD de direcciones. La GB sólo tiene 4 bits para el estado del Joypad por lo que solo se puede recuperar el estado de los 4 botones o las 4 direcciones por cada lectura. De intentar recuperar ambos (o intentar que se produzcan interrupciones por ambos), no se podría distinguir si la pulsación ha sido de un botón o una dirección. Éste diseño obliga a los programadores a implementar la lectura del estado del Joypad por *polling* en 2 fases (para leer estado de botones y estado de PAD) con una espera previa a ambas lecturas para evitar el efecto de los rebotes.

3.4.4 Memoria

La Game Boy maneja varios tipos de memoria. En principio como tiene un bus de direcciones de 16 bits, la cantidad de memoria accesible por la misma debería ser de $2^{16} = 65536$ posiciones de memoria (bytes). Al ser una arquitectura de E/S mapeada en memoria, ciertas posiciones de memoria en realidad son registros internos del hardware o, dicho de otro modo, no existen instrucciones especificas de E/S (in/out) sino que se accede a la memoria de los dispositivos con instrucciones load/store ya que en realidad son posiciones de memoria principal.

Aunque 64KB direccionables puedan parecer pocos a priori, mediante la técnica del

intercambio de bancos (bankswitching) que posteriormente se verá, la GB puede acceder a más memoria de la inicialmente prevista.

3.4.4.1 Manejo de memoria (bankswitching)

La técnica de bankswitching es común en las videoconsolas de 8 y 16 bits. En éstos sistemas el espacio de direcciones está limitado, debido principalmente al coste en aquella época de la memoria y al tamaño de los buses de direcciones. Por supuesto, éstos sistemas carecen de una unidad MMU o de cualquier tipo de sistema operativo. Un pequeño programa almacenado en una ROM interna a la máquina era el responsable de inicializar la misma, dejarla en un estado inicial mínimo para empezar a usarse y cederle el control al juego que tenía el control total y absoluto de la máquina a partir de entonces.

Para suplir la carencia del limitado espacio de direcciones accesible se usan los MBC (Memory Bank Controller), un hardware que mediante la técnica de bankswitching (intercambio de bancos) permite mapear en una misma región de espacio de direcciones varios bancos de memoria, de manera que éstos son intercambiables y sólo uno de ellos, el que está mapeado, es accesible cada vez. Los MBC se consideran los precursores de las MMU de las consolas actuales, mucho más complejas y con auténticos sistemas operativos embebidos

Los cartuchos de GB y GBC incorporan un MBC para poder mapear diferentes trozos de la ROM o la RAM del cartucho cada vez. La GBC incorpora además un MBC interno a la consola para permitir bankswitching

3.4.4.2 Organización, tipos y uso de memoria

El mapa resumen de la memoria accesible por la Game Boy es el siguiente:

Bytes	Game Boy	Bytes	Game Boy Color
0000-3FFF	16KB ROM banco 0	0000-3FFF	16KB ROM banco 0
4000-7FFF	16KB ROM bancos 1-n	4000-7FFF	16KB ROM bancos 1-n
8000-9FFF	8KB RAM de Vídeo (VRAM)	8000-9FFF	8KB VRAM banco 0-1 en modo GBC
A000-BFFF	8KB RAM externa o de cartucho(puede tener bancos o no existir)	A000-BFFF	8KB RAM externa o de cartucho(puede tener bancos o no existir)
C000-DFFF	8KB RAM interna o de trabajo	C000-CFFF	4KB RAM interna o de trabajo banco 0
		D000-DFFF	4KB RAM interna o de trabajo bancos 1-7 en modo GBC
E000-FDFF	Echo RAM (mismos valores que C000-DDFF)	E000-FDFF	Echo RAM (mismos valores que C000-DDFF)
FE00-FE9F	Sprite Attribute Table (OAM)	FE00-FE9F	Sprite Attribute Table (OAM)
FFA0-FEFF	No usable	FFA0-FEFF	No usable
FFOO-FF7F	Puertos (registros) de E/S	FFOO-FF7F	Puertos (registros) de E/S
FF80-FFFE	RAM interna (Hight RAM)	FF80-FFFE	RAM interna (Hight RAM)
FFFF	Interrupt Enable Register	FFFF	Interrupt Enable Register

Tabla 3.9: Mapa de memoria en la Game Boy y Game Boy Color
(ver tabla a color A.1)

Las zonas anaranjadas pueden manejar más memoria mediante bankswitching. La zona roja no es usable por el programador (ver tabla a color). A continuación se detalla cada zona:

- ROM del cartucho: Es accesible en la región 0000-7FFF (32KB), destinándose siempre los primeros 16KB al banco 0 y los 16 siguientes siendo mapeables a cualquier otro banco. Las lecturas en esta zona de memoria leen posiciones de memoria de los bancos mientras que las escrituras modifican el estado del MBC del cartucho (de existir éste) permitiendo, entre otros, el mapeo de bancos de ROM y RAM de cartucho. En el apartado de los cartuchos se describirá brévemente los tipos de MBC y sus regiones de escritura.

- RAM de Vídeo (VRAM): Accesible en las región 8000-9FFF. La GB posee 8KB de memoria de vídeo mientras que la GBC tiene 2 bancos de 16KB mapeables por MBC interno de la consola. Ésta memoria no es accesible en todo momento, depende del modo del LCD en el que éste la máquina. Será vista en mayor detalle en el apartado correspondiente del Hardware de Video.

- RAM externa: Algunos cartuchos poseen RAM para guardar el estado de la partida entre otros. De ser así, ésta es accesible en la región A000-BFFF. Sólo se permiten 8KB se memoria RAM externa accesibles por vez, ésto es, el tamaño del banco de memoria externa

es 8KB. Si el cartucho posee una cantidad de RAM superior, necesitará de un MBC (el mismo que se usa para mapear los bancos de ROM). Su acceso puede habilitarse o deshabilitarse mediante escrituras en el MBC.

- RAM interna: Accesible en la región C000-DFFF. La GB posee 8KB de memoria RAM en la propia consola por lo que no necesita ningún tipo de mapeado ya que toda su memoria RAM es accesible en esa región Sin embargo la GBC posee 32KB en 8 bancos de 4KB cada uno, estando mapeado el banco 0 a C000-CFFF y el resto siendo mapeables en D000-DFFF. La RAM interna es usada como cualquier otra RAM: Para almacenar datos intermedios, p.e la pila del sistema, algunos gráficos y cálculos, etc.

- OAM: Accesible en FE00-FE9F, almacena información sobre los sprites. Ésta memoria no es accesible en todo momento, depende del modo del LCD en el que éste la máquina. Será vista en mayor detalle en el apartado correspondiente del Hardware de Video.

- Puertos: Los puertos de la GB y GBC son accesibles en FFOO-FF7F y FFFF. Su escritura modifica el comportamiento de algunos parámetros del hardware. Sólo se han incluido en este documento los más importantes.

- Otras zonas:

  - Echo RAM: Accesible en E000-FDFF, cualquier lectura o escritura en la misma es equivalente a hacerlo en C000-DDFF, ésto es, ambas zonas de memoria son idénticas (o visto de otro modo, son la misma accesible desde dos regiones diferentes). Éste tipo de zonas son comunes en las consolas de 8 y 16 bits. Típicamente no son usadas por los programadores y el fabricante no recomienda usarlas.
  - Zona no usable: La región FFA0-FEFF no se puede usar para leer o escribir. No obstante, la documentación no especifica qué ocurre en la GB si se intenta acceder a alguna de éstas posiciones. Éste tipo de zonas muertas también son comunes en las consolas de 8 y 16 bits.

  - Hight RAM: Accesible en FF80-FFFE, es la única zona accesible durante una transmisión Sprite DMA por lo que el código que la inicie ha de encontrarse forzosamente en esa zona de memoria. Algunos opcodes están optimizados (son más rápidos) para esta zona de memoria por lo que es usada frecuentemente para almacenar datos.

3.4.4.3 DMA

La GB dispone de un modo de DMA (Sprite DMA) capaz de hacer transferencias de 160 bytes desde memoria ROM o RAM a memoria OAM, esto es, desde una posición en 0000-F100 copiar A0 bytes consecutivos a FE00. Como la Memoria OAM ocupa exactamente 160 bytes, una transferencia Sprite DMA sobrescribirá completamente toda la memoria OAM, cambiando los atributos de los sprites. Como ya se ha comentado, durante la transmisión Sprite DMA, toda la memoria es inaccesible a excepción de Hight RAM (FF80-FFFE). Es común entre los programadores iniciar la transferencia en la interrupción V-Blank, cambiando así los sprites para el próximo refresco. También es común su uso en la interrupción LCD Stat con la condición LY = LYC, para permitir por ejemplo el doble de sprites en pantalla (40 en la parte superior y 40 en la inferior).

La GBC dispone, además del modo anterior, de otros 2 modos denominados VRAM DMA: General Purpose DMA (GDMA) y H-Blank DMA (HDMA). Éstos nuevos modos son necesarios debidos a la mayor velocidad de la GBC y al uso de color y varios bancos en VRAM. Ambos modos pueden hacer transferencias de entre 16 y 2048 bytes en total desde memoria ROM o RAM a memoria VRAM. Concretamente pueden copiar datos desde 0000-7FF0 o A000-DFF0 hasta 8000-9FF0. La diferencia entre ambos es:

- GDMA: transmite todos los bytes en una sola vez, deteniendo la ejecución del programa hasta que la transferencia ha terminado. No puede ser interrumpido y aunque la CPU está parada durante la transmisión, la velocidad a la que los datos son copiados es muy superior a la de la CPU haciendo la misma tarea, por lo que al estar menos tiempo parada tiene sentido.

- HDMA: Una vez activado, transmite bloques de 16 bytes en cada interrupción H-Blank. La ejecución del programa se detiene en cada transferencia pero continua entre transferencias. Puede ser interrumpida.

3.4.4.4 Algunos puertos de la Game Boy

A continuación se describen algunos puertos de la GB y GBC. La descripción completa de los mismos se puede encontrar en [FFRK01], aquí solo se explicarán algunos brevemente:

- Estado del Joypad: P1/YOYP (FF00). Sólo puede consultarse el estado de los botones o el PAD cada vez (sólo se aprovechan 4 bits para el estado). La GB consulta el estado del Joypad mediante dos consultas de polling consecutivas. También es usado para enviar paquetes de comandos a la SNES mediante el SuperGameBoy, un accesorio que permite jugar a los juegos de GB en la SNES.

- Comunicaciones: SB (FF01) y SC (FF02). Los registros Serial Transfer Data y Serial Transfer Control son usados para la comunicación por puerto serie entre dos GB (cable gamelink). La GBC permite además la comunicación mediante el puerto infrarrojo con otra GBC mediante el puerto RP (FF56).

- Timers (temporizadores): DIV (FF04) TIMA (FF05) TMA (FF06) y TAC(FF07). Los dos primeros, Divider Register y Timer Register son 2 temporizadores, el segundo de ellos de frecuencia programable. Cuando TIMA desborda, se recarga con el valor de TMA (Timer Modulo) y se produce una petición de interrupción. TAC (Timer Control) permite variar la frecuencia de TIMA, así como activarlo o desactivarlo.

- Gestión de interrupciones: IF (FF0F), IE (FFFF) e IME. El primero, Interrupt Flag, es usado para las peticiones individuales de interrupción ; el segundo, Interrup Enable, para habilitar o deshabilitar las interrupciones individualmente y el tercero, Interrupt Master Enable para habilitar o deshabilitar todas las interrupciones. Para que una interrupción se produzca ha de estar habilitado IME así como los bits correspondientes en IE e IF. IME no es accesible por el programador, su valor se fija mediante las instrucciones EI (Enable Interrupts) , DI (Disable Interrupts y RETI (Return From Interrupt).

- Relativos al LCD: LCDC (FF40), STAT (FF41), SCY (FF42), SCX (FF43), LY (FF44), LYC (FF45), WY(FF4A), WX (FF4B). El primero, LCD Control, controla el visualizado de las capas, sus prioridades, el origen de los datos de los sprites y sus mapas así como el tamaño de los mismos. El segundo, LCD Status, indica el modo actual del LCD y permite habilitar/deshabilitar las posibles causas de interrupción LCD. SCY (Scroll Y), SCX (Scroll X), WY(Window Y) y WX(Window X) especifican las posiciones del la capa background y la capa window, ésta última relativa a background. LY(LCDC Y-Coordinade) contiene la linea actualmente dibujada por el LCD y LYC (LY Compare) la linea a comparar para la posible interrupción LCD (en caso de que esté habilitada la interrupción y ésa condición). Ésto se verá en mayor profundidad en el apartado correspondiente del Hardware de Video.

- Relativos al DMA: DMA (FF46), HDMA1 (FF51), HDMA2 (FF52), HDMA3 (FF53), HDMA4 (FF54), HDMA5 (FF55). El primero se usa para iniciar una transferencias del Sprite DMA, poniendo los dos bytes más significativos de la dirección origen. Para transmisiones VRAM DMA, HDMA1 y 2 contienen la dirección origen y HDMA3 y 4 la destino, mientras que HDMA5 se usa para indicar el modo (GDMA o HDMA), la longitud e iniciar/parar la transmisión.

- Relativos a las paletas: BGP (FF47), OBP0 (FF48), OBP1 (FF49), BGPI (FF68), BGPD (FF69), OBPI (FF6A), OBPD (FF6B). Los 3 primeros se usan únicamente en modo DMG (juegos sin color) mientras que los 4 segundos son exclusivos de la GBC para juegos en color. Su uso se explicará en el apartado correspondiente del Hardware de Video.

- Otros registros: Existen otros registros relativos a GBC además de los ya citados con anterioridad como pueden ser KEY1 (FF4D) usado para cambiar el modo de velocidad de la CPU, o SVBK (FF70) que permite cambiar entre los bancos de WRAM en D000-DFFF. También están los relativos al manejo del PSG de la GB que son los comprendidos entre las regiones FF10-FF3F. Su uso va desde encender o apagar por completo el sonido, a cambiar la frecuencia o volumen de cada canal. Ya que la emulación del sonido queda descartada por imposibilidades de la API, su descripción queda fuera del alcance de éste proyecto.

3.4.5 Hardware de Video

Ya se expuso con anterioridad en el capítulo "el Hardware de una videoconsola – Consolas de 8 y 16 bits" el funcionamiento común de éstos sistemas. Varias capas dibujadas unas encima de otras, compuestas cada una de ellas por diferentes teselas a las cuales denominamos tiles. Los sprites básicamente son tiles dibujados en la última capa y con posición propia. A continuación se explicará en mayor profundidad el hardware de video de la GB, sus diferentes capas, registros y efectos involucrados en el funcionamiento del mismo. Para facilitar la comprensión del funcionamiento, se han simplificado algunos detalles:

- El cálculo de prioridades de pintado de tiles y sprites: Influyen múltiples prioridades en diferentes registros que no vienen al caso para comprender el concepto.

- El segundo área de datos (8800-97FF): La numeración de tiles es de [-128-127] en lugar de [0-255].

- Algunos registros y propiedades almacenan los valores con un desplazamiento. Así, WX y las posiciones X e Y de sprites almacenan los valores con un desplazamiento positivo.

Estos detalles sólo son importantes para una correcta emulación por lo que se pueden observar en el código fuente del emulador. También vienen descritos en la descripción del hardware de video en [FFRK01].

3.4.5.1 El LCD

La GB dispone de una pantalla de 160x144 pixels de resolución. Se trata de un LCD capaz de presentar 4 tonos (verdes o grises dependiendo del modelo) mientras que en la GBC puede mostrar un total de 56 colores diferentes en pantalla de 32768 posibles (15 bits de profundidad).

Un LCD realmente no utiliza tecnología de rayos catódicos por lo que la pantalla no se pinta linea a linea ni es necesario tiempos de retorno del haz de electrones (V-Blank y H-Blank). El uso de estos intervalos se ha mantenido en todas las videoconsolas porque son útiles desde el punto de vista de la programación al poder aprovechar unas rutinas periódicas cada frame para realizar cálculos con los gráficos, p.e cambios de paletas y efectos (degradado, fadein, fadeout), sprites y efectos gráficos (rotaciones, etc). Además estos tiempos permiten escribir en VRAM ya que no está siendo accedida por el controlador LCD.

La linea que está siendo actualmente calculada por el hardware de video se encuentra en el registro LY. Cada linea pasa por 3 modos diferentes antes de calcularse que son en éste orden:

- Modo 2: El LCD está leyendo de la memoria OAM por lo que la CPU no puede acceder a ésta.
- Modo 3: El LCD está leyendo de las memorias OAM y VRAM por lo que la CPU no puede acceder a ambas. En modo GBC tampoco puede acceder a los datos de las paletas (registros BGPD y OBPD).
- Modo 0: El LCD está en el intervalo H-Blank por lo que toda la memoria es accesible por la CPU.

El modo 1 es el intervalo V-Blank en el que toda la memoria es accesible. En un ciclo completo (en un frame) el LCD pasa por los modos 2, 3 y 0 para cada linea, es decir, 144 veces consecutivas. Tras ésto, el tiempo de 10 líneas ficticias lo pasa en modo 1. Ésto es así porque aunque la GB tiene 144 líneas visibles, existen otras 10 más invisibles que son las que hipotéticamente se recorren en el tiempo que dura el V-Blank.

El modo actual del LCD se puede consultar en el registro STAT así como activar/desactivar las posibles causas de petición de la interrupción LCD Stat.

3.4.5.2 Las capas y sus prioridades

El hardware dispone de 3 capas, 2 de las cuales contienen tiles y la última sprites. A continuación, se detalla cada una de ellas:

- capa background: Es la menos prioritaria de las 3 (la primera que se dibuja). Como su propio nombre indica, generalmente se usa para dibujar los fondos de los videojuegos que suelen tener un contenido estático. Aunque la GB puede mostrar 160x144 pixel en pantalla, el

tamaño real de la capa background es de 256x256 pixels, lo que le posibilita mediante el uso de los registros SCX y SCY realizar efectos de scroll lateral. SCX y SCY contienen la posición de la esquina superior izquierda de la pantalla, es decir, la esquina superior izquierda de los 160x144 pixels visibles. Está capa tiene un efecto envolvente (wrap) de manera que si parte de ese área visible cae fuera de los 256x256 pixels, continúa por el lado opuesto tal y como muestra la siguiente figura:

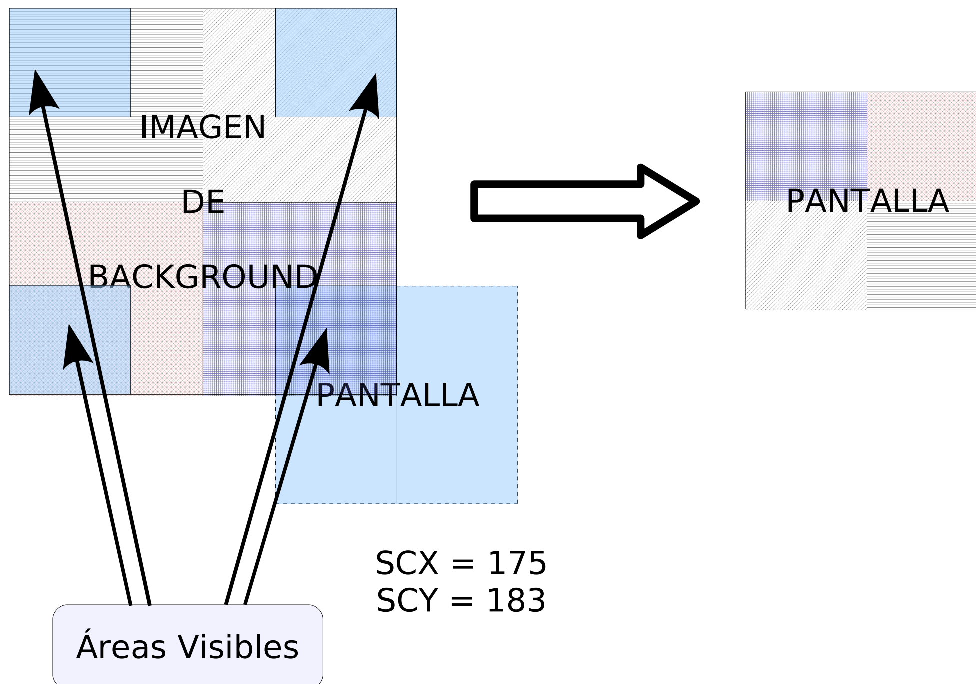


Ilustración 3.11: Efecto wrap en capa Background (ver ilustración a color A.2)

- capa window: Es pintada despues de background y antes de sprite. Es usada comúnmente, entre otros, para pintar la zona de estado del juego (la barra de vida, tiempo disponible, puntuación, número de vidas y demás parafernalia que suelen ocupar varias lineas consecutivas en el LCD). Los registros WY y WX contienen la posición relativa a background de la esquina superior izquierda de ésta capa.

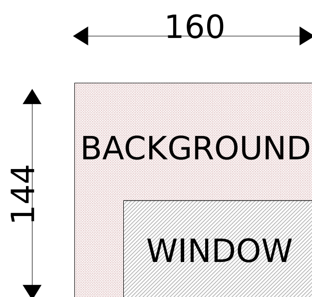


Ilustración 3.12: Capa Window sobre capa Background

(ver ilustración a color A.3)

- capa de sprites: Es la capa superior (se pinta sobre las otras dos). Contiene cualquier tile que requiera animación, ésto es, sea un sprite. Los sprites tienen, entre otras propiedades, sus propias coordenadas relativas a background. El personaje y los enemigos de un videojuegos está formados por múltiples sprites y son el ejemplo típico: su posición en pantalla varía independientemente del posible scroll lateral.
  Los sprites también suelen usarse para pintar partes del fondo de la pantalla que ocultan al personaje principal (por ejemplo, una columna) en los sistemas que no implementan prioridad entre tiles, p.e Nintendo DS.

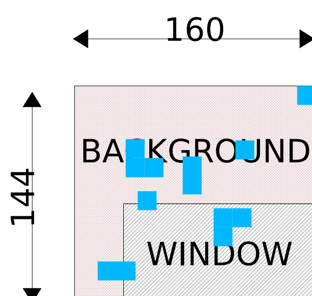


Ilustración 3.13: Capa Sprite sobre Window y Background

(ver ilustración a color A.4)

Se permite habilitar/deshabilitar el pintado de cada capa (BG,WIN y SPRITE) en el registro LCDC. Cabe destacar que la GB implementa prioridad por tile, ésto es, independientemente de las prioridades de las 3 capas, un tile puede tener prioridad sobre cualquier capa (se pinta encima de cualquier capa) y un sprite puede perder su prioridad (no se pinta). En el apartado de tiles y sprites se verá más en profundidad su funcionamiento.

Por último se muestran unas capturas reales del tratamiento de capas en GINAGE para el juego Aladdin de GB:

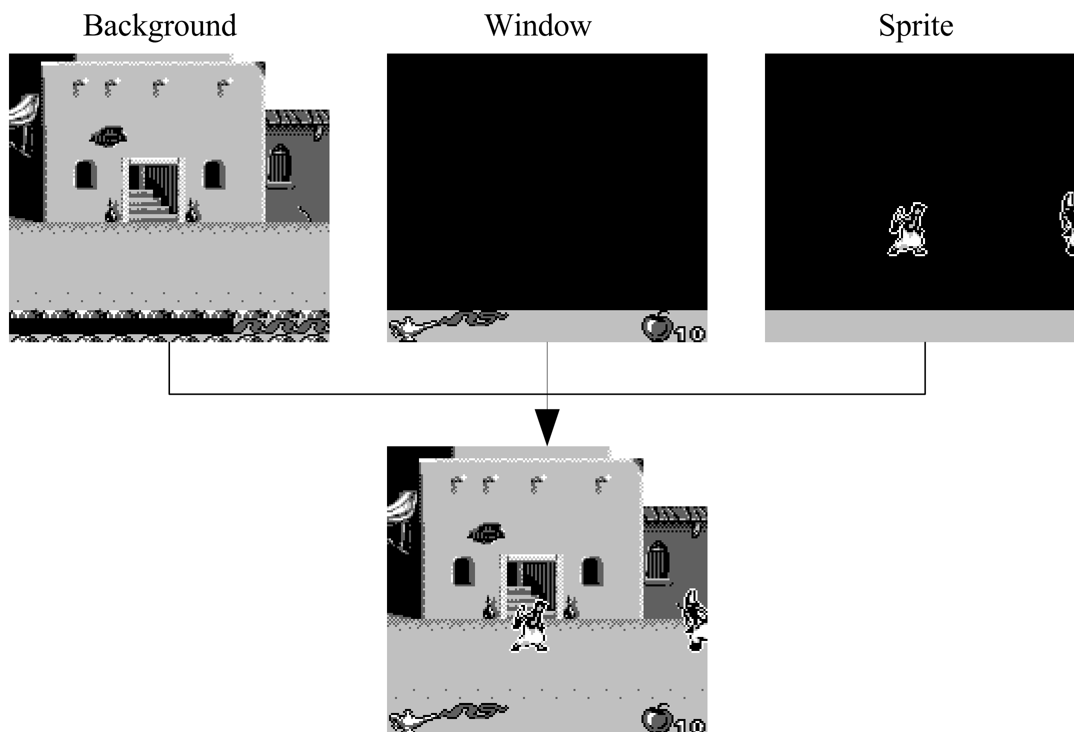


Ilustración 3.14: Combinación de las 3 capas para formar la imagen en pantalla

3.4.5.3 Las paletas

La GB original tiene 3 paletas con 4 colores cada una con una profundidad de 2 bits por color. Esto quiere decir que la GB sólo puede mostrar 4 colores diferentes que se traducirán, dependiendo del tipo de LCD, a 4 tonos verdes o grisáceos Una paleta se usa para los tiles, esto es, capas BG y WIN y las otras 2 para sprites de la capa SPRITE. Los registros BGP, OBP0 y OBP1 contienen los 4 colores de las 3 paletas, de manera que cada 2 bits conforman un color correspondiendo el valor 0 al color blanco (o verde más claro) y el 3 el negro (o verde más oscuro). El color 0 en las paletas de sprite (OBP0 y 1) es transparente, esto es, el color 0 en los sprites no se pinta.

Bits	*0-1*	*2-3*	*4-5*	*6-7*
	Color 0	Color 1	Color 2	Color 2

Tabla 3.10: Contenido de una paleta de Game Boy

Podría parecer que el uso de las paletas es inútil o cuanto menos extraño. Sin embargo su uso está justificado por el mero hecho de, al contener los datos gráficos de tiles y sprites un número de color para una paleta (con 2 bits) en lugar de el propio color (también con 2 bits), es mucho más eficiente y sencillo los cambios de color de tiles y sprites. Basta con cambiar el color de una/varias paletas para que en el siguiente refresco de pantalla se vea reflejado en todos los tiles o sprites

afectados. Además algunos efectos de paletas, p.e fade o color inverso, son fáciles de hacer (restando o sumando 1 a cada color en cada refresco o intercambiando las posiciones de los colores respectivamente).

La GB color en lugar de las anteriores paletas y sus correspondientes registros, dispone de 16 paletas con 4 colores cada una con una profundidad de 15 bits por color. Se usan 8 paletas para tiles y 8 para sprites. De nuevo, el color 0 en las paletas de sprite es transparente por lo que el número diferentes en pantalla queda limitado a 56 (8x4 + 8x3). No obstante es posible superar ésta limitación mediante el uso de la interrupción LDC Stat, cambiando los colores de las paletas en cada H-Blank si se quiere.

Los registros indice de paletas, BGPI y OBPI, apuntan al byte de la paleta correspondiente actualmente accedido, bien sea para leerlo o modificarlo. Al tener 8 paletas con 4 colores cada una y 2 bytes por color (15 bits + 1 desaprovechado), obtenemos que pueden almacenar valores en el rango 0-63:

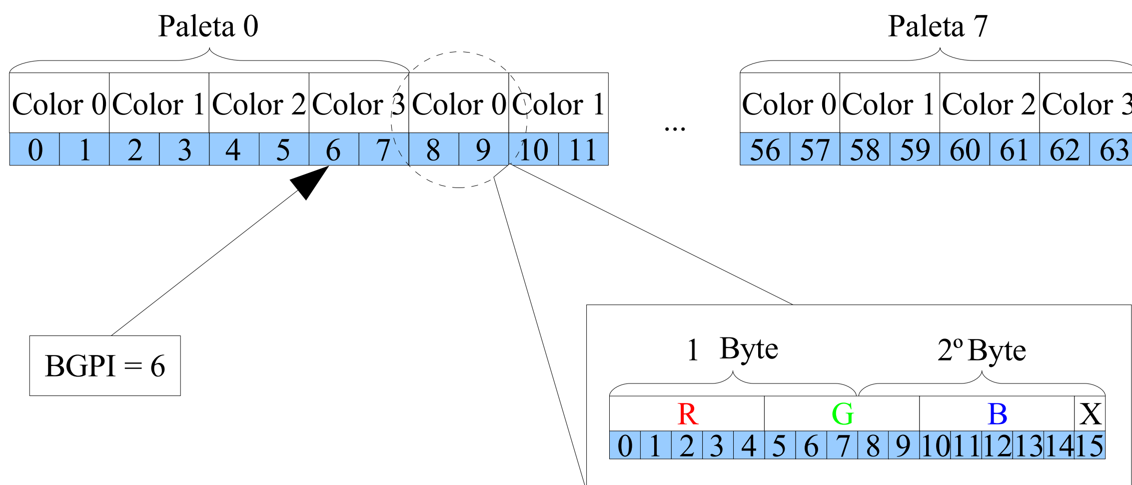


Ilustración 3.15: Organización en memoria de las paletas de Game Boy Color

Los registros de datos de paletas, BGPD y OBPD, son los que permiten leer o escribir el byte apuntado por los registros anteriores. Como las paletas en color son inaccesibles en el modo 3 del LCD, no se puede leer o escribir en estos registros durante ese tiempo. Los colores están compuestos por 2 bytes por lo que modificar un color implicar varias escrituras a los registros índice y de datos correspondientes.

3.4.5.4 Datos y propiedades de los tiles y sprites

Ya se ha visto el concepto de tile y sprite, las paletas que usan y su colocación en diversas capas. A continuación se verá dónde y cómo se almacena la información de los tiles y sprites y las propiedades de los mismos.

Toda la información de datos y propiedades de tiles y sprites se almacena en la VRAM

(8000-9FFF) y en la memoria OAM (FE00-FE9F). Se distinguen varios tipos de información de los mismos:

- Datos de tiles y sprites (8000-97FF): Esto es, la información gráfica (bitmaps) de cada tile. Un sprite no deja de ser un tile con algunas propiedades extras por lo que su información gráfica también se almacena aquí. Como ya se ha dicho, la información de los pixels almacenada hace referencia a un número de color dentro de una paleta y no al propio color. Por lo tanto, los datos de tiles consistirán en los números de colores para cada pixel del mismo, asociados a la paleta del tile. La paleta del tile viene determinada por dos factores:

    - la capa a la que pertenece éste: GB y WIN o SPRITE. En juegos DMG solo hay una paleta para las capas GB y WIN. Para el resto de casos hay múltiples paletas posibles y es donde entra en juego el segundo factor.

    - El atributo número de paleta: Como más adelante se verá, los tiles y sprites tienen atributos entre los que figura un numero de paleta (0-7 para juegos GBC o 0-1 para sprites de juegos DMG).

Un tile tiene un tamaño de 8x8 pixels y cada pixel necesita 2 bits para almacenar un número de color de los 4 posibles para su paleta por lo que se puede determinar que la información gráfica de un tile ocupa: 8x8x2 = 128 bits (16 bytes). La zona de memoria asignada a los datos es 8000-97FF, es decir, de 6144 bytes por lo que puede almacenar 6144/16 = 384 datos de tiles distintos. Ahora bien, la GB divide esta zona de memoria en 2 de 256 tiles que se solapan (8000-8FFF y 8800-97FF): Las capas BG y WIN pueden tener cualquiera de las 2 pero ambas comparten la elegida. La capa SPRITE siempre obtiene los datos de la primera. La conclusión es que GB y WIN comparten uno de los dos juegos de 256 tiles y SPRITE siempre los toma del primero.

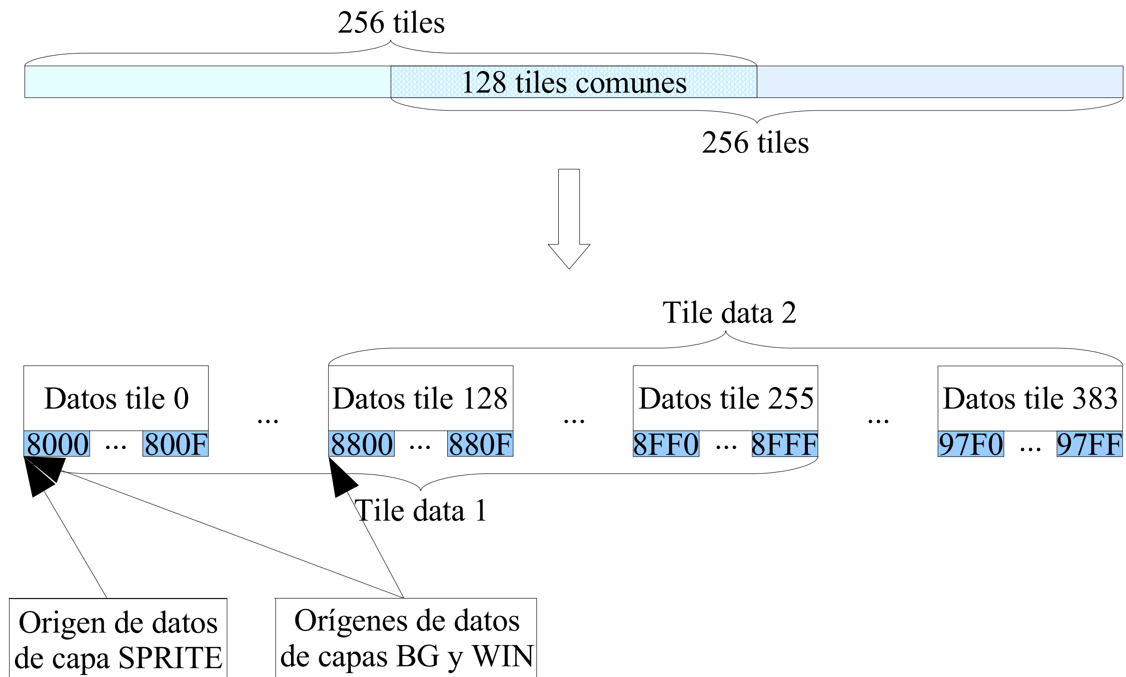


Ilustración 3.16: Organización de los dos juegos de datos de 256 tiles cada uno.

Además los 2 bits de cada color no se almacenan de manera consecutiva como cabría esperar en un principio sino que, para un pixel en la fila “y” y posición “x” dentro de un tile dado, el bit menos significativo del número de color se almacena en el byte y bit x, mientras que el bit más significativo del número de color se almacena en el byte y+1 y el bit x.

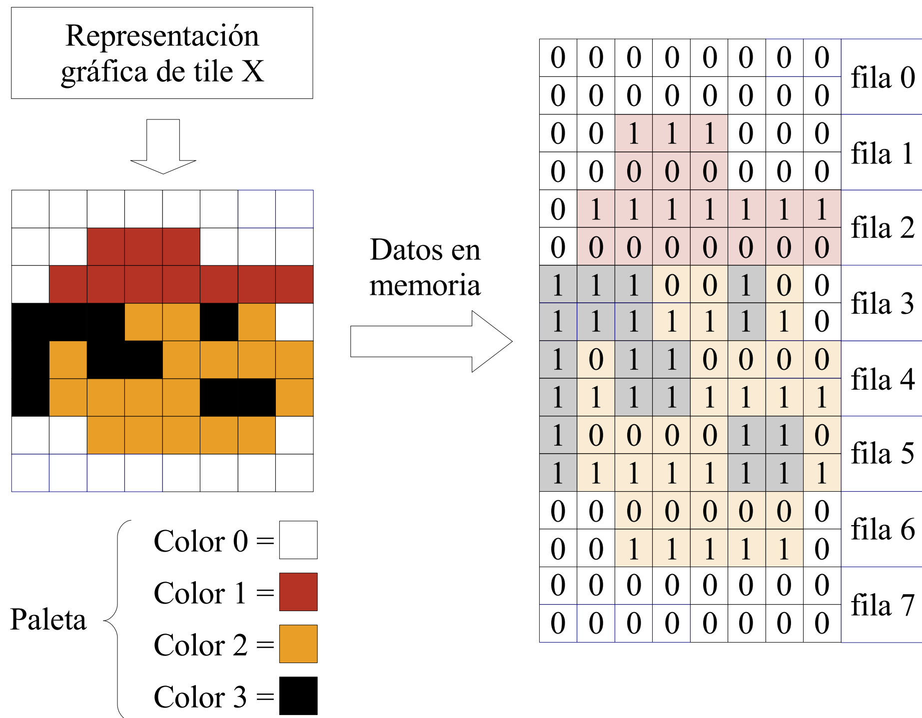


Ilustración 3.17: Representación de tile en memoria.

El color de cada pixel es codificado con 2 bits en 2 bytes consecutivos en memoria (ver Ilustración a color A.5)

La GBC posee dos bancos de VRAM por lo que tiene otro banco con otras 2 zonas de 256 tiles que se solapan en las mismas posiciones: 8000-8FFF y 8800-97FF. En este caso existe una propiedad de los tiles que indica de cual de los dos bancos proceden los datos del mismo. Gracias a ésto, la GBC puede acceder al doble de datos de tiles para las capas BG y WIN: 512 tiles diferentes (256 datos de tiles x 2 bancos selecciona bles).

- Mapas de tiles (9800-9FFF): Ya se vió que la ventaja principal de los tiles es que pueden repetirse en la pantalla ahorrando espacio en memoria y tiempo en transferencias. Las capas de BG y WIN tienen cada una un tamaño real de 256x256 pixels, esto es, 32x32 tiles de 8x8 pixels. Determinar qué hueco del puzzle de 32x32 tiles corresponde con qué datos de tile es la tarea de los mapas de tiles. La GB posee 2 de ellos en los rangos de direcciones 9800-9BFF y 9C00-9FFF y cada capa de tiles, BG y WIN, puede usar cualquiera de ellos, esto es, ambos pueden usar el mismo mapa o diferentes. Como las áreas de datos de tiles comprenden 256 tiles cada una, está claro que cada byte de cada mapa de tiles hace referencia a uno de éstos tiles [0-255]. De hecho, cada mapa de tiles son 1024 bytes es decir, los 1024 (32x32) tiles presentes en cada capa.

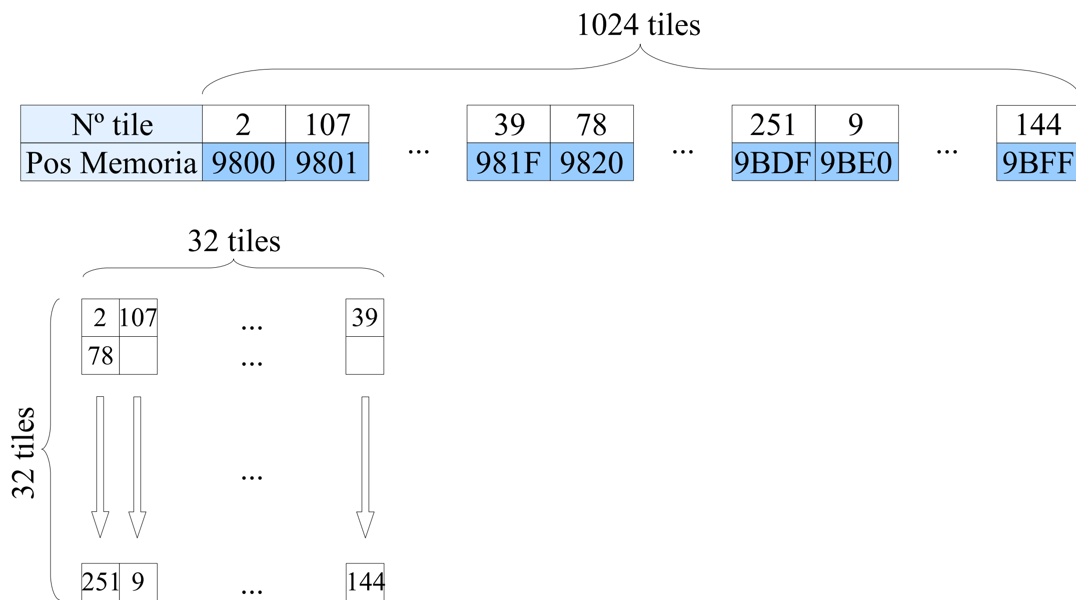


Ilustración 3.18: Descripción de un mapa de tiles

- GBC: Propiedades de tiles (9800-9FFF banco 1): Como la GBC tiene 2 bancos de VRAM, en la posición 9800-9FFF en el primero de ellos se almacena la información de mapas de tiles ya vista y en la misma posición del segundo banco se almacenan los atributos de los tiles. Consiste en 1024 bytes consecutivos donde cada uno de ellos almacena las propiedades de un tile:

  - Paleta de BG [0-7]: La GBC posee 8 paletas de BG por lo que es necesario indicar cual de ellas se usa para cada tile.

  - Nº de banco de datos del tile: La GBC posee 2 bancos de VRAM y los datos del tile pueden proceder de cualquiera de ellos.

  - Otras propiedades: Prioridad de tile sobre Sprites y volteos en eje x e y.

- Atributos de sprites en OAM (FE00-FE9F): Estos 160 bytes son usados para almacenar las propiedades de los sprites. La GB puede mostrar un máximo de 40 sprites por pantalla y de 10 por linea (debido a una limitación del hardware de video). Mediante el registro LCDC se puede establecer el tamaño de los sprites en 8x8 o 8x16. Cada sprite tiene 4 bytes consecutivos para propiedades (40x4 = 160 bytes). A continuación se resume el significado de cada uno de ellos:
  - Coordenada Y: La coordenada en el eje Y relativa a la pantalla del sprite. Se puede ocultar un sprite cambiando la coordenada a un valor de fuera de la pantalla.

- Coordenada X: La coordenada en el eje X relativa a la pantalla del sprite. Se puede ocultar un sprite cambiando la coordenada a un valor de fuera de la pantalla.

- Número de tile: El número de tile dentro del área de datos de tile (8000-8FFF).

- Propiedades: Prioridad de Sprite bajo tiles, volteos en eje X y Y, paleta de Sprites ([0-1] en GB y [0-7] en GBC)y número de banco de datos del tile ([0-1] solo en GBC).

3.4.6 Cartuchos

Tanto la GB como la GBC funcionan con cartuchos de diferentes capacidades y características. Un cartucho de GB tiene el siguiente aspecto:

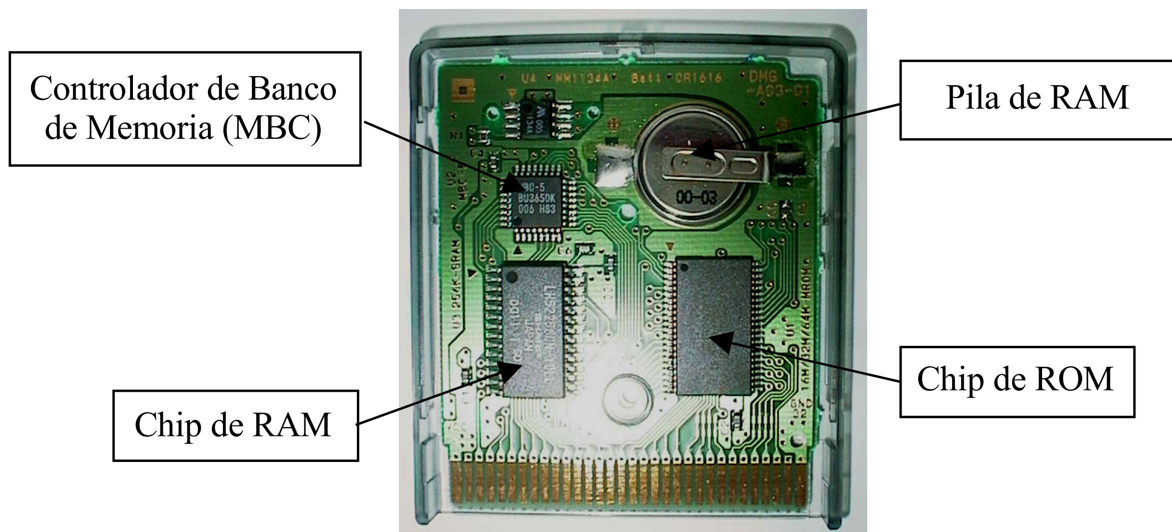


Ilustración 3.19: Interior de un cartucho de Game Boy

Se pueden distinguir claramente 3 componentes principales:

- MBC (Memory Bank Controller): Este chip es el encargado de mapear bancos de ROM o RAM del cartucho a las posiciones 0000-7FFF y A000-BFFF respectivamente. En los cartuchos con 8KB (1 banco) de RAM o menos y con tamaño de ROM menor o igual a 32KB (2 bancos) no está presente por no ser necesario.

- Memoria ROM: Siempre presente y de un tamaño variable, almacena los datos y gráficos del juego. El tamaño de un banco ROM es de 16KB por lo que el tamaño de la ROM siempre es un múltiplo de éste número (mínimo 32KB).

- Memoria RAM: También de tamaño variable y no siempre presente, almacena los avances del juego (partidas salvadas, récords, items desbloqueados, etc). Requiere de una pila para mantener los datos almacenados. El tamaño de un banco RAM es de 8KB aunque existen cartuchos con un único banco de 2KB. Como curiosidad, en los juegos antiguos que carecen de RAM se usan passwords para avanzar en los mismos y también para usar trucos.

Ilustración 3.20: Cartucho de Game Boy sin RAM

A continuación se mostrarán los diferentes tipos de MBC y sus características y limitaciones.

3.4.6.1 Tipos de controladores de banco

Todos los MBC comparten ciertas regiones de memoria donde son accesibles. A continuación se muestran unos cuadros resumen del espacio de memoria controlado por los MBC y sus funciones más habituales. Algunas regiones se usan con objetivos diferentes dependiendo del MBC y algunos MBCs usan una misma zona de múltiples formas. Cuando ésto es así, el color verde es el uso más común que se da a esa zona de memoria y los demás son otros usos específicos a un MBC. Estas tablas no deja de ser una simplificación por lo que para una descripción exacta se recomienda leer [FFRK01] y [FFRK98].

<table>
<tr><th>Bytes</th><th colspan="2">Lectura</th><th colspan="2">Presente en MBC</th></tr>
<tr><td>0000-3FFF</td><td colspan="2">16KB ROM banco 0</td><td colspan="2">Todos</td></tr>
<tr><td>4000-7FFF</td><td colspan="2">16KB ROM bancos 1-n</td><td colspan="2">Todos</td></tr>
<tr><td>A000-BFFF</td><td>8KB RAM banco n</td><td>registros RTC</td><td>Todos</td><td>3</td></tr>
</table>

Tabla 3.11: Lecturas en espacio de memoria de MBC y sus efectos

<table>
<tr><th>Bytes</th><th colspan="3">Escritura</th><th colspan="3">Presente en MBC</th></tr>
<tr><td>0000-1FFF</td><td colspan="3">Habilitar RAM</td><td colspan="3">Todos</td></tr>
<tr><td>2000-3FFF</td><td colspan="3">Número de banco de ROM mapeado</td><td colspan="3">Todos</td></tr>
<tr><td>4000-5FFF</td><td>Número de banco de RAM mapeado</td><td>Bits altos del número de banco de ROM mapeado</td><td>Activar/ Desactivar Rumble</td><td>1,3 y 5</td><td>1</td><td>5 (algunos)</td></tr>
<tr><td>6000-7FFF</td><td>Modo ROM/RAM</td><td colspan="2">Congelar reloj</td><td colspan="2">1</td><td>3</td></tr>
<tr><td>A000-BFFF</td><td>8KB RAM banco n</td><td colspan="2">registros RTC</td><td colspan="2">Todos</td><td>3</td></tr>
</table>

Tabla 3.12: Escrituras en espacio de memoria de MBC y sus efectos

Como puede verse, todos los MBC permiten deshabilitar el acceso a la RAM del cartucho (principalmente para evitar posibles perdidas de datos al apagar la consola) y modificar el banco de ROM mapeado en 4000-7FFF y el de RAM mapeado en A000-BFFF. Escribir en cualquier posición dentro de una misma zona de memoria produce los mismos efectos pues toda la zona mapea el mismo registro interno del MBC. El valor concreto a escribir para realizar la acción es específico de cada zona y MBC.

Existen 4 tipos principales de MBCs documentados más otros 2 no documentados. Debido a la ausencia de información sobre éstos en la documentación y a que ningún juego conocido hace uso de ellos, creo que simplemente se tratan de identificadores que Nintendo reservó para un previsible uso de más tipos de bancos en el futuro que nunca llegó a usarse. En GINAGE sólo se han implementado los MBC 1,2,3 y 5 que cubren prácticamente todos los juegos de la plataforma. Seguidamente se exponen las características que distinguen a cada MBC:

- MBC 1: Admite un máximo de 2MB de ROM y 32KB de RAM. No obstante posee dos modos de funcionamiento entre los que se cambia escribiendo en 6000-7FFF. En ninguno de ellos puede acceder a toda la ROM y toda la RAM a la vez:

    - Modo ROM: Son accesibles 2MB de ROM y 8KB de RAM. 4000-5FFF se utiliza para cambiar los 2 bits más significativos del número de banco de ROM mapeado
    - Modo RAM: Son accesibles 512KB de ROM y 32Kb de RAM. 4000-5FFF se utiliza para cambiar el número banco de RAM mapeado

- MBC 2: Admite un máximo de 256KB de ROM y 512bytes de RAM de los cuales, sólo son aprovechables los 4 bits más bajos.

- MBC 3: Admite un máximo de 2MB de ROM y 32KB de RAM. Además posee un reloj (RTC: Real Time Controller) que funciona aún estando la consola apagado. Este reloj lleva un contador de días, horas, minutos y segundos cuyo estado puede consultarse previa escritura en 6000-7FFF y lectura en A000-BFFF (que puede mapear un banco de RAM o uno de los registros RTC). La saga "pokemon" usa el RTC para saber la hora real del juego (establecida al crear una nueva partida) y lo usa para eventos de día y de noche.

- MBC 5: Admite un máximo de 8MB de ROM y 128KB de RAM. Algunos MBC5 incorporaran un Rumble Pack (accesorio vibratorio) y tienen limitada la RAM a un máximo de 32KB. El motor se activa y para escribiendo en 4000-5FFF.

- Desconocidos: Existen algunos MBCs indocumentados. La poca información que se ha recopilado aquí está sacada del código fuente del emulador GEST y del fichero Readme de GNUBoy.

    - MBC 4: Se sospecha que este MBC es idéntico al 5 pero anterior a la salida al mercado de la GBC. No se conoce ningún juego que lo use.
    - MMM01: La única ROM que usa este MBC es "*Momotarou Collection 2*" y se sospecha que puede haber sido volcada incorrectamente.
- Otros:

    - MBC7: Usado únicamente por el juego "*Kirby Tilt & Tumble*", es un MBC que incorpora un sensor de movimiento.
    - HuC1: MBC desarrollado por Hudson similar al MBC1 y que permite usar el puerto de infrarrojos de GBC. Sólo lo usa el juego "*Fighting Phoenix*".
    - HuC3: MBC desarrollado por Hudson que permite usar el puerto de infrarrojos de GBC. Solo lo usan los juegos "*Robopon Star*" y "*Robopon Sun*".

3.4.6.2 Cabecera

Todos los cartuchos incorporan una cabecera al principio del primer banco ROM, concretamente en las posiciones 0100-014F. El formato de la misma sirve para identificar al cartucho (y a la ROM) como un juego de GB. La información más relevante desde el punto de vista de la emulación se resume a continuación:

- GBC Flag (0143): Indica si se trata de un juego GBC Only (0xC0), GBC Compatible (0x80) o de DMG (cualquier otro valor).

Ilustración 3.21: Apariencia externa de cada tipo de cartucho

Los cartuchos GBC Only son transparentes y evidentemente sólo funcionan en GBC. Suponen 3/4 del catálogo de juegos de GBC (Compatibles + Only GBC). Los cartuchos GBC Compatibles son de color negro. Aunque funcionan en ambas plataformas en la GBC presentan un aspecto remozado gracias al uso de 56 colores. Los cartuchos originales DMG son de color gris y funcionan en ambas plataformas.

- Tipo de cartucho (0147): Especifica el MBC usado por el cartucho así como la presencia de RAM o hardware externo en el mismo (RTC, Rumble, etc).

- Tamaño de ROM (0148): Especifica la cantidad de memoria ROM del juego que puede oscilar entre 32KB (sin MBC) y 8MB (512 bancos).

- Tamaño de RAM (0149): Especifica la cantidad de memoria RAM del juego que puede oscilar entre 0KB (ausencia de memoria RAM) hasta 128KB (16 bancos).

- Otros campos: Punto de entrada al código, logo de Nintendo, título del juego, código del licenciatario del cartucho, SGB Flag, código de destino (Japón o resto del mundo), version de la ROM y checksums de cabecera y ROM completa.

3.4.7 Otros periféricos (no emulados)

Debido al éxito de la GB, Nintendo fabricó varios accesorios, la mayoría de ellos acoplables a la misma en la ranura de cartuchos. A continuación se presentan brevemente pues no han sido emulador por el emulador:

- Super Game Boy: Permite jugar a los juegos de GB y GBC Compatibles en la SNES. Básicamente es una GB con forma de cartucho de SNES y que se comunica con ésta para la

E/S y pintado gráfico. Existen juegos especialmente preparados para SGB para, por ejemplo, permitir un sonido de mayor calidad usando el PSG de la SNES.

Ilustración 3.22: Accesorio Super Game Boy

- Game Boy Camera y Game Boy Printer: El primero de ellos consiste en una cámara acoplable en la ranura de cartuchos capaz de tomar fotos en blanco y negro (2 bpp) de 128x123 pixels. Se vendía junto al segundo, una impresora térmica sin tinta y capaz de imprimir mediante un papel especial las imágenes en blanco y negro. Aparte del uso evidente de imprimir fotos, también permite imprimir puntuaciones de juegos e imágenes. Ambos dispositivos no tuvieron mucho éxito comercial aunque algunos emuladores actuales dan soporte a la impresora (como KiGB).

Ilustración 3.23: Accesorios Game Boy Printer y Game Boy Camera

- Cable Gamelink: Usado para conectar dos GB con el mismo juego entre sí (multijugador) o 2 juegos compatibles (p.e pokémon edición roja y azul que permiten desbloquear pokémons exclusivos al conectarse). El conector de la consola ha cambiado de forma en cada modelo de GB aunque el protocolo ha permanecido inalterado en GB,GBP y GBC: Un protocolo serie crudo que envía 1 byte cada vez usando los registros SB (FF01) y SC (FF02). Aunque ha sido implementado por varios emuladores, el juego online es casi imposible (salvo contados juegos) debido a la velocidad de transferencia de este puerto y los retardos en TCP/IP. Sin embargo, el multijugador en la misma máquina si es posible.

Ilustración 3.24: Accesorio cable Gamelink

Capítulo 4. MÉTODO DE TRABAJO

4.1 Decisiones de diseño

4.1.1 Elección del sistema a emular

Para cumplir con los objetivos enunciados en el capítulo 2, se decidió implementar un emulador de Game Boy usando la plataforma Ewe. De todos los sistemas que se podrían emular, la portátil Game Boy de Nintendo se presenta como la mejor candidata por varios motivos:

- *Arquitectura interna más simple que otras videoconsolas*: La Game Boy es una consola portátil de 8 bits, mucho menos compleja que las actuales, con un funcionamiento más simple y, en definitiva, más fácil de emular. Por lo tanto, el emulador es más sencillo y es más factible de funcionar en una PDA a una velocidad decente.

- *Gran catalogo de juegos disponible*: La Game Boy posee un catalogo de juegos muy amplio, lo que la hace útil como emulador.

- *Posee bastante documentación*: La documentación sobre el hardware de videoconsolas no abunda precisamente. La mayoría de los documentos que están en la red son incompletos, están anticuados o tienen inconsistencias (hay que tener en cuenta que los redactores de esos documentos obtuvieron ese conocimiento por ingeniería inversa). La Game Boy, al ser una consola antigua y ampliamente emulada en muchas arquitecturas, se encuentra un poco mejor parada que sus contemporáneas en éste aspecto.

4.1.2 Elección del lenguaje y la plataforma

En la elección de la plataforma y el lenguaje se estudiaron diferentes alternativas expuestas a continuación.

Java es un lenguaje de sobra conocido por el autor: La experiencia obtenida en los estudios universitarios y puesto de trabajo de programador en Indra con Java durante varios años facilita la codificación. Por otro lado es un lenguaje maduro en lo que a opciones de portabilidad de refiere, poseyendo máquinas virtuales para un rango de arquitecturas considerable, haciéndolo ideal para el desarrollo de este proyecto.

En un principio se pensó en J2ME. Aunque es ampliamente usado por la industria del

videojuego para dispositivos móviles y existe abundante documentación, tiene el problema de el alto coste de licencia por compilador y el no ser libre. Por otro lado, J2ME sólo funciona en un perfil concreto de dispositivos móviles (un conjunto de dispositivos con capacidades computacionales similares). Lamentablemente, no funciona en PC.

El resto de alternativas presentaban un estado de desarrollo prematuro, con MV que implementaban el lenguaje parcialmente o sin soporte a ninguna librería gráfica, escasa documentación, etc. Las únicas alternativas superiores a J2ME que se encontraron fueron Superwaba y Ewe (ver [Rol06]). Los criterios a continuación expuestos son los que llevaron a la elección de Ewe como plataforma de desarrollo:

- Ewe funciona en un rango de PDAs mayor que Superwaba. El que no funcione en Symbian descarta los teléfonos móviles y algunos híbridos de móvil/PDA. Por otro lado es muy improbable que estos dispositivos llegaran ejecutar a una velocidad razonable el emulador por su menor potencia de cómputo. Aunque no funciona en Palm OS, su creador asegura que está previsto portar la máquina virtual pronto.

- Ewe tiene una librería más potente (y también un máquina virtual un poco más grande). Por ejemplo, es capaz de distinguir entre eventos keypressed y keyreleased (algo imprescindible para la implementación cómo despues se verá) y tiene varias formas de manipular imágenes pixel a pixel, por lineas e incluso rotarlas e invertirlas en un eje.

- La máquina virtual de Ewe tiene licencia LGPL mientras que la de Superwaba es propietaria. Todas librerías tienen licencia LGPL (en Superwaba no están todas disponibles bajo GPL, algunas son privativas y con coste económico asociado).

- A diferencia de Superwaba, toda la documentación para desarrolladores y de la API de Ewe está disponible si bien la calidad de la documentación deja mucho que desear en comparación con la de Superwaba: API más caótica, algunas clases/métodos sin documentar, etc.

- Otros factores:
  - Compilador Jikes soportado entre otros
  - Librerías para interfaz de usuario de más calidad que Superwaba (más cercana a Swing que a AWT)

4.1.3 Elección del tipo de emulador

Como ya se comentó en el apartado 3.1.2 según la forma en la que se implementen los emuladores, se puede distinguir entre emuladores intérpretes (o interpretadores) y emuladores

recompiladores. Aunque las ventajas iniciales de un emulador recompilador podrían hacerlo preferible desde el punto de vista de la eficiencia, la complejidad para llevarlo a cabo excedería el ámbito de un proyecto fin de carrera. El desarrollo de un emulador de éste tipo requiere una implementación a muy bajo nivel (de ensamblador) lo que obligaría a hacer diferentes versiones del emulador para cada arquitectura diferente tanto en PDA como en ordenadores de sobremesa. También requiere de conocimientos avanzados de la arquitectura de ambas máquinas (emuladora y emulante), en especial del juego de instrucciones de ambas. En el caso de emuladores más complejos, estos conocimientos podrían necesitar extenderse a ámbitos como detenciones en el cauce de la CPU, MMU, ocupación de buses....Un emulador interpretador es mucho más fácil de implementar puesto que el núcleo del emulador es mucho más sencillo (véase apartado 1.3.2) . Además, de implementarse en un lenguaje de alto nivel, solo requiere conocer detalles de la arquitectura emulada.

4.1.4 Elección del modelo de desarrollo

Se ha escogido un ciclo de desarrollo evolutivo y orientado a prototipos por ser el que mejor se adapta a la propia naturaleza de un emulador. Ésta es la metodología seguida por todos los desarrolladores de emuladores y en los próximos apartados se verá su funcionamiento y sus ventajas.

4.2 El ciclo de desarrollo en emulación

El desarrollo de un emulador es una tarea compleja y extremadamente lenta. Resulta especialmente frustrante la lentitud en la obtención de resultados frente a la codificación. Los principales retos a los que se enfrenta el desarrollador son:

- Documentación del sistema a emular: Como ya se ha citado antes, es extremadamente difícil encontrar buena documentación del sistema a emular. La documentación existente procede de ingeniería inversa y suele ser incompleta, tener contradicciones o lagunas importantes. En los mejores casos puede que el desarrollador encuentre un documento oficial filtrado por Internet, explicando en mayor o menor grado de detalle parte del funcionamiento. En los peores, la ausencia de documentación del sistema es total especialmente con las videoconsolas más recientes en el mercado. Esto obligará con toda seguridad al desarrollador a armarse de paciencia y altos conocimientos de electrónica y arquitectura de computadores para estudiar el valor de los bits en memoria y registros en cada instante y tratar de saber cual es el comportamiento interno real de la máquina. Las dudas sobre el comportamiento de un componente se solventan usando algunas veces el método del ensayo

y error. Es frecuentemente la implementación de hacks en los emuladores que consisten en tratamientos específicos en la emulación de diferentes juegos para que funcionen, muchas veces porque la documentación del funcionamiento de algunas partes del hardware no es completa o no es del todo correcta y con algunos juegos específicos, la implementación genérica simplemente no funciona. A día de hoy no existe el emulador perfecto de ninguna videoconsola. Los que más cerca están de una compatibilidad total utilizan abundantes hacks, oscureciendo aún más si cabe el código de estos emuladores.

- Dificultad del propio desarrollo: En parte debido al punto anterior, en parte debido a la dificulta inherente al desarrollo de emuladores, la mayoría de las veces el desarrollo se convierte en un proceso lento y tedioso. El desconocimiento del funcionamiento de ciertos componentes conlleva inevitablemente a no saber implementar correctamente ciertas partes. No existen depuradores por lo que cada desarrollador se fabrica el suyo propio que permita ejecutar instrucción a instrucción cada ROM, comprobando el estado interno del emulador (registros, memoria, interrupciones, regiones protegidas o duplicadas de memoria, timers, chips activos.....). En ocasiones, sí existen emuladores del sistema a emular lo suficientemente buenos que tengan depurador integrado, es posible usar uno de ellos para comparar resultados instrucción a instrucción con el desarrollado. La complejidad de desarrollo del depurador propio vendrá dada por la complejidad del hardware a emular. Las consolas más modernas son extremadamente complejas. El hardware de las consolas tiende a converger cada vez más con el de los Pcs convencionales, complicando el diseño interno y añadiendo nuevos componentes que antes no existían El desarrollo de esos emuladores, por todo lo anteriormente citado, puede prolongarse durante años. En el caso de consolas previamente emuladas, el desarrollador puede tener acceso a código fuente que le ayuden cuando la documentación falla, pero en los emuladores nuevos normalmente se requiere de conocimientos elevados de ingeniería inversa para estudiar el comportamiento del hardware ante diferentes entradas.

Por estas razones es por las que un emulador debe desarrollarse con una metodología ágil y que permita una evolución incremental y robusta del proyecto: La metodología orientada a prototipos y evolutiva.

4.2.1 Metodología evolutiva orientada a prototipos

En ésta metodología se realizan múltiples iteraciones. Se parte de un conjunto de funcionalidades a emular (algunas independientes y otras dependientes de otras) y la unión de todas ellas da lugar a la funcionalidad final del emulador. En cada iteración *i* se seleccionan un subconjunto de funcionalidades nuevo que engloba a los subconjuntos anteriores ([0...i-1]), se implementa el

prototipo con funcionalidad reducida, y se prueba para asegurarse que las funcionalidades nuevas han sido correctamente implementadas y las antiguas siguen estándolo también

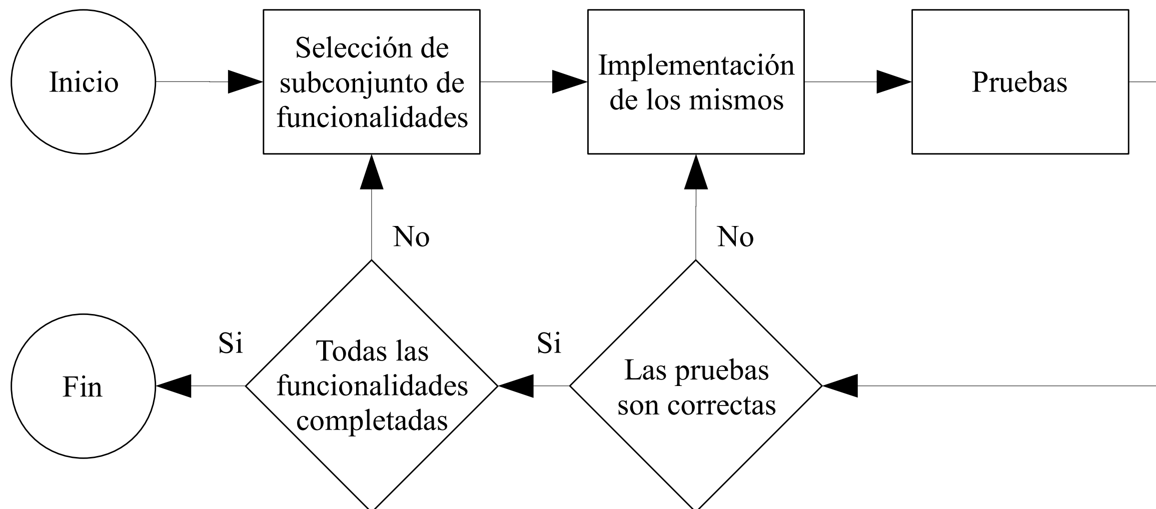


Ilustración 4.1: diagrama de etapas de metodologia evolutiva orientada a prototipos

4.2.2 Las pruebas

La naturaleza de un emulador facilita la metodología de las propias pruebas. Las pruebas son de tipo caja negra en su mayoría y generalmente consisten en ejecutar una ROM en el emulador e ir comprobando el resultado de la emulación.

En las etapas iniciales, éstas pruebas son difíciles de hacer al haber muy poca funcionalidad implementada (especialmente cuando aún no se han implementado los gráficos) pero se pueden llegar a hacer con ROMs extremadamente sencillas: demos de homebrew. Tienen la ventaja de que suele estar disponible el código fuente. Las hay desde las más simples con apenas opcodes o uso de características avanzadas de la máquina y por lo tanto ideales para las primeras pruebas, hasta las más avanzas, algunas preparadas para probar emuladores, pues realizan un test de todos los caminos lógicos, opcodes, etc.

En etapas posteriores se suelen usar también ROMs comerciales junto a un desensamblador para obtener el código original a fin de saber que instrucción se ejecuta en cada momento.

Se comparan el funcionamiento, los gráficos y sonidos obtenidos con los de otro emulador de referencia o, mejor aún, el hardware real. Si los resultados difieren en algún momento (gráficos corruptos, emulación detenida, saltos de excepciones, sonido incorrecto, etc) hay un fallo en el código y habrá que revisarlo.

Es en este momento cuando se pasa a ejecutar instrucción a instrucción el emulador, comparando que los valores de ciertas variables y posiciones de memoria tras cada instrucción sean los esperados. Para realizar ésto se hace imprescindible el desarrollo de un depurador a la par que el emulador que permita, al menos, ejecutar instrucciones una a una, ejecutar un número de

instrucciones seguidas y ver los valores de ciertas posiciones de memoria y variables por pantalla. Por todo esto, la depuración es un proceso lento y tedioso.

4.2.3 Ventajas de la metodología evolutiva orientada a prototipos

Este modelo presenta algunas ventajas como son:

- Depuración de errores temprana: Al ir escribiendo pequeños subconjuntos de funcionalidades, es más fácil encontrar los posibles fallos que se han cometido en la implementación. La depuración de errores es más cómoda y rápida al tener una cierta garantía de la corrección de las funcionalidades previamente escritas. En el peor de los casos, una funcionalidad que deje de funcionar en la iteración actual, podrá ser revisada en la inmediatamente anterior y comprobar las diferencias rápidamente.

- Obliga a seguir una política de control de versiones: Cada una de ellas es una iteración. Sin este modelo, el desarrollador correría el riesgo de querer implementar muchas/todas las funcionalidades en pocas iteraciones/una iteración, lo cual conduciría a un casi seguro fracaso del proyecto.

- Mejor estructuración /planificación del proyecto: El desarrollador se ve obligado a pensar en términos de "paquetes de funcionalidades" por lo que se consigue una idea global del proyecto mejor y la planificación y estructuración del mismo se ve mejorada.

- Mayor motivación del desarrollador: Es mucho más gratificante el ver un programa que gradualmente gana funcionalidad a otro que parece no progresar (aunque en realidad lo esté haciendo) por más que su desarrollador escribe lineas de código. El ir desarrollando, probando y cerrando funcionalidades permite al desarrollador sentir que el proyecto avanza.

4.3 Arquitectura del sistema

La videoconsola Game Boy, como cualquier otro sistema, consta de varios componentes físicos como pueden ser: CPU, Memoria, cartuchos (con su propia memoria y MBC) y hardware de Video entre otros. Podría considerarse una arquitectura del sistema en la que cada componente fuese una clase. Cada clase encapsularía únicamente el componente de su componente, conocería al resto de las clases de las que depende su funcionalidad y seria conocida por las que poseen una funcionalidad dependiente con ella. Se podrían usar interfaces para la CPU o el Video que definieran la funcionalidad común de éstos componentes para, posteriormente, implementar cada

una de ellas en 2 clases: una para GBC y otra para GB, cada una con sus particularidades.

Sin embargo, ha de tenerse presente las plataformas a las que va orientado este emulador, las PDA. El uso de múltiples clases, herencia, polimorfismo y otras caracteristicas avanzadas de la programación orientada a objetos son un lastre al rendimiento aunque pudiera ser una ayuda a la programación. La encapsulación perfecta y los patrones han de relajarse en este proyecto en pos de la eficiencia. Así, se han identificado 2 grandes componentes: CPU y Video. Cada uno de ellos es implementado por la clase del mismo nombre:

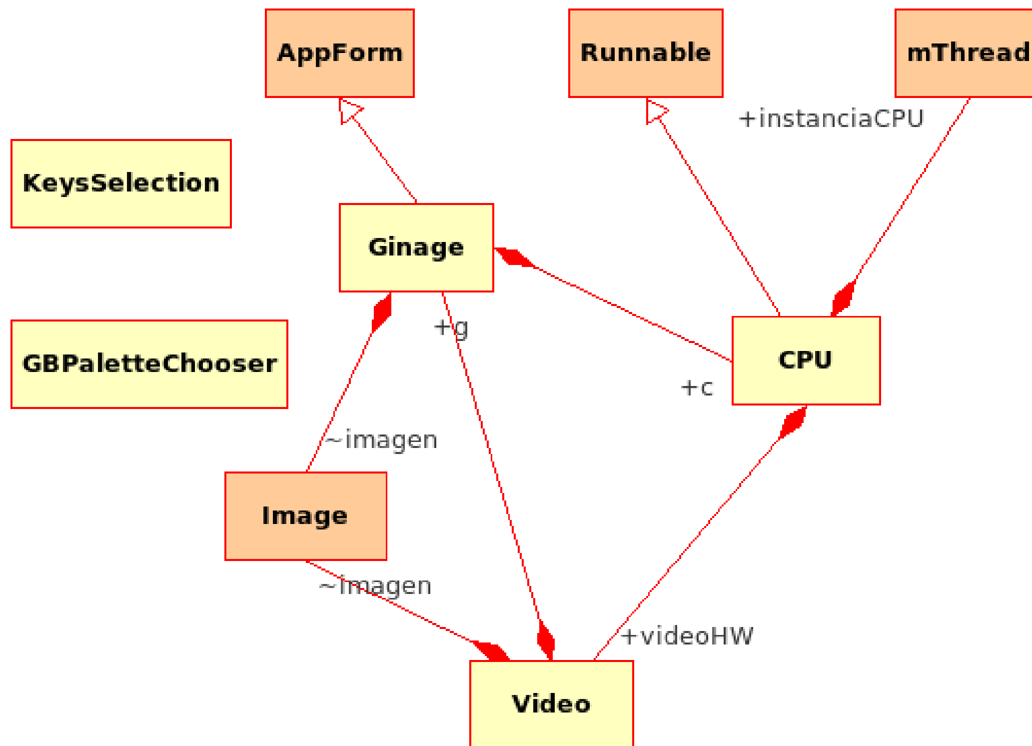


Ilustración 4.2: Diagrama UML general de clases

En el diagrama UML pueden distinguirse clases propias (color amarillo) y algunas de las clases de Ewe usadas, más importantes. A continuación se resume brevemente el funcionamiento de cada clase:

- Ginage: Es la clase principal (especialización de formulario principal de Ewe). Gestiona la ventana principal y las secundarias (edición de paletas y configuración de teclas), gestiona los eventos de teclado, y las diversas opciones de los menús, así como guarda/recupera el estado de las opciones/paletas y teclas personalizadas. Conoce a la clase CPU e Imagen, esto es, mantiene una referencia a las mismas.

- KeysSelection y GBPaletteChooser: Ambas clases son especializaciones de formularios y representan los objetos ventana de configuración de teclas y edición de paletas

respectivamente. Forman parte del paquete ginageUtils y son creadas desde GINAGE únicamente cuando son mostradas (de hay que GINAGE no conozca a esas clases en el diagrama).

- CPU: Es la principal clase emuladora y se encargada de la emulación de la CPU de la GB. Algunas tareas que realiza son las siguientes:

  - lectura/escritura en registros/memoria: La CPU tiene acceso al mapa completo de memoria y puertos (0000-FFFF) y a los registros (A,B,C,D,E,H,L,F,PC,SP). Mediante varias funciones gestiona la lectura y escritura de bytes en las distintas posiciones de memoria.

  - gestión de las interrupciones y timing interno: Se comprueba antes de cada opcode si hay interrupciones y de haberlas se atienden. También se lleva un control de los ciclos de procesador de GB transcurridos, incremento de timers y del estado actual del LCD (0,1,2 o 3). Cada vez que se cambia el estado, se realizan tareas propias al mismo, p.e en el estado 3 se manda calcular la linea actual a la clase Video y en el estado 1 se manda refrescar la pantalla también a Video.

  - inicio/parada y reinicio de la CPU: Parar la CPU equivale a para la emulación. Se puede parar/iniciar en cualquier momento el emulador mediante la interfaz de usuario, así como hacer un reset hardware de la GB (reinicio de la ROM actual).

  - carga de ROMs y RAM (save) asociado: Desde la interfaz principal se permite cargar una nueva ROM. Al cargarse la ROM se reinicia el estado interno del emulador, se carga en (A000-BFFF) un save previo de existir éste y se inicializan ciertas variables dependientes de la información de la cabecera que condicionarán la emulación (si es una ROM de GBC, tamaños ROM, RAM, tipo de MBC, etc).

  - inicialización de estructuras internas: Como tablas precalculadas para las instrucciones SWAP y DAA, estado de variables internas a la propia emulación, etc.

  - implementación del cauce de instrucciones y todos sus correspondientes opcodes: Como un gran switch donde en función del opcode se realiza el tratamiento adecuado al mismo, por lo general consistente en modificar el valor de un registro o posición de memoria y los flags (registro F).

Como se puede ver en el diagrama, la CPU se ha implementado como un hilo (mThread) de Ewe (hereda de Runnable y mantiene una referencia a sí misma como hilo). Ewe no soporta múltiples hilos interrumpibles (pre-emptive multi-threading) para aumentar la portabilidad de la MV, esto es, Ewe puede trabajar con varios hilos a la vez pero sólo uno de ellos se ejecutará en casa instante y al no ser interrumpible éste por la MV, monopolizará el procesador hasta que:

- Acabe su método run()
- Deje paso a otro voluntariamente (invocando a métodos de tipo sleep() o wait())

Ewe posee una cola de eventos interna, gestionada por el hilo de eventos (independiente del/los hilo/s de usuario y otros hilos como el recolector de basura). En un programa típico con GUI, la mayor parte del tiempo la CPU se encuentra ociosa esperando eventos. Las operaciones de GUI y de I/O en Ewe hacen llamadas a sleep() o wait() de manera que todos los hilos colaboran para cederse la CPU de forma voluntaria y la gestión de eventos funciona correctamente.
Sin embargo, la CPU realiza un gran consumo de procesador debido al bucle infinito de interpretación de instrucciones. Ésto hace que el hilo de usuario de la CPU monopolice el procesador hasta el punto de que, sin una interrupción periódica voluntaria del mismo, los eventos nunca son atendidos. Es por esto que periódicamente (cada 4 frames), el hilo de CPU se duerme para que los eventos puedan ser procesados.

- Vídeo: Se encarga de la emulación de los gráficos. Ésto comprende tareas propias de la GB como:

  - Calcular la linea actual: Generar el color para cada pixel de la linea actual, teniendo en cuenta las 3 capas (BG,WIN y SPRITE) y sus prioridades y colores de paleta.

  - Refrescar la imagen 60 veces por segundo: Una vez calculadas todas las lineas, al entrar en el modo 1 del LCD se pinta en contenido por pantalla.
  - Actualizar las paletas: Las escrituras en las posiciones de memoria FF47-FF49 y FF68-FF6B que mapean los registros de las paletas de GB y GBC respectivamente producen cambios en los colores de las paletas.

  También se encarga de tareas ajenas al propio funcionamiento de la GB pero que son parte de la emulación como:

- Escalado de imagen: Desde la interfaz puede seleccionarse tamaño 1x (sin escalado) o tamaño 2x. El tamaño 2x requiere de un escalado de imagen previo a mostrarse por pantalla.

- Cálculo de imágenes por segundo (FPS): Los datos de FPS obtenidos dan una idea de la velocidad del emulador. El número de frames pintados y tiempo transcurrido es llevado por esta clase.

- Implementación del salto de frames (frameskip): Esta característica permite evitar pintar algunos frames por pantalla (y calcular sus lineas) aumentando la velocidad general del emulador a costa de animaciones más bruscas.

- Cambio de colores en GB: La ventana de personalización de paletas (clase GBPaletteChooser) permite cambiar los 4 colores de la GB normal para mostrar hasta 12 colores diferentes.

A continuación se verán en mayor detalle cada una de las clases y se explicarán en pseudocódigo las partes más importantes de las mismas.

4.3.1 Diagrama de estados genérico

El emulador pasa por lo siguientes estados desde que se inicia hasta que es terminado por el usuario

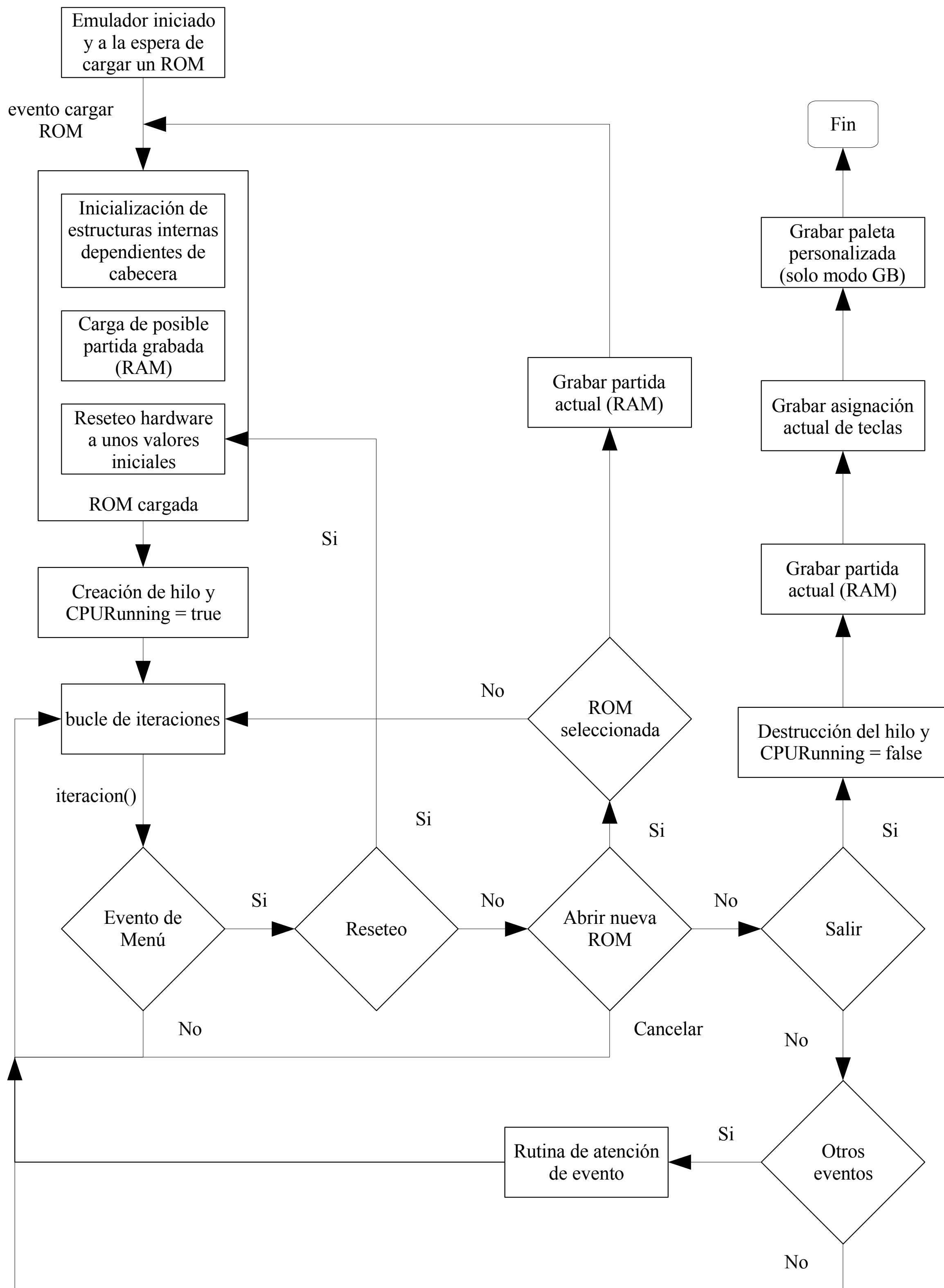


Ilustración 4.3: Diagrama de estados genérico

En el diagrama se han obviado algunos eventos de menú menos importantes. También se

han obviado los eventos de teclado. Ambos tipos de eventos son gestionados por la clase GINAGE y se verán en el capítulo sobre la misma clase.

El evento principal para que se inicie la emulación de opcodes es la carga de una nueva ROM. Tras éste, el emulador entra en un bucle infinito de decodificación de opcodes. La carga de una ROM se verá en detalle en el apartado referente a la clase CPU

Otros eventos no detallados en el esquema que pueden alterar el bucle son:

- Pausar el emulador: Detiene el hilo principal o lo vuelve a lanzar si ya se encontraba previamente pausado
- Mostrar cualquier menú: Detiene el hilo principal
- Ocultar cualquier menú: Reanuda el hilo previamente parado al mostrarse el menú (excepto si está en modo pausa claro).

4.3.2 La clase CPU

Tareas principales

Su tarea principal consiste en ejecutar instrucciones del LR35902 una a una , implementando el funcionamiento completo y preciso de la ISA del procesador: Timming interno de cada opcode, operación y resultado de la misma y posibles cambios en los flags (registro F).

La CPU está implementada como un hilo de Ewe (mThread) que constantemente ejecuta el siguiente código:

```
public void run(){
  while(CPURunning)
    iteración();
}
```

Ilustración 4.4: Bucle principal del emulador

El método run está constantemente realizando iteraciones de la CPU excepto en las condiciones vistas en el punto anterior. El siguiente diagrama detalla el funcionamiento del método iteración:

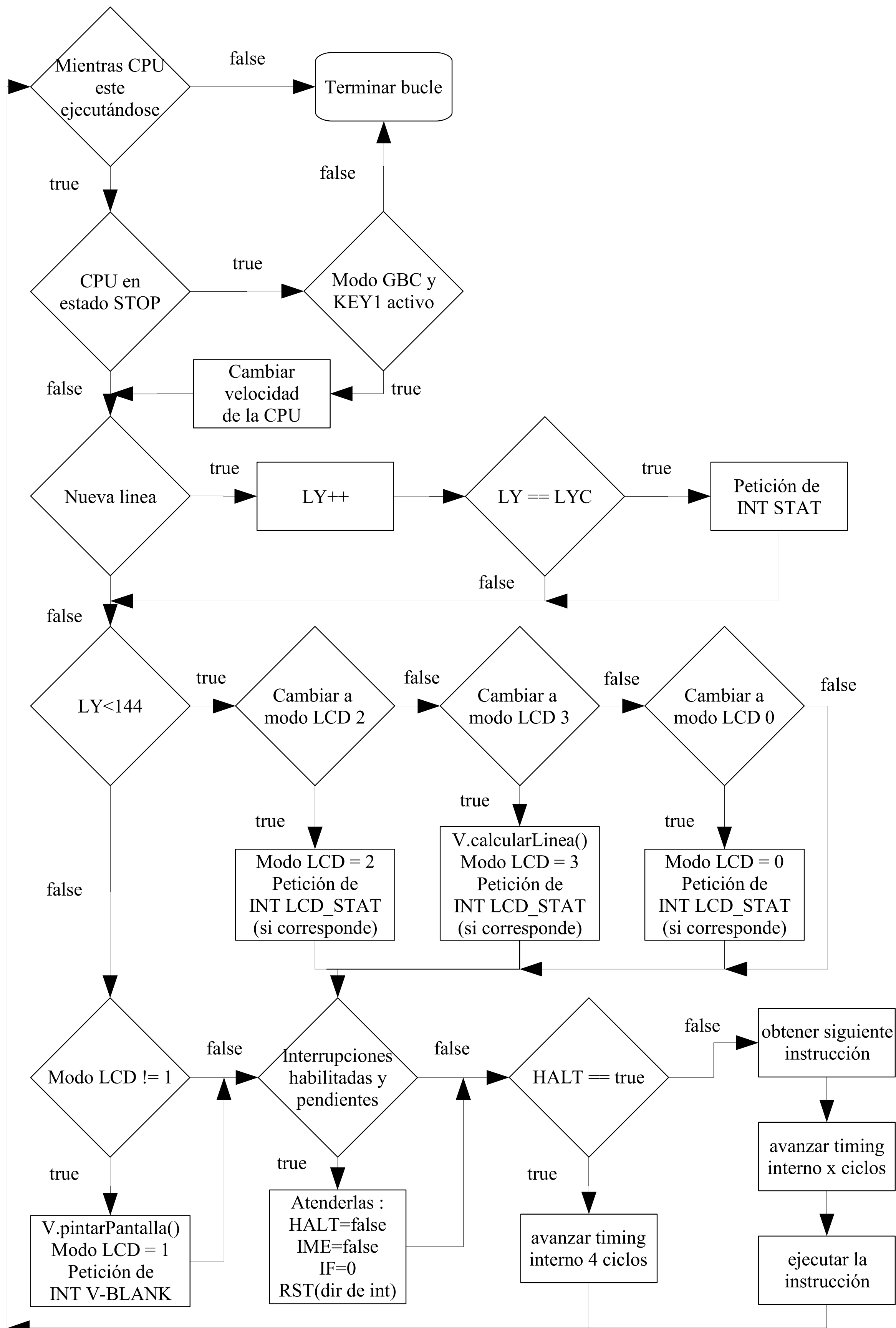


Ilustración 4.5: Diagrama de funcionamiento del método iteracion()

Como puede observarse, el método iteración pasa por las siguientes fases:

- Comprobación de posible cambio de velocidad (en modo GBC)
- Comprobación de posible cambio de linea: Una linea dura aproximadamente 456 ciclos (912 en modo GBC).
  - Si es una de las lineas visibles, comprobar posible cambio de modo LCD: El modo LCD 2 dura 80 ciclos, el 3 dura entre 93 y 213 dependiendo del número de sprites que halla en la linea actual y el 0 dura el resto hasta 456. Sabiendo el timing interno del emulador para la linea actual se puede controlar en que momento se ha de producir un cambio del modo LCD
  - Si no es una de las lineas visibles, comprobar posible cambio a modo LCD 1 (V-Blank). Las lineas 144 a 154 no son visibles por pantalla y corresponden al modo 1 del LCD.
- Si hay interrupciones habilitadas y pendientes, atenderlas: deshabilitando las interrupciones y posible modo HALT y saltando a la rutina de atención de la interrupción más prioritaria.
- Si CPU está en modo HALT, avanzar timing y terminar iteración. Obsérvese que la única forma de salir del modo HALT es mediante la producción de una interrupción.
- Si no está en modo HALT:
  - Obtener siguiente instrucción apuntada por PC
  - Avanzar timing interno: llamando a la función internalTiming(nCiclos). Aunque parte del timing interno es realizado en la propia iteración (concretamente el timing interno para cambiar el modo del LCD), el timing interno para incrementar los timers de la GB (registro DIV y TIMA) y por lo tanto la petición de interrupción de timer es gestionada aquí. Obsérvese que durante el modo HALT los timers no avanzan.
  - ejecutar instrucción: realizar los cálculos asociados al opcode y modificar los flags.

La función internatTiming realiza el siguiente pseudocódigo:

```
internalTiming(int nCiclos) :
  incrementar contadores de ciclos;
  si contador DIV > ciclos avance DIV :
    incrementar DIV
    si DIV desborda : DIV=0
  si timer activo :
    incrementar contador de timer
    si contador TIMA > ciclos avance TIMA :
      incrementar TIMA
      si TIMA desborda :
        TIMA = TMA
        petición de interrupción de TIMER
```

Ilustración 4.6: Pseudocódigo de la función internalTiming

Emulación de atributos principales

Como ya se ha comentado, ésta es la clase más importante desde el punto de vista de la emulación. A continuación se pasa a describir como se han emulado las variables más importantes de la misma:

- Emulación de registros: Cómo ya se ha visto, la GB posee varios registros de 8 bits (A,B,C,D,E,H,L y F) y dos de 16 (PC y SP). Todos los registros de 8 bits a excepción del F son accesibles directamente por el programador. También son direccionables en parejas de 16 bits (AF,BC,DE y HL) aunque la mayoría de las instrucciones los usan en modo 8 bits a excepción del registro HL que es usado por muchas más instrucciones del ISA como pareja de 16 bits que por separado (ver anexo B: juego de instrucciones). Es evidente que la forma de acceso más frecuente a los registros ha de ser la más eficiente por lo que se ha optado por implementar los siguientes registros como variables int de java: A,F,B,C,D,E,HL,PC y SP). Así, por ejemplo, cuando se quiera acceder al registro H se requerirá una operación de desplazamiento de HL y una de AND: H = (HL>>>8)&0xFF y cuando se quiera acceder al registro BC se requerirá una operación de desplazamiento de B y una de OR con C (BC = B<<8 | C;). Obsérvese que se confía en que el contenido de los registros B y C sean en verdad un valor de 8 bits ya que no se hace una operación de tipo &0xFF. En realidad basta con asegurarse que los valores escritos en los registros de 8 bits sean menores o iguales a 0xFF para ahorrarse la operación de AND en los accesos y, ya que los accesos a registros son la operación más frecuente (especialmente los accesos de 8 bits), se estará una vez más haciendo más eficiente la operación más común.
  Las lecturas de registros de 8 y 16 bits, al ser tan frecuentes en los opcodes, se realizan mediante las funciones getR8L, getR8H y getR16. Las dos primeras reciben por parámetros un registro de 16 bits (int) y devuelve su parte baja o alta respectivamente. La última recibe dos registros de 8 bits (dos ints) y los compone devolviendo el registro de 16 bits resultante.
  Las escrituras no se han implementado como funciones porque habría que crear una para cada registro.

- Emulación de la memoria: La GB puede acceder a 0xFFFF posiciones de memoria diferentes. Algunas de ellas mapearán siempre la misma zona de memoria (p.e 0000-3FFF que mapea el banco 0 de ROM). y otras mapearán diferentes zonas (p.e A000-BFFF mapea en cada instante un banco de RAM del cartucho diferente). La memoria principal se ha implementado como un array de ints de 0xFFFF posiciones y las otras memorias presentes en la GB también se han implementado así:

  - ROM de cartucho: array de ints de tamaño dependiente de la cabecera. Las lecturas en

0x0000-0x7FFF son redireccionadas al banco de esta memoria que corresponda (el actualmente mapeado)

- RAM de cartucho: array de ints de tamaño dependiente de la cabecera. Las lecturas en 0xA000-0xBFFF son redireccionadas al banco de esta memoria que corresponda (el actualmente mapeado)

- RAM interna: array de ints de tamaño 0x2000 en modo GB (2 bancos) o 0x8000 en modo GBC (8 bancos). Las lecturas en 0xC000-0xDFFF son redireccionadas al banco de esta memoria que corresponda (el actualmente mapeado).

- VRAM: array de ints de tamaño 0x2000 en modo GB (1 banco) o 0x4000 en modo GBC (2 bancos). Las lecturas en 0x8000-0x9FFF son redireccionadas al banco de esta memoria que corresponda (el actualmente mapeado).

Por lo tanto muchas de las 0xFFFF posiciones de la memoria principal en realidad no contienen datos, simplemente se limitan a hacer la lectura en otro array. Aunque estas posiciones son una pérdida de espacio, hacen las lecturas mucho más eficientes y como ya se vio, las lecturas de bytes de memoria era la función más llamada por el emulador por lo que hacerla lo más eficiente posible repercutirá de manera notoria en el rendimiento. Los otros arrays de memorias si son redimensionados cada vez que una nueva ROM es cargada.

Las lecturas de memoria se realizan mediante las funciones:

- readByte(int dir): Lee un byte unsigned de la dirección de memoria dir. Es la función principal y la más usada con diferencia por casi todos los opcodes. Las dos funciones siguientes utilizan a ésta.

- readNextSignedByte(): Lee el siguiente byte (con signo) apuntado por PC. El byte leído es convertido a un valor entre 127 y -128. En algunas operaciones concretas (como el salto relativo a PC: JR) se usan bytes con signo.

- readNextWord(): Lee los siguientes dos bytes unsigned apuntados por PC. Las instrucciones que trabajan con direcciones de memoria (JP y CALL), así como algunas instrucciones LD de 16 bits trabajan con palabras de 16 bits sin signo.

```
public int readByte(int dir){

  //Direccion tiene que ser de 16 bits
  dir&=0xFFFF;

  switch(dir&0xF000){
    case 0x0000:
    case 0x1000:
    case 0x2000:
    case 0x3000:
      //Primer banco ROM
      return CROM[dir];

    case 0x4000:
    case 0x5000:
    case 0x6000:
    case 0x7000:
      //Segundo banco ROM (mapeado)
      return CROM[dir-0x4000+(ROMBankNumber*0x4000);

    case 0x8000:
    case 0x9000:
      //RAM de Video
      return VRAM[dir-0x8000+(VRAMBankNumber*0x2000)

    [....]

    }
}
```

Ilustración 4.7: Parte del código de la función readByte()

Las escrituras en memoria son realizadas por la funcione writeByte. Esta función es muy compleja porque las escrituras en diferentes zonas de memoria tienen "efectos colaterales". Por ejemplo, escribir en 0x2000-0x3FFF modifica el banco de ROM mapeado en 4000-7FFF y escribir en 0xFF46 inicia una transmisión DMA de 160 bytes a OAM.

```
writeByte(dirección, dato){

  Si dirección entre 0x0000 y 0x7FFF :
    //modificarRegistrosMBC : Bancos ROM y RAM mapeados

  Si dirección entre 0x8000-0x9FFF :
    //Escribir en banco de VRAM actual

  Si dirección entre 0xA000-0xBFFF :
    //Escribir en banco de CRAM actual

  Si dirección entre 0xC000-0xDFFF :
    //Escribir en banco de WRAM actual

  Si dirección entre 0xE000-0xFDFF :
    //Escribir en banco de WRAM actual (ECHO de 0xC000-0xDDFF)

  Si dirección entre 0xFE00-0xFE9F :
    //Escribir en OAM (zona de memoria principal)

  Si dirección entre 0xFF00-0xFF70 o igual a 0xFFFF :
    //Escribir en puerto

  En cualquier otro caso :
    //Escribir en dirección de memoria principal
```

Ilustración 4.8: Pseudocódigo de la función writebyte()

Los registros y la memoria de la GB almacena bytes sin signo, esto es, pueden tomar valores entre 0-255 (0-65535 en registros de 16 bits). Todas los tipos básicos de Java son con signo por lo que un byte toma valores entre -128 y 127. De usarse bytes para la memoria y registros de 8 bits, algunas operaciones tan sencillas como una suma o una resta se complicarían enormemente. Por ésta y otras razones se han usado variables de más de 8 bits para la memoria y registros:

- Permiten almacenar valores de 8 bits sin signo (0-255).
- Facilitan el cálculo de bit CY (overflow): en 0-7 está el valor de 8 bits y en el bit 8 el posible overflow

Sin embargo, las variables de más de 8 bits no tienen efecto wrap de forma que el valor que contengan esté siempre entre 0-255 por lo que éste ha de hacerse manualmente tras cada cálculo con una operación de and con 0xFF para valores de 8 bits o con 0xFFFF para 16 bits.

Visión general

A modo de vísta panorámica, se presenta el conjunto de funciones ofrecidas por la clase CPU :

CPU
startCPU() : void
stopCPU() : void
run() : void
getCGBEnable() : boolean
getTitulo() : String
getVideo() : Video
<<create>> CPU(KEYS_PRESSED : boolean[],ic : ImageControl,imagen : Image,ff : FormFrame,g : Ginage)
getR8L(reg16 : int) : int
getR8H(reg16 : int) : int
getR16(reg8H : int,reg8L : int) : int
readNextSignedByte() : int
readNextWord() : int
readByte(dir : int) : int
writeByte(dir : int,val : int) : void
reset() : void
cargarROM(fileName : String) : void
cargarRAM() : void
guardarRAM() : void
ADC(val : int) : void
ADD(val : int) : void
ADD16HL(val : int) : void
ADDSP(val : int) : int
AND(val : int) : void
BIT(bit : int,val : int) : void
CALL() : void
CCF() : void
CP(val : int) : void
CPL() : void
DAA() : void
DEC(val : int) : int
DEC16(val : int) : int
INC(val : int) : int
INC16(val : int) : int
JP() : void
JR() : void
OR(val : int) : void
POP() : int
PUSH(val : int) : void
RES(bit : int,val : int) : int
RET() : void
RL(val : int) : int
RLA() : void
RLC(val : int) : int
RLCA() : void
RR(val : int) : int
RRA() : void
RRC(val : int) : int
RRCA() : void
RST(dir : int) : void
SBC(val : int) : void
SCF() : void
SET(bit : int,val : int) : int
SLA(val : int) : int
SRA(val : int) : int
SRL(val : int) : int
SUB(val : int) : void
SWAP(val : int) : int
XOR(val : int) : void
internalTiming(cicles : int) : void
iteracion() : void
createDAATable() : void
createSwapTable() : void

Ilustración 4.9: Funciones de la clase CPU

4.3.3 La clase Video

Tareas principales

Esta clase es la encargada de gestionar todo lo relativo al cálculo de los gráficos que aparecen por pantalla. Realiza un cálculo de los gráficos linea a linea, esto es, tras el número de ciclos de CPU que se tarda en pasar por los modos 2,3 y 0, calcula la linea actual dibujada por pantalla (valor del registro LY) mediante la función calcularLineaComun(), concretamente en el instante en que la clase CPU cambia el modo LCD a 3. Cuando la CPU cambia el modo LCD a 1 (V-Blank), Video pinta el contenido por pantalla mediante al función pintarPantalla().

Esta clase por tanto necesita acceso a la memoria principal pues accede a la memoria OAM, la memoria VRAM, las paletas y los registros LCDC, SCX, SCY, WX, WY entre otros por lo que mantiene una referencia a la misma.

A continuación se expone el funcionamiento de la función pintarPantalla()

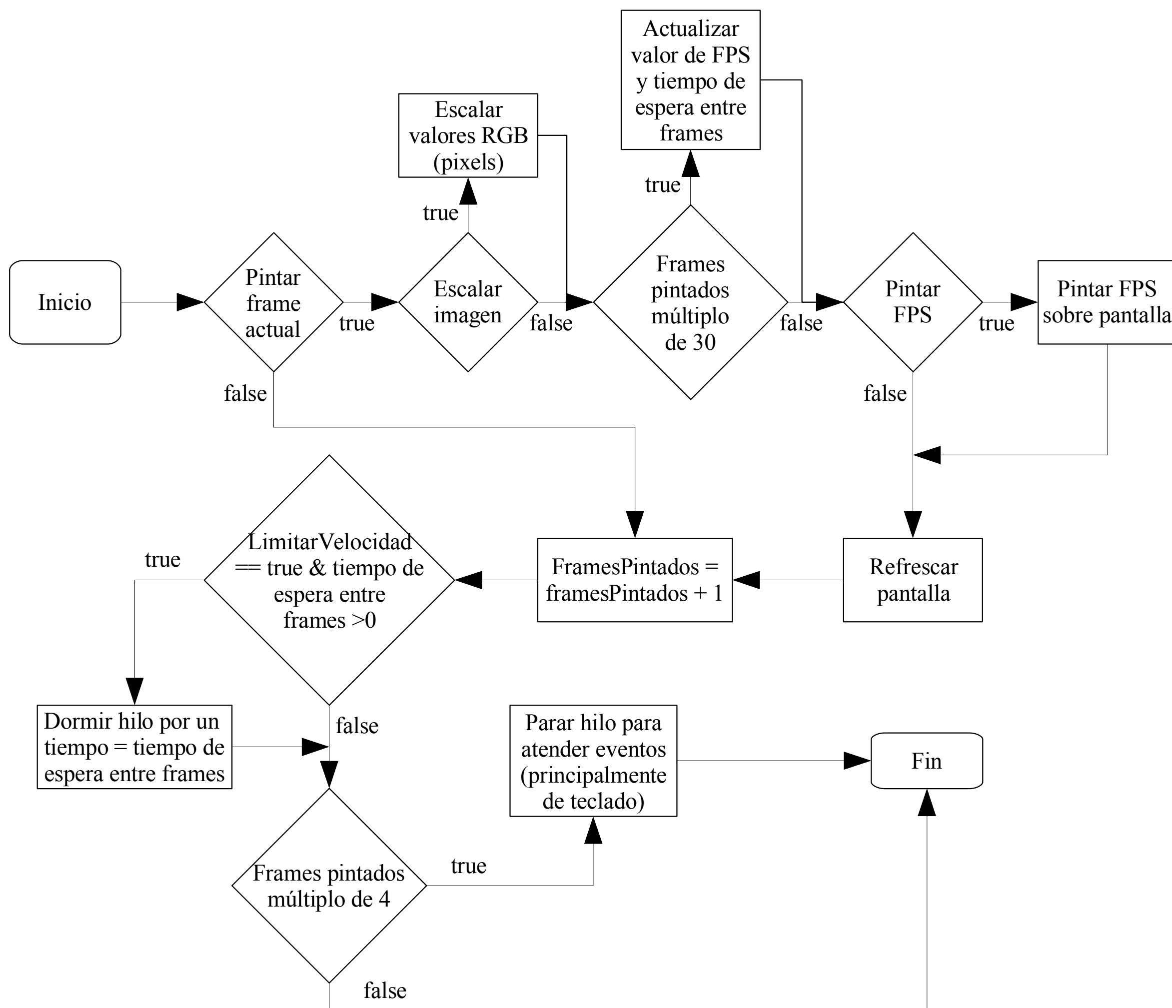


Ilustración 4.10: Diagrama de funcionamiento del método pintarPantalla()

Esta función no solo tiene la finalidad obvia de pintar el contenido de la pantalla, sino que también es la encargada de la sincronización del emulador (timing externo) y de dormir el hilo principal periódicamente para atender los eventos (principalmente de teclado).

Lo primero que comprueba es si el frame actual ha de ser pintado o no. Dependiendo del valor de frameskip, algunos frames no son calculados ni dibujados, ahorrando cálculos y mejorando la velocidad. Observese que aunque no se pinte el frame actual, la función sigue cumpliendo su cometido de timing externo y paradas para atención a eventos.

- Si el frame actual ha de pintarse se comprueba lo siguiente:
    - Si está selecionado el escalado 2x desde la interfaz, se hace el escalado de de los pixels.
    - Cada 30 frames se actualiza el valor del porcentaje de los frames por segundo consumidos (el valor mostrado en pantalla de habilitarse la opción en el menú) además de calcularse el posible tiempo que permanecerá dormido el hilo para ajustar el timing externo (limitar la velocidad del emulador en ordenadores rápidos).
    - Si ha de pintarse el valor de los FPS, se pinta
    - Se actualiza la imagen de pantalla con el contenido de los valores RGB del arry de pixels (el pintado propiamente dicho).

- El número de frames pintados se incrementa en 1 (aún no habiéndose pintado realmente por pantalla pues éste valor se utiliza para los cálculos de timing externo y parada del hilo para eventos).

- Si la opción de la interfaz "limitar velocidad " es true y el tiempo de espera es > 0, dormir el hilo principal por un tiempo = tiempo de espera.

- Si número de frames pintados es múltiplo de 4: dormir el hilo principal para que se puedan atender los eventos.

La función calcularLineaComun() realiza el siguiente pseudocódigo:

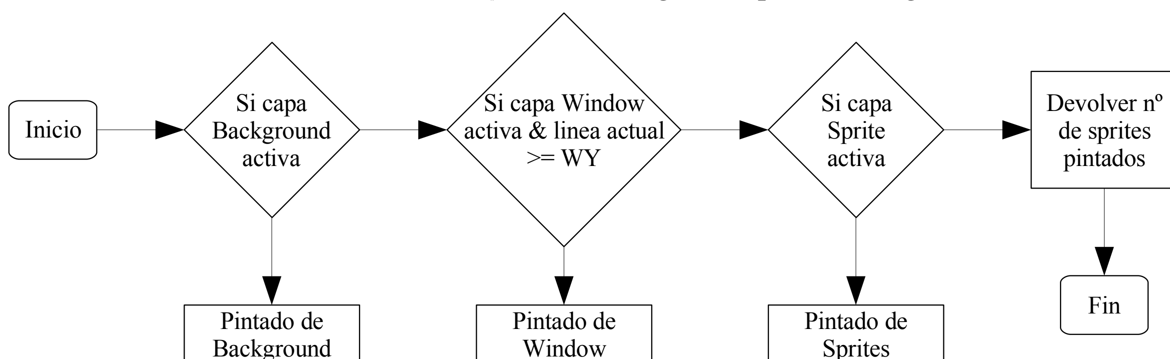


Ilustración 4.11: Diagrama de funcionamiento del método calcularLineaComun()

El funcionamiento la clase Vídeo consiste en calcular las 144 lineas mostradas por pantalla llamando a esta función y finalmente llamar a pintarPantalla() para refrescarla. Se ha expuesto primero el funcionamiento de pintarPantalla() únicamente por ser éste más sencillo.

Siguiendo el diagrama previo, para la linea actual pintada (LY), se pintan en orden cada una de las 3 capas si éstas se encuentran activadas en el registro LCDC. Finalmente la función devuelve el número de sprites pintados en la línea actual para propósitos de timing (el modo 3 del LCD, desde donde se llama a esta función, tiene una duración proporcional al mismo). A continuación se verá más a fondo el pintado de cada capa comenzando por la Background:

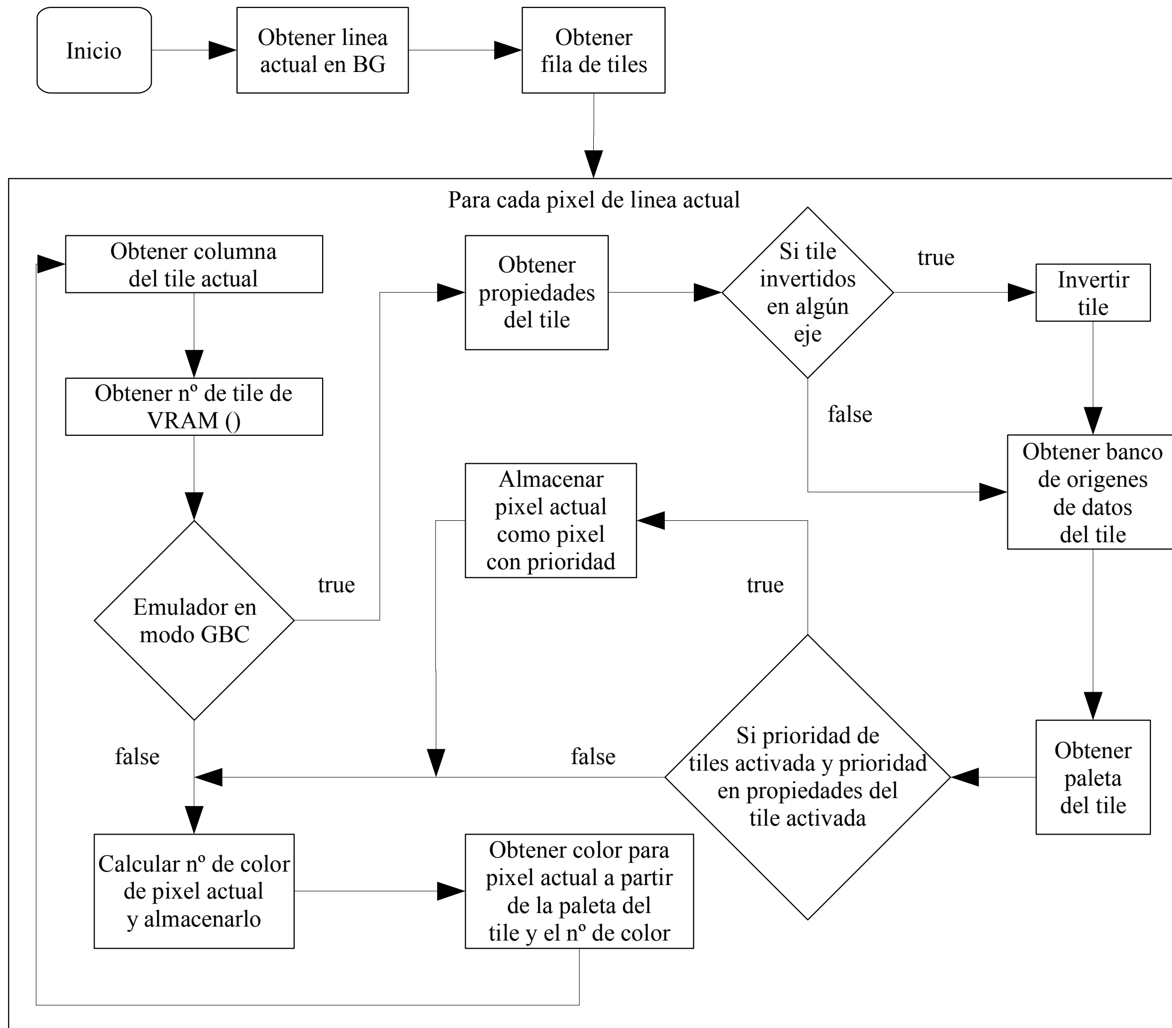


Ilustración 4.12: Diagrama de funcionamiento del pintado de la capa Background

Lo primero consisten en obtener la linea actual de BG, esto es, la linea en el array de 32x32 tiles que forman la verdadera pantalla (256x256pixels) también conocida como mapa de tiles (ver [ilustración 3.14]) a partir de la linea pintada por la pantalla de 160x144 pixels. Para ello se suman LY (linea actual en pantalla) con SCY (linea en mapa de tiles donde comienza BG) y como el resultado tiene efecto wrap (ver [ilustración 3.7]) se le hace el módulo con 256.

La fila de tiles es la fila del mapa de tiles donde están todos los tiles "atravesados" por la linea actual de BG. Como cada tile tiene 8x8 pixels, la división de la anterior cifra por 8 da el número de fila de tiles.

Dentro del bucle de 160 pixels del ancho de una linea se calcula en cada iteración:

- La columna del tile actual: Se calcula como (SCX + x)/3 siendo x el pixel actual [0-160] y SCX la columna en el mapa de tiles donde comienza BG.
- Número de tile actual de VRAM: Los números de tile se encuentran en el mapa de tiles de BG que puede ser 9800-9BFF o 9C00-9FFF dependiendo del valor de LCDC. Como cada fila en el mapa de tiles tiene 32 tiles, el número de tile se calcula como VRAM[BGTileMapPointer + (yTile * 32) + xTile], siendo BGTileMapPointer la dirección origen del mapa de tiles (800 o C00), yTile la fila del tile actual y xTile la columna del tile actual.
- En modo GBC existen 2 bancos de RAM por lo que:
  - Además del mapa de tiles están las propiedades del tile en la misma posición del número de tile pero en el banco 2. Entre la información del mismo se encuentran:
    - yFlip y xFlip: Indican si el tile está invertido en algún eje. De estar invertido en ambos ejes, el pixel 0,0 pasaría a ser el 7,7 por lo que para invertir las coordenadas basta con restarlas de 7.
    - Banco de origen de datos: el primero o el segundo
    - Paleta del tile: La GBC tiene 8 paletas de BG por lo que es necesario obtener cual es la aplicada al tile actual.
    - Si la prioridad para tiles está activa en LCDC y la prioridad en las propiedades del tile también, se marca el pixel actual como prioritario de manera que tendrá preferencia ante un sprite (el pixel de esa posición en la capa sprites no se pintará).
- El número de color: Se obtiene mediante dos accesos a bytes consecutivos de la zona de datos 8000-8FFF o 8800-97FF dependiendo del valor de LCDC (ver ilustración 3.13). El número de color del pixel actual se almacena pues es necesario para saber si se han de pintar o no los pixels de sprites como ya se verá .
- El valor RGB del color: Obtenido como BGPal[(paleta*4)+color] siendo paleta el número de paleta y color el número de color. Es almacenado en la screenRGB[(LY*160) + x] donde LY es la linea actual y x el pixel actualmente calculado.
- Incrementar la columna del tile actual y realizar otra iteración para el siguiente pixel si x<160: La columna del tile actual también tiene efecto wrap por lo que se le hace modulo con 256.

La capa Window es descrita a continuación:

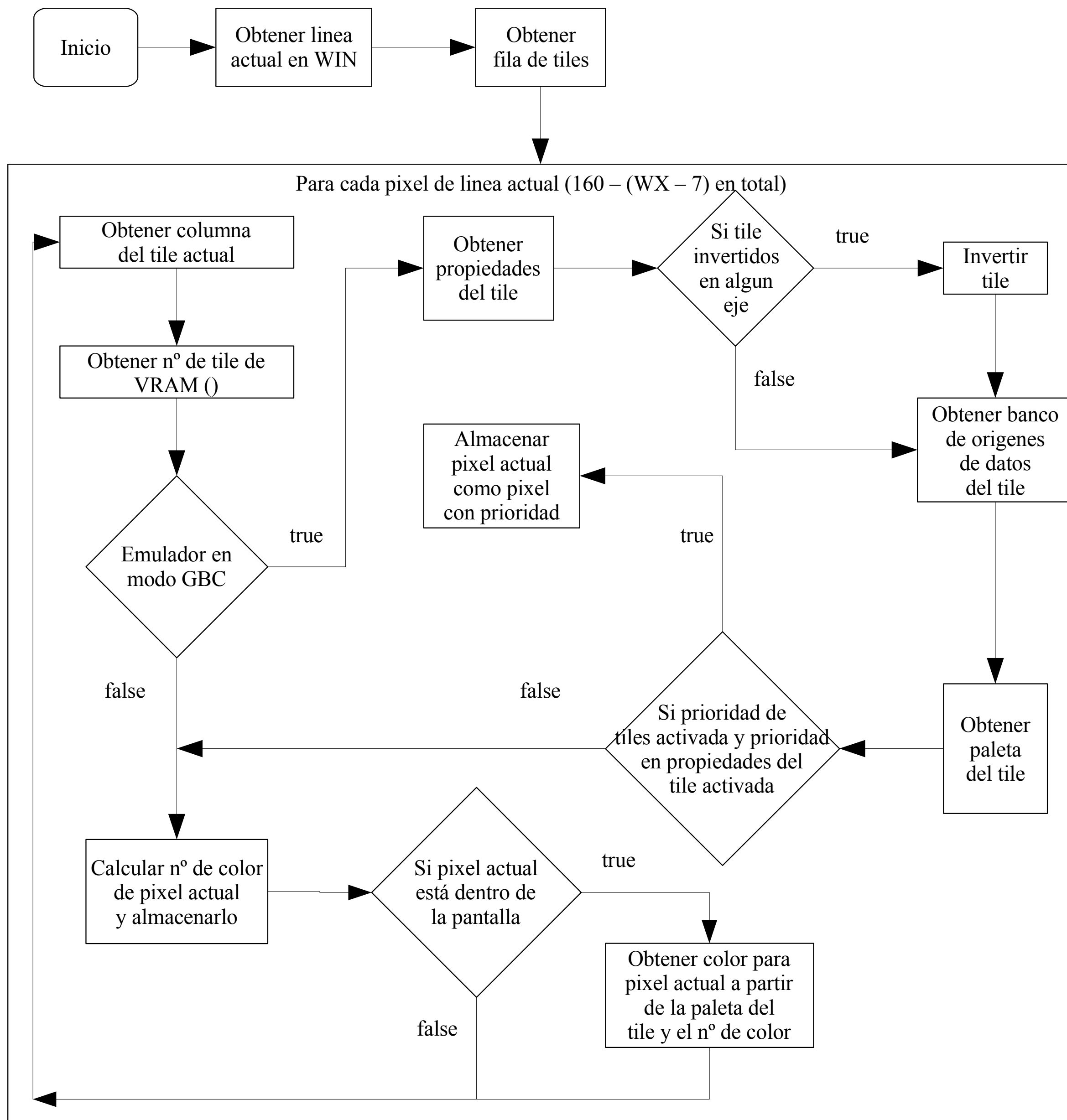


Ilustración 4.13: Diagrama de funcionamiento del pintado de la capa Window

La capa Window sólo se pinta si LY >= WY, esto es, la linea actual que está siendo calculada es posterior a la linea origen de WIN. Como puede verse, es muy similar a la anterior por lo que se describirán las partes que difieran:

- Obtener linea actual en Window: La capa Window se pinta de forma relativa a la posición de BG. Los registros WX e WY contienen sus coordenadas relativas a BG (o si se prefiere, al comienzo de la pantalla visible de 160x144 pixels). Por lo tanto la linea actual en Window es WY.

- Obtener fila de tiles: A diferencia de la capa BG, la capa WIN es relativa a la posición de BG por lo que la fila de tiles se obtiene como (LY – WY)/8 siendo LY la linea actual de pantalla pintada y WY la linea donde empieza WIN. Es decir, el número de lineas desde WY hasta LY dividido por la altura en lineas de un tile.

- El resto del procesamiento es idéntico al de la capa Background salvo ésta capa no tiene efecto wrap y por lo tanto hay que asegurarse que el pixel que calculado está entre entre 0 y 160 en el eje x y de ser así, se calcula el color RGB.

La capa sprite es la última pintada, sobrescribiendo los colores RGB de los tiles a los que cubran los sprites, salvo que esos tiles tengan prioridad.

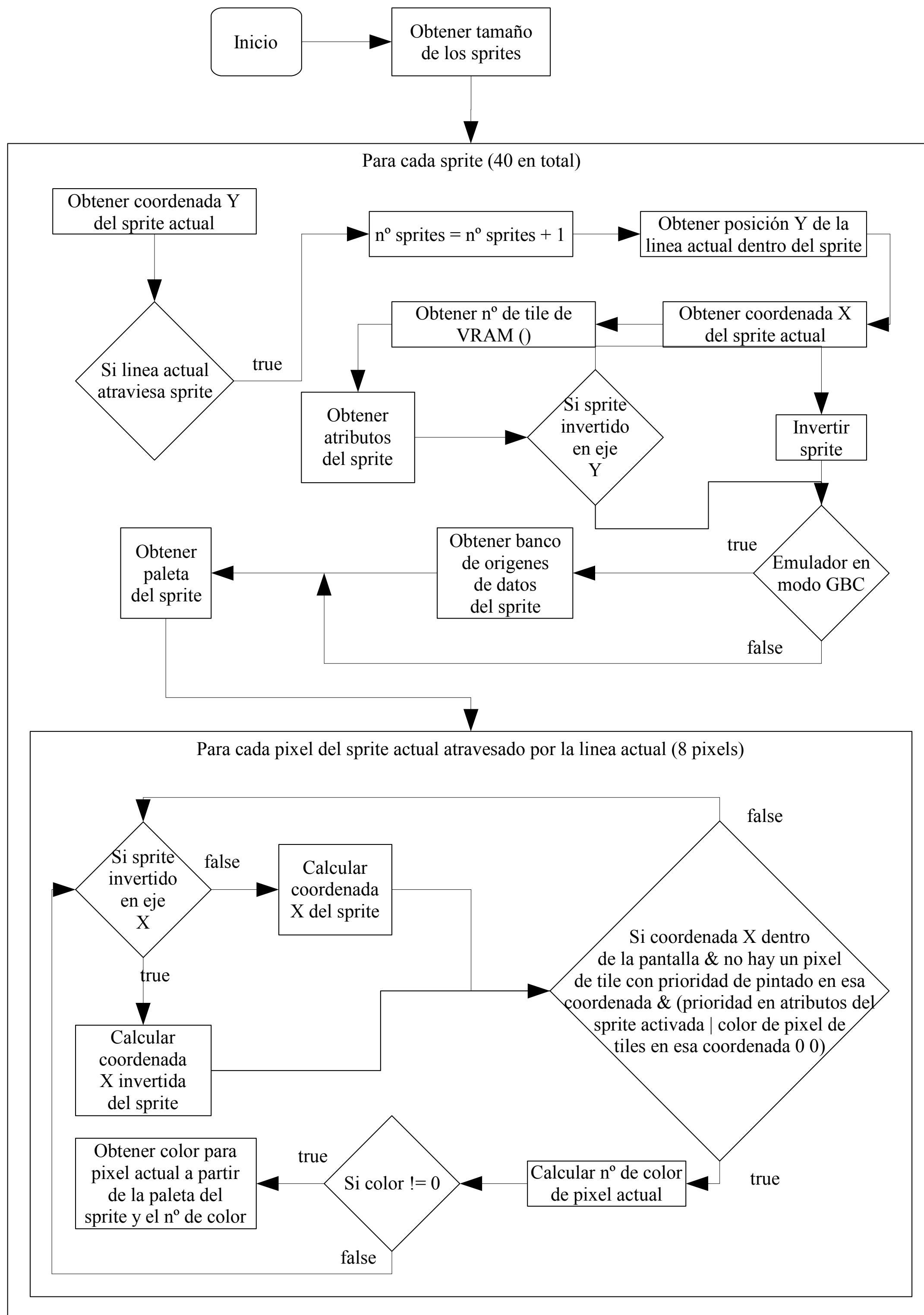


Ilustración 4.14: Diagrama de funcionamiento del pintado de la capa Sprite

Los sprites son pintados desde el ultimo al primero pues en caso de solapamiento se pinta el más prioritario (prioridad de mayor a menor: 0 ... 39). En modo GB, la prioridad es mayor cuanto menor sea la coordenada x del sprite. Pintar los sprites de esta manera aumentaría la complejidad de

los calculos por lo que para la emulación, simplificaremos considerando que los sprites son pintados siempre de la forma antes citada.

Las funcionamiento del pintado de la capa sprite es el siguiente:

- Obtener el tamaño de los sprites: 8x8 o 8x16. Ésto es importante si el sprite está invertido en el eje Y.
- El siguiente proceso se repite para los 40 sprites en memoria:
  - Obtener coordenada Y del sprite actual: Como las propiedades de cada sprite en OAM ocupan 4 bytes, se obtiene la coordenada Y del sprite actual accediendo a GBMem[(0xFE00 + currentSprite)]-16 siendo currentSprite = número sprite actual *4; Al resultado obtenido hay que restarle 16 porque la GB almacena este valor con ese offset.
  - Si la linea actual atraviesa el sprite actual pintarlo, sino siguiente sprite. Es decir, si la linea actual está entre la coordenada Y del sprite y la coordenada Y del sprite + el tamaño del sprite.
    - Incrementar el número de sprites pintados. La duración del modo de LCD 3 es directamente proporcional al número de sprites pintados.
    - Calcular la posicion Y de la linea actual dentro del sprite. Como los sprites tienen una altura de 8 o 16 pixels, el número de linea dentro del sprite se calcula como: spriteY = LY – coordenada Y del sprite actual;
    - Obtener coordenada X del sprite actual: Se almancena en el segundo byte de propiedades del sprite por lo que se obtiene accediendo a GBMem[(0xFE00 + 1 + currentSprite)]-8 siendo currentSprite = número sprite actual *4; Al resultado obtenido hay que restarle 8 porque la GB almacena este valor con ese offset.
    - Obtener el número de tile asociado al sprite actual: Es el tercer byte. Se obtiene accediendo a GBMem[(0xFE00 + 2 + currentSprite)].
    - Obtener los atributos del sprite actual: Es el cuarto byte. Se obtiene accediendo a GBMem[(0xFE00 + 3 + currentSprite)].
    - Si yFlip activo en atributos del sprite, la posición Y de la linea actual dentro del sprite cambia, calculándose como: spriteY=spriteSize – spriteY;
    - Si emulador en modo GBC: Obtener banco de origenes de datos del sprite actual (primer o segundo banco).
    - Obtener paleta del sprite actual: En modo GBC la paleta puede tomar valores entre 0-7 y en GB entre 0-1.
    - Para cada uno de los 8 pixels de la linea actual del sprite actual:
      - Si xFlip activo en atributos del sprite, el pixel actual de la linea actual dentro del sprite para una de las 8 iteraciónes cambia, calculándose como: x = spriteX + (7-x). Si no está activo x = spriteX+x.

        - Si pixel actual está dentro de la pantalla (0-160) y no hay un pixel de tile con prioridad de pintado en esa misma coordenada (los pixels de la capa BG y WIN no tiene prioriridad sobre los de SPRITE) y [prioridad de sprite en atributos del sprite activada (el sprite tiene su prioridad normal: encima de BG y WIN) o número de color de pixel de tiles en esa coordenada es 0 (sprite está por encima de BG y WIN cuando éstos tienen número de color 0)]. Si se dan estás condiciones:
            - Calculo del número de color: Se obtiene mediante dos accesos a bytes consecutivos de la zona de datos 8000-8FFF.
            - Si el número de color es distinto de 0 (El color 0 es transparente en los sprites por lo que no se pinta):
                - Obtener valor RGB del color a patir den número de color y el número de paleta: Se obtiene como SpritePal[(paleta*4)+color] siendo paleta el número de paleta y color el número de color. Es almacenado en la screenRGB[(LY*160) + x] donde LY es la linea actual y x el pixel actualmente calculado.
        - Incrementar pixel de la linea actual del sprite actual (0-8) repitiendo el proceso.
    - Repetir el proceso completo para los 40 sprites.

Emulación de atributos principales

A continuación se pasa a describir cómo se han emulado las variables más importantes de la misma:

- Valores RGB de los pixels de pantalla (variable screenRGB): La pantalla de la GB tiene 160x144 pixels por lo que se ha implementado como un array unidimensional de ints de 160x144 posiciones. Al ser unidimensional es más eficiente acceder a cualquier posición del mismo pues solo requiere una suma (dirección base del array + posición a la que se quiere acceder) y un acceso a memoria (acceso a la posición de memoria calculada).
  Además, el objeto Image de Ewe admite arrays unidimensionales con valores RGB por lo que esta representación es ideal.

- Paletas (BGPal y SpritePal): Se mantienen 8 paletas para tiles de Background y Window y otras 8 para sprites consistentes en sendos array unidimensionales de 8x4 ints (8 paletas por 4 colores cada una). En el modo de emulación GB (no color), sólo se usan 3 paletas: 1 para tiles y 2 para sprites. Ha de destacarse que las paletas siempre contienen valores RGB de 32 bits y son gestionadas por métodos diferentes dependiendo del funcionamiento del emulador en modo GBC o no. En modo GB se usan los métodos:

- setDMGBackgroundPalette(): Las escrituras en el puerto BGP en modo GB modifican los colores apuntados por la paleta Background.
- setDMGObjectPalette(int paleta): Las escrituras en el puerto OBP0 y 1 en modo GB modifican los colores apuntados por alguna de las 2 paletas de Sprites.

En modo GBC se usan:

- setCGBBackgroundPalette(int val): Las escrituras en el puerto BGPD en modo GBC modifican los bytes de color RGB (2 bytes por color) de alguna de las 8 paletas de Background.
- setCGBObjectPalette(int val): Las escrituras en el puerto OBPD en modo GBC modifican los bytes de color RGB (2 bytes por color) de alguna de las 8 paletas de Sprites.

Además de los arrays de paletas con los valores RGB, existen otros arrays como:

- backgroundColorData y spriteColorData: Ambos son dos arrays unidimensionales de 8x4 ints (8 paletas x 4 colores cada una). Contienen los valores de color de las paletas en formato interno de la GB, esto es RGB de 15 bits (ver [ilustración 3.11]). Son usadas como buffers intermedios en el calculo de los valores RGB de 32 bits (ya que en modo GBC cada color requiere modificar 2 bytes)

- coloresBG, coloresFG0 y coloresFG1: Son 3 arrays de 4 colores cada uno y representan cada uno de los posibles colores de cada una de las paletas del modo GB (no color). Los juegos de GB tienen 4 tonos porque aunque poseen 3 paletas solo pueden hacer referencia a 4 tonos diferentes. Nada impide hacer referencia a 12 colores diferentes en lugar de 4, siendo además personalizables desde la interfaz de usuario.

Visión general

A modo de vísta panorámica, se presenta el conjunto de funciones ofrecidas por la clase Video:

```
Video
<<create>> Video(GBMem : int[],ic : ImageControl,imagen : Image,ff : FormFrame,g : Ginage)
setModoColor(modoColor : int) : void
setPintadoFPS(pintadoFPS : boolean) : void
setFrameskip(frameskip : int) : void
setLimitarVelocidad(limitarVelocidad : boolean) : void
setEscalado(factorEscalado : int) : void
setVRAM(VRAM : int[]) : void
setCGBEmulation(CGBEnable : boolean) : void
calcularFPS() : void
pintarFrame() : boolean
pintarPantalla() : void
calcularLineaComun() : int
calcularLineaDMG() : int
setVideoPointers() : void
getDMGBackgroundColors() : int[]
getDMGObjectColors(palette : int) : int[]
setDMGBackgroundColors(colors : int[]) : void
setDMGObjectColors(palette : int,colors : int[]) : void
setDMGBackgroundPalette() : void
setDMGObjectPalette(paleta : int) : void
setCGBBackgroundPalette(val : int) : void
setCGBObjectPalette(val : int) : void
resetCGBBackgroundPalette() : void
```

Ilustración 4.15: Funciones de la clase Video

4.4 Fases en el desarrollo (Algunas iteraciones del modelo)

A continuación se mostrarán algunos de los hitos conseguidos durante las sucesivas iteraciones que han compuesto la etapa de desarrollo de GINAGE. El lector interesado podrá seguir el desarrollo del emulador desde sus fases más tempranas hasta la versión final, comprobando su evolución. Los siguientes apartados están ordenados por orden cronológico y aparecen en el cd adjunto dentro de la carpeta /versiones_de_desarrollo numeradas según el número del su própio subpunto (por ejemplo el código referente al punto 4.4.2 aparece en el subdirectorio 2). Algunos puntos pueden englobar más de una versión. Si ese es el caso, cada versión tendrá su própio subdirectorio dentro del subdirectorio del punto. Dentro de cada subdirectorio final se encuentra un fichero leeme.txt que explica los cambios realizados respecto a la versión anterior. Así mismo, cada carpeta es autosuficiente y contiene todo lo imprescindible para ejecutar esa versión. A continuación se resumen los hitos más importantes:

4.4.1 Lectura de metadatos de una ROM (cabecera)

La primera iteración tiene por objetivo obtener los metadatos de la cabecera de una ROM, mostrarlos por pantalla e inicializar unas estructuras de memoria básicas. Posteriormente, esta función será modificada para realizar la carga de las ROMs del emulador. Por lo tanto, el procesamiento correcto de la cabecera es extremadamente importante pues de ello depende el

correcto funcionamiento del emulador. Los datos más importantes desde el punto de vista de la emulación son:

- MBC: Dependiendo del tipo, las escrituras en 0x0000-0x7FFF tendrán unos efectos u otros
- Tamaño de ROM: La identificación del correcto tamaño de la ROM es imprescindible para inicializar correctamente el array interno que lo contiene.
- Tamaño de RAM: Idéntico razonamiento al del punto anterior solo que la RAM puede existir o no.
- Checksum: El procesamiento del checksum permite identificar si se trata o no de una ROM de GB.

4.4.2 Generación del switch de procesamiento de instrucciones

El switch de la función iteración() se generó a partir de un programa en python que procesa un fichero txt con los opcodes. Este fichero se obtuvo copiando y pegando los opcodes del documento [Lloyd00], añadiendo algunos códigos de operación que faltaban.

4.4.3 Comparativa de pintado de imágenes de Ewe y Superwaba

Ante el desconocimiento de las capacidades gráficas de ambas librerias, se hizo un pequeño programa de prueba para comprobar el soporte a imágenes de ambas librerias. El programa, genera colores aleatorios y los coloca en la pantalla.

La prueba no llegó a probar la eficiencia de ambas librerias porque el soporte para pintado de imágenes con valores RGB en Superwaba es muy deficiente por lo que queda descartada con alternativa.

4.4.4 Implementación de readByte() writeByte() y reset()

Las funciones de acceso a memoria y la de reset

4.4.5 Pruebas de rendimiento de implemetación de instrucciones y Flags

Se probaron diferentes implementaciones de la instrucción ADD y CCF así como de los flags. Los flags aparentemente resultaron más eficientes implementados como variables booleanas independientes que como un int. Sin embargo, la mayoría de emuladores los implementan como int y por sencillez se ha hecho así. Tratar los flags como booleanos hubiera traido el problema de componer el registro F para los operaciones pop y push de AF. Estas operaciones de pila son bastante frecuentes como se verá en la próxima sección por lo que podría haber supuesto una pérdida de rendimiento.

4.4.6 Pruebas de accesos aleatorios a memoria

Para probar el correcto funcionamiento de la función readByte, se rellenaron los arrays de memoria y cada una de sus bancos con valores numéricos diferentes. De esta forma, se creo un programa que realiza accesos a posiciones aleatorias de memoria y cambiar el mapeo de los bancos de ROM, RAM del cartucho, VRAM y RAM interna, comprobando que los valores devueltos por cada lectura corresponden con el esperado para el valor de los bytes de ese banco

4.4.7 Emulación de la ROM homebrew Apocalipsis Now

La elección de esta ROM homebrew como la primera para ser emulada responde a la disponibilidad del código fuente en ensamblador y a la sencillez de la misma: Utiliza muy pocos opcodes, ni usa interrupciones o intercambios de banco.

Se implementó el código imprescindible para hacerla funcionar. Ésto incluye únicamente los opcodes usados y por supuesto ningún tipo de emulación gráfica.

La única salida que presenta este programa es por la linea de órdenes mediante el minidepurador integrado. Éste muestra el estado de los registros de la CPU (incluidos los flags) y algunos puertos importantes como TIMA, IF, IE, LY o el modo de LCD tras la introducción de cada comando.Los comandos aceptados son (<n> significa número entero):

- <n>: avanzar a la instrucción <n>
- + <n>: avanzar <n> instrucciones desde la actual
- enter: avanzar una instrucción

Ante la ausencia de contenido gráfico, se realizó una comparación con el emulador Javaboy para constatar que el valor de los registros era el esparado para cada instrucción.

4.4.8 El soporte de Video y E/S

En las sucesivas versiones se fué incorporando la salida de Video. Así, se empezo implementando la capa Background, después se emuló la Window y por último la Sprite. Después de ésto se empezó a implementar el soporte de los controles mediante teclado.

4.4.9 Soporte a más homebrew: Game Boy Demo, BigScroller y LandScape

El soporte de nuevas ROMs hombrew implicó la implementación de sus opcodes usados y el soporte a interrupciones. A partir de esta versión, en el fichero leeme se comentan los bugs descubiertos y corregidos en el código durante las pruebas y los números de lineas de código modificadas en cada clase.

4.4.10 La reescritura del pintado de la pantalla

El método pintarPantalla fue reescrito para evitar pintar la pantalla pixel a pixel mediante un objeto Graphics, cogiendo los valores RGB de un array bidimensional.

Se cambió completamente el funcionamiento interno de la clase, cambiando los arrays bidimensionales a unidimensionales y reescribiendo el pintado de capas. El objeto Graphics fue desechado por la lentidud de escribir pixel a pixel y en su lugar de utilizó el método setPixels de la clase Image que recibe un array completo de valores RGB.

4.4.11 La primera ROM comercial: "Castlevania 2 – Belmont's Revenge"

El soporte a los juegos comerciales comenzo con éste título. Se completaron muchos opcodes necesarios para su funcionamiento y por medio del depurador se arreglarón bugs en la implementación.

4.4.12 Varias ROMs comerciales funcionando y soporte a GBC

Tras las pruebas con varias ROMs de GB se empezaron a probar juegos de GBC y a corregir posibles errores en la emulación

4.4.13 Mejoras en Ewe

Se ha modificado la librería Java de Ewe por las siguientes razones:

- Ausencia de eventos KEY_RELEASE: Una de las principales razones por las que se eligió Ewe frente a Superwaba era la posibilidad del primero de generar diferentes eventos para el presionado y el despresionado de teclas. Aunque ésto era lo que aseguraba la documentación oficial, más tarde se comprobó que en la versión actual de la librería para Java (la única disponible actualmente en la web) los eventos KEY_RELEASE no se generaban. Tras investigar el funcionamiento de la misma, se corrigió el problema modificando las librerías.

- Modificación de tratamiento de eventos de teclado: Algunas teclas como espacio, enter y flechas eran tratadas por Ewe como cambiadoras de foco entre componentes y no propagaban sus eventos de presionado/despresionado. Se ha cambiado este comportamiento para permitir ser usadas como cualquier otra tecla.

- Ausencia de widget similar al Jspinner de Swing: Este objeto consiste un campo con dos pequeños botones para obtener el siguiente o anterior valor de una secuencia. Resulta particularmente útil en pantallas táctiles para aumentar o disminuir un valor. Se ha implementado la clase mSpinnerNumber que permite tener un objeto de este tipo con una

secuencia de valores enteros dentro de un rango definible.

Se puede encontrar información concreta de los cambios hechos sobre Ewe en el subdirectorio 13 *modificaciones de librería* Ewe dentro del directorio *código*.

4.4.14 La creación de la GUI

Después de conseguir que funcionaran un buen número de ROMs en el emulador, se comenzo a crear la interfaz de usuario de menús.

4.4.15 Mejoras

Tras obtener una GUI funcionando, se pasó a completar algunas funcionalidades pendientes y a desarrollar pequeñas mejoras.

Entre las primeras se implementó el salvado/carga de partidas, el selector de colores para paletas de GB (hasta 12 colores diferentes), el escalado de la pantalla a 2x o el modo de color a 24bits o 256 colores.

Entre las segundas destacan algunas mejoras más en la eficiencia.

4.4.16 La versión final

Corregidos numerosos bugs que aumentan la compatibilidad así como limpieza y puesta a punto del código de GINAGE.

4.5 Consideraciones para mejorar la eficiencia

Durante el desarrollo del emulador se han seguido unas pautas para mejorar la eficiencia del mismo. A continuación se resumen algunas de ellas:

- Modificadores ***final*** en las clases y ***final*** y ***private*** en los métodos más llamados: El compilador realiza optimizaciones extra en estos métodos/clases y trata de hacer inlining de esos métodos.

- variables **int**: Las más eficientes en arquitecturas de 32 bits (y muy eficientes en arquitecturas 64 bits de tipo **x86-64)**. Algunas CPUs trabajan más rápido cuando leen/escriben datos del tamaño base y alineados a direcciones múltiples del tamaño base. Además Java promociona las variables a int en operaciones aun no interviniendo un int como operando por lo que trabajar con ints siempre será más rápido. Se han usado variables int para los registros y arrays de ints para las memorias.

- Optimización de casos más frecuentes frente a los menos frecuentes: Por ejemplo, los registros que pueden ser direccionados como uno de 16 o 2 de 8 han sido implementados con una o dos variables dependiendo del número de operaciones que los usen de uno u otro modo. Así, se han implementado los registros A,B,C,D,E,F (8-bits) y HL,PC y SP (16-bits).
  Otro ejemplo se da con el acceso a memoria. El switch de la función readByte es simple y se evitan comparaciones para mejorar aún más su eficiencia. La función writeByte sin embargo es mucho más compleja debido en parte a la propia naturaleza de los “efectos colaterales” al escribir en ciertas posiciones de memoria, pero es llamada en varios órdenes de longitud menos por lo que no es tan importante.

- Precálculos en algunas instrucciones: Concretamente las instrucciones BCD y SWAP para 8 bits usan sendas tablas en memoria.

- Uso de operadores << y >>> (desplazamientos lógicos) y &(n-1) para módulos de n siendo n potencia de 2: Para evitar multiplicaciones, instrucciones if (que implican instrucciones máquina de saltos que son muy lentas en arquitecturas x86) y módulos. Ya que el uso de estos operadores puede empeorar la legibilidad del código, en los casos donde se han usado, la linea inmediatamente anterior contiene un comentario cuyo primer carácter es # con el código equivalente más legible.
  Por lo general, se han centrado los esfuerzos en las funciones que implementan instrucciones del núcleo de la CPU y las que están en el bucle más interno por ser las más llamadas. En otras funciones, como reset de la CPU o carga de la ROM, no importa tanto su eficiencia por lo que he optado por la legibilidad directamente.

4.6 Software utilizado durante el desarrollo

Todo el software usado durante el desarrollo de este proyecto ha sido software libre. Se usó:

- Sistema Debian GNU/Linux
- Editor de texto kate
- Eclipse durante las pruebas de profiling
- SDK de Java de Sun y librería Ewe
- Umbrello y ArgoUML para los diagramas UML
- OpenOffice para el desarrollo de esta documentación y los dibujos

Las pruebas de rendimiento han sido realizadas por colaboradores voluntarios en sus máquinas particulares. De ahí que la mayoría de las mismas sean sobre windows.

Capítulo 5. RESULTADOS

5.1 ROMs comerciales soportadas

Durante el desarrollo se han probado diferentes ROMs homebrew, así como comerciales y aunque la compatibilidad alcanzada no es total, es bastante alta sobre todo con juegos de GB. A continuación se detalla una tabla con los juegos probados de GB y GBC durante el desarrollo. A partir de la misma se ha extrapolado el porcentaje de compatibilidad de GINAGE.

Título	*Sistema*	*Funcionamiento*	*Comentarios*
Castlevania 2 Belmont's Revenge	GB	OK	
Aladdin	GB	OK	
Alleyway	GB	NO	Paleta no se mueve y el juego empieza pausado
Final Fantasy I	GB	OK	
Megaman	GB	OK	
Donkey Kong Land	GB	OK	
Addams Family	GB	OK	
Legend of Prince Valiant	GB	OK	
Pokemon Blue Version	GB	OK	
Super Mario Land	GB	OK	
Tetris	GB	OK	
Super Mario Land 2	GB	NO	Juego bloqueado tras pantalla de inicio
The Legend Of Zelda: Link's Awakening	GB	OK	
Battletoads	GB	OK	
Double Dragon	GB	OK	
Galaga & Galaxian	GB	OK	
Gargoyle's Quest – Ghost'n Goblins	GB	NO	Fallos gráficos. Al salir de la primera habitación se bloquea.
Gauntlet II	GB	OK	
Ghosts 'N Goblins	GB	OK	
Motocross Maniacs	GB	OK	
Who Framed Roger Rabbit	GB	OK	
World Cup USA 94	GB	OK	
Contra: The Alien Wars	GB	OK	
Darwing Duck	GB	OK	
Dragon Ball Z	GB	OK	
Earthworm Jim	GB	OK	
F-15 Strike Eagle	GB	OK	

Título	Sistema	Funcionamiento	Comentarios
FIFA Soccer '98 – Road to the World Cup	GB	OK	
The Flintstones	GB	OK	
Indiana Jones and the Last Crusade	GB	OK	
International Superstar Soccer	GB	OK	
Jurassic Park	GB	??	Tamaño de ROM incorrecto. ¿ROM mal dumpeada?
Killer Instinct	GB	OK	
Kirby's Dream Land 2	GB	NO	No arranca
Lemmings	GB	OK	
Looney Tunes	GB	OK	
Metroid 2	GB	OK	
Mortal Kombat	GB	NO	Se queda parado tras la intro
Nintendo World Cup	GB	OK	
Pac-Man	GB	OK	
Parodius	GB	OK	
Prince of Persia	GB	OK	
Samurai Shodow	GB	OK	
The Simpsons – Escape from Camp Deadly	GB	OK	
Teenage Mutant Hero Turtles – Fall of the Foot Clan	GB	OK	
Tintin in Tibet	GB	NO	Se queda parado en la primera pantalla
Yoshi's Cookie	GB	OK	
Pinball Fantasies	GB	OK	
V-Rally – Championship Edition	GB	OK	
NBA Jam '99	GBC	OK	Pequeños problemas de sprites pero perfectamente jugable
Mario Golf	GBC	NO	Gráficos corruptos
Tetris DX	GBC	OK	
The Legend Of Zelda: Link's Awakening DX	GBC	OK	
Harvest Moon	GBC	OK	
Wario Land	GBC	NO	Problema de capas y sprites invisibles.
Quest for Camelot	GBC	OK	
Pokemon Pinball	GBC	NO	No llega a arrancar (emulador parado)
Super Mario Bros Deluxe	GBC	NO	Problema con sprites.
1942	GBC	OK	
R-Type DX	GBC	OK	

Título	*Sistema*	*Funcionamiento*	*Comentarios*
Alone in the Dark – The New Nightmare	GBC	NO	Problemas gráficos
Formula One 2000	GBC	OK	
Heroes of Might and Magic	GBC	NO	Problemas de capas
Spider-Man 2 – The Sinister Six	GBC	NO	Capa background invisible
Xtreme Wheels	GBC	OK	
Yu-Gi-Oh! - Dark Duel Stories	GBC	OK	Algún pequeño problema gráfico pero jugable
Batman Beyond – Return of the Joker	GBC	NO	Problemas de capas
Bomberman Max – Red Challenger	GBC	OK	
Bomberman Quest	GBC	OK	
Donkey Kong Country	GBC	NO	Emulador se cuelga
Dragon's Lair	GBC	NO	Algún problema con capas y funcionamiento incorrecto
Dragon Ball Z – Legendary Superwarriors	GBC	NO	Gráficos corruptos
Duke Nukem	GBC	NO	Sprites corruptos
Earthworm Jim – Menace 2 the Galaxy	GBC	??	¿ROM mal dumpeada?
Frogger	GBC	OK	
Grand Theft Auto 2	GBC	OK	
Harry Potter and The Sorcerer's Stone	GBC	NO	Pantalla en blanco
Indiana Jones an the Infernal Machine	GBC	NO	Sprites no se muestran y BG en intro se ve rosa
Internation Superstar Soccer '99	GBC	OK	
Lufia – The Legend Returns	GBC	NO	Emulador bloqueado
The Legend of Zelda – Oracle of Ages	GBC	NO	Pantalla en blanco
Mario Tennis	GBC	NO	BG incorrecto
Mega Man Xtreme	GBC	OK	Problema con la intro y escenas cinemáticas pero el juego es perfectamente jugable
Mortal Kombat 4	GBC	OK	
Paperboy	GBC	OK	
Perfect Dark	GBC	NO	Gráficos corruptos
Pokemon - Gold Version	GBC	OK	
Space Invaders	GBC	OK	
Star Wars Episode I – Obi-Wan's Adventures	GBC	NO	Pantalla en negro
Street Fighter Alpha – Warrior's Dreams	GBC	NO	Sprites corruptos

Título	Sistema	Funcionamiento	Comentarios
Tomb Raider	GBC	NO	Gráficos corruptos
Turok 3 – Shadow of Oblivion	GBC	NO	Pantalla en negro
X-Men: Mutant Academy	GB	NO	Sprites corruptos

Tabla 5.1: Títulos soportados por GINAGE

Sistema	Titulos probados	Funcionan	Estimación de compatibilidad
Game Boy	49	42	86%
Game Boy Color	44	20	45%

Tabla 5.2: Porcentajes de compatibilidad

Muchos de los títulos de GBC que no funcionan lo hacen parcialmente, es decir, presentan problemas gráficos que los hacen injugables. Aunque habría que depurar cada ROM problemática una a una para saber donde está el fallo se pueden intuir algunas pautas de donde puede estar el problema. Al ser las ROMS de GBC las que presentan la tasa de compatibilidad más baja, el problema podría estar en alguna parte de la emulación específica de GBC como el MBC5 o la gestión de los bancos de VRAM o RAM interna. Los esfuerzos de depuración deberían centrarse en primer lugar en estos puntos para después hacer una depuración minuciosa de las ROMs una a una.

No obstante, la compatibilidad inicial obtenida, sobre todo con GB, es bastante alta.

5.2 Comparativa de eficiencia en diferentes arquitecturas

GINAGE está basado en Ewe y como tal puede funcionar en varias plataformas:

- Sobre una MV java: En todas las máquinas donde exista una versión de la MV compatible con java 1.2 o superior.
- Sobre una MV de Ewe, nativa a cada plataforma: Ésta es la única forma de ejecutar la aplicación en dispositivos móviles como Pocket PC. También existen MV nativas para PC (Windows y GNU/Linux).

Las pruebas a continuación presentadas se han realizado en diferentes Pcs con la MV de java. Sobre una MV nativa el emulador funciona a unos 3 frames por segundo por lo que con las MV nativas y PDAs actuales, el emulador no es usable.

5.2.1 Organización de las pruebas

Las pruebas consisten en calcular los FPS y el tiempo que tarda la máquina emuladora en calcular cada frame. Se ha probado el emulador en los siguientes PCs:

Sistema	*CPU*	*Cantidad de RAM*	*Sistema Operativo*	*SDK java*
1 (curro)	Pentium IV 3Ghz	512MB	Windows XP Professional SP2	5
2 (casa)	AMD Athlon XP 2400+ (2Ghz)	512MB	Windows XP Professional SP2	5
3 (pinis)	Pentium IV 3Ghz	512MB	Windows XP Professional SP2	5
4 (focus)	Intel Core 2 Duo 2,13 Ghz	2GB	Windows XP Professional SP2	5
5 (josean)	AMD Sempron 2400+ 1.67GHz	1GB	Windows XP Professional SP2	6
6 (juancho)	AMD Athlon XP 2600+ (2083Mhz)	512MB	Windows XP Professional SP2	5

Tabla 5.3: Sistemas usados en las pruebas

En cada PC se han usado diferentes versiones del SDK de java, compilando el código sin optimizaciones para cada arquitectura diferente, y cada prueba ha consistido en emular la ROM del juego “Castlevania 2 – Belmonts Revenge” en todos los sistemas, dejando el emulador en ejecución mostrando la intro del juego hasta volver a la pantalla de título por segunda vez. Éste tiempo es suficiente como para obtener una media de FPS realista. Durante las pruebas el emulador se encontraba en en reposo, esto es, sin pulsar ninguna tecla. Cada sistema se ha probado con todas las combinaciones de tamaño y modo de color de pantalla: tamaño normal (1x) y color real (24bits), tamaño doble(2x) y color real (24bits), tamaño normal (1x) y color indexado (256colores), tamaño doble(2x) y color indexado (256colores).

5.2.2 Diferencias entre arquitecturas

Para el SDK java 5 (máquinas 1-4 y 6) se han obtenido los siguientes resultados:

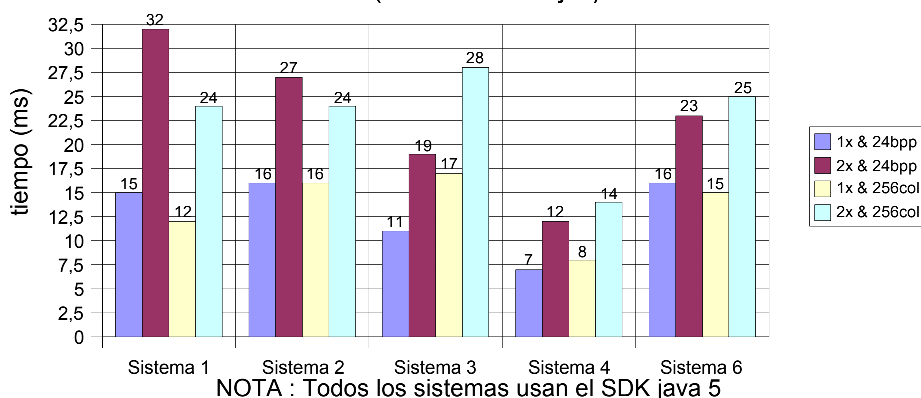


Ilustración 5.1: Comparativa de rendimiento de sistemas para SDK Java 5 de SUN

Se puede observar que los sistemas 1 y 3, aún teniendo en común el procesador , cantidad de RAM y sistema operativo, presentan resultados muy diferentes: Por un lado, los mejores tiempos para el modo 24bpp los da el sistema 3, presentando una mejora respecto al 1 del 36% para tamaño 1x y del 68% para 2x. Sin embargo en las pruebas a 256 colores, el sistema 1 es un 42% más rápido para tamaño 1x y un 17% para 2x.

Por si fuera poco, el sistema 1 con 256 colores es más rápido que con 24bpp, concretamente un 25% más rápido en 1x y un 33% en 2x mientras que el sistema 3 es más rápido con 24bpp, un 55% en 1x y un 47% en 2x.

El modo 1x es más rápido que el modo 2x siempre: En el sistema 1 con 24bpp es un 113% más rápido y con 256 colores un 100% más rápido (el doble). En el sistema 2 con 24bpp es un 73% más rápido y con 256 colores un 65% más rápido.

A la luz de éstos datos se puede sacar varias conclusiones, al menos para sistemas Windows:

- Incluso entre sistemas parecidos los resultados pueden ser radicalmente diferentes: Es cierto que en la medición de tiempo influyen muchos más factores que la velocidad del procesador, cantidad de RAM y sistema operativo. Los aquí escogidos son una simplificación, necesarios por otra parte para poder repartir un documento de pruebas “sencillo” entre varios betatesters.

- Las arquitecturas de múltiples núcleos son aprovechadas por java: La reducción de tiempos en el sistema 4 (de doble núcleo) así lo atestigua. La cantidad de RAM queda descartada como factor influyente al menos a partir de 512 MB, esto es, con 512 MB o más de RAM, la variación de rendimiento es nula. De hecho el sistema 5 también cuenta con 2 gigas y

presenta los peores resultados de todos.

- Con tamaño 2x, el tiempo para pintar un frame es mayor: debido al escalado de la pantalla por software.

- El modo 256 colores presenta resultados dispares: mejores en algunas arquitecturas mientras que en otras los tiempos son peores. Incluso hay arquitecturas, como el sistema 6, en las que en un tamaño de pantalla es más rápido 24bpp y en el otro 256 colores. Quedará pues a decisión del usuario final activar un modo u otro para su computadora, ya que siempre podrá ver los frames por segundo con una u otra opción.

5.2.3 Diferencias entre máquinas virtuales

Para los sistemas 1 y 2 se han realizado pruebas con diferentes versiones del SDK (máquina virtual y compilador). Los resultados se resumen en la siguiente gráfica:

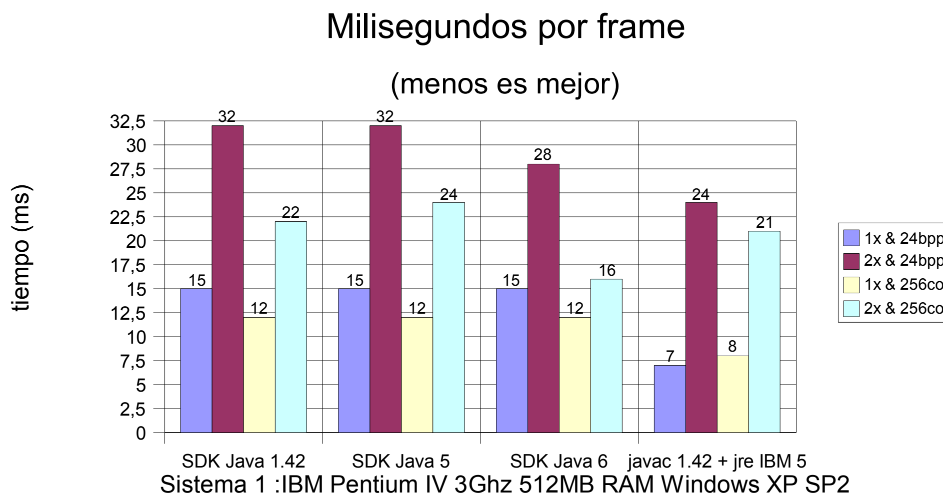


Ilustración 5.2: Comparativa de rendimiento del sistema 1 para diferentes SDKs

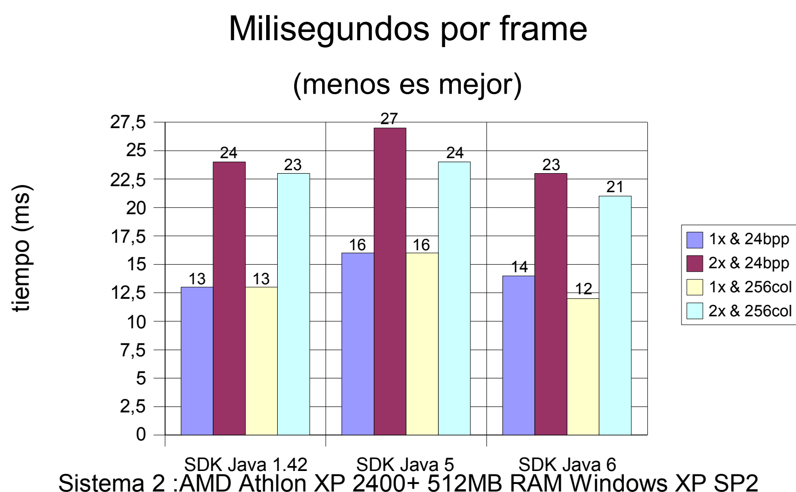


Ilustración 5.3: Comparativa de rendimiento del sistema 2 para diferentes SDKs

Se puede concluir que:

Java 1.42 es asombrósamente más eficiente que Java 5 en el sistema 2.

Las diferencias entre Java 1.42 y Java 5 no son significativas en el sistema 1. Si se aprecia una mejora real en Java 6 respecto a la versión 5 en ambos sistemas:

Mejora entre Java 6 y Java 5		
Prueba	***Sistema 1***	***Sistema 2***
1x y 24bpp	-	14%
2x y 24bbp	14%	17%
1x y 256 colores	-	33%
2x y 256 colores	50%	14%

Tabla 5.4: Diferencias de rendimiento entre Java 5 y Java 6

La máquina virtual de IBM es sorprendentemente mucho más eficiente que la de Sun, especialmente con imágenes sin escalar (1x). El motivo por el cual no se ha probado en la arquitectura 2 es porque no es instalable: IBM restringe la instalación de la misma a máquinas IBM mediante una comprobación de BIOS en el instalador. La mejora en este caso es la siguiente:

Mejora en Sistema 1		
Prueba	***JRE 5 de IBM vs Java 5 de Sun***	***JRE5 de IBM vs Java 6 de Sun***
1x y 24bpp	114%	114%
2x y 24bbp	33%	17%
1x y 256 colores	50%	50%
2x y 256 colores	14%	-31% (31% peor)

Tabla 5.5: Porcentaje de mejora en sistema 1

5.2.4 Diferencias entre optimizaciones

Ésta sección hace referencia a optimizaciones logradas mediante mediante el uso de software especializado. Las optimizaciones “manuales” se detallan en el apartado correspondiente del capitulo 4.

- HotSpot: Las máquina virtuales modernas de Sun cargan en memoria un compilador JIT (HotSpot) que, entre otros, realiza compilaciones e inlining de las funciones más llamadas. Existen 2 compiladores HotSpot diferentes:
  - Client: Orientado a aplicaciones gráficas en las que se requiere un tiempo de respuesta y de arranque cortos.
  - Server: Orientado a servidores (2 gigas de RAM en adelante) en las que se requiere un rendimiento sostenido mayor frente al tiempo de arranque o de respuesta ante eventos.

Hasta ahora, todas los resultados mostrados han sido obtenidos con HotSpot client (la usada por defecto en Windows). Aún no disponiendo de los 2 gigas de RAM recomendados por Sun, se puede forzar a usar la máquina server mediante un parámetro. Así, se ha usado la máquina server en el sistema 2 con resultados contraproducentes:

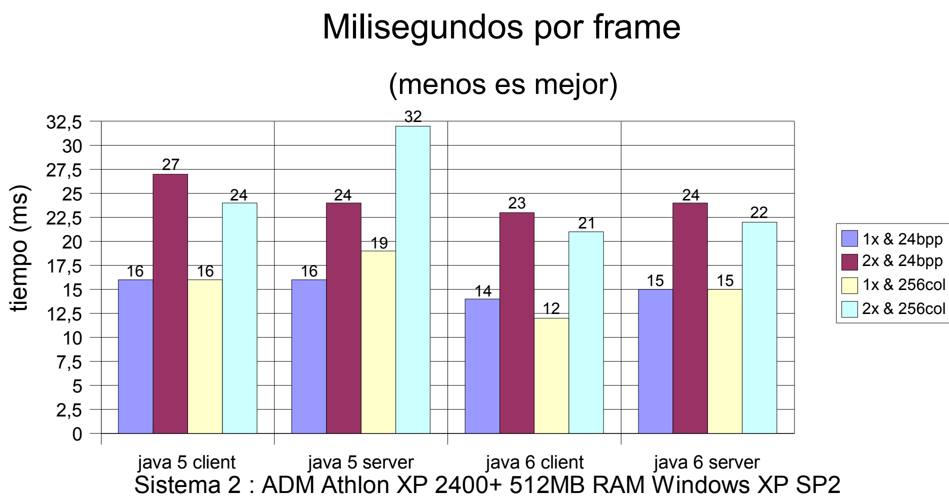


Ilustración 5.4: Comparativa de rendimiento del sistema 2 para diferentes HotSpot

Como puede verse, los tiempos al utilizar HotSpot server son claramente mayores. Por otro lado los resultados son lógicos pues este compilador JIT está optimizado para máquinas servidoras, es decir, con varios GB de memoria RAM entre otros.

- Opción -O: El compilador javac admite la opción -O para realizar optimizaciones del código. Tanto en java 5 como en 6, ésta opción no ha producido ningún cambio en los tiempos por sorprendente que parezca.

- Proguard: Proguard es un optimizador, ofuscador y reductor (shrinker) de código Java de licencia GPL, uno de los mejores disponible [14]. Ofuscar el código, aparte de la aplicación obvia de sustituir nombres comprensibles por humanos por letras sin sentido (dificultando así enormemente la comprensión del código decompilado), tiene un efecto secundario: al sustituir nombres de variables, métodos y clases consistentes en una o varias palabras por unas pocas letras se reduce el tamaño del archivo class resultante. Reducir el tamaño de una clase tiene, aparte del evidente ahorro de memoria RAM, más posibilidades de ser más eficiente al poder caber más código en la misma cantidad de cache, con el consecuente ahorro de accesos a memoria que conlleva.

  Se ha usado proguard 4.0 beta 5 con optimización y ofuscación y los resultados obtenidos son los siguientes:

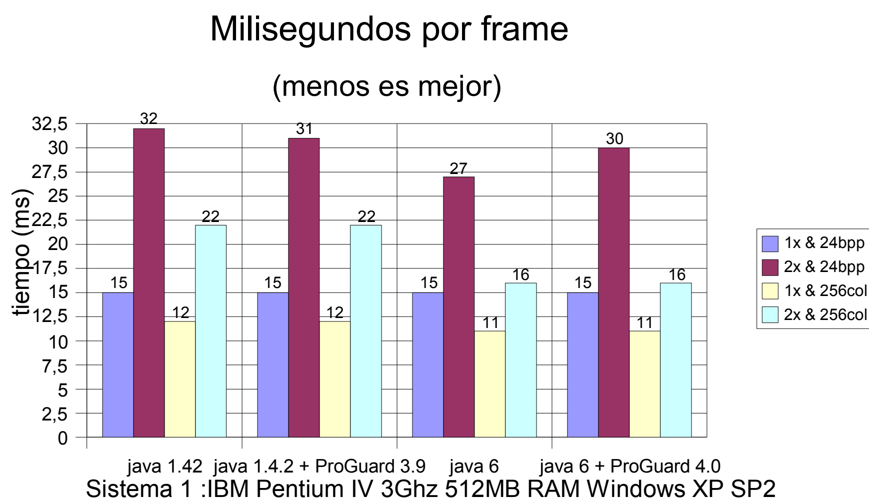


Ilustración 5.5: Comparativa de rendimiento del sistema 1 para proguard

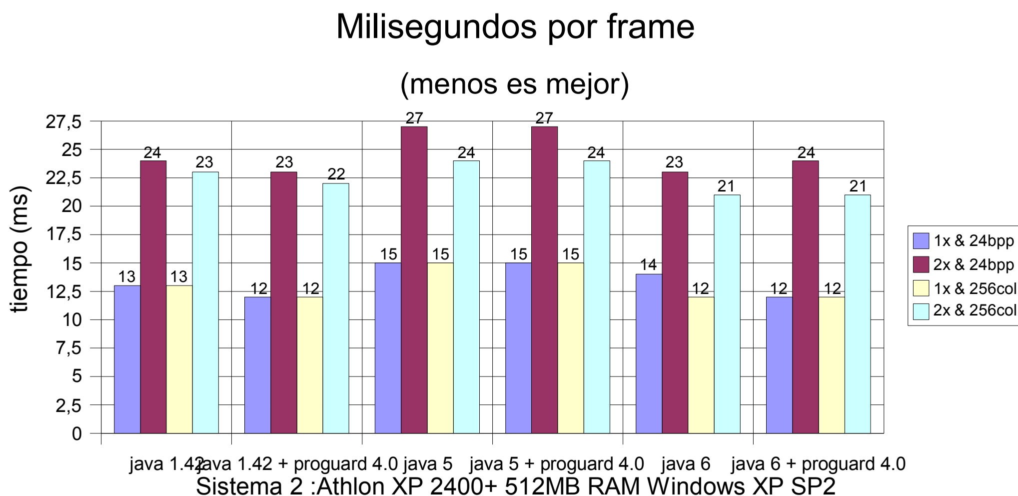


Ilustración 5.6: Comparativa de tiempos del sistema 2 para proguard

Las optimizaciones producen ligeras mejoras en algunos sistemas con ciertas MV pero en otros producen unos tiempos ligeramente peores. Se observa el mayor aumento de rendimiento para el sistema 2 con java 1.4.2: 8% para las pruebas de tamaño 1x, 4% para 2x y 24bpp, y 5% para 2x y 256colores. En java 6 la prueba 2x y 24bpp da peores resultados tras la optimización que antes de realizarla. Después de los resultados obtenidos, con rendimientos tan dispares y dependientes de la arquitectura, se concluye que no merece la pena "optimizar" el código final.

5.3 Profiling

Saber qué partes del código se ejecutan más y cuales gastan más tiempo es muy importante si se quiere realizar una optimización del código. Ya que un emulador es un programa que consume mucho tiempo de procesador, es de una gran importancia el recopilar ésta información sobre el mismo. Se han realizado pruebas con el plugin para profiling de Eclipse perteneciente a la TPTP (ver [12]) y se han medido 3 factores:

- Tamaño en memoria: La cantidad de memoria RAM gastada por el programa en diferentes condiciones. En una PDA, la RAM es más escasa que en un PC por lo que es una medida a tener en cuenta.

- Tiempo empleado en cada función: La cantidad de milisegundos que tarda en ejecutarse cada función. Estos datos han servido para centrar los esfuerzos de optimización en aquellas funciones que más tiempo total (que no por cada llamada) del procesador consumen.

- Número de llamadas a cada función: El número de veces que se llama cada función. Las funciones más llamadas son siempre las candidatas a optimizaciones. Concretamente las funciones cuyo producto del número de llamadas multiplicado por el tiempo consumido por llamada sea máximo, el llamado "tiempo total consumido por función".

Se denomina tiempo base al producto del número de llamadas multiplicado por el tiempo consumido en función, sin contar el tiempo empleado en posibles llamadas a subfunciones. Se denomina tiempo acumulativo al producto del número de llamadas multiplicado por el tiempo consumido en función, contando el tiempo empleado en posibles llamadas a subfunciones. En el caso de que una función no llame a ninguna otra, el tiempo base y acumulado coincidirán.

Las pruebas se han realizado con la ROM de referencia que ya se usó para la comparativa de eficiencia, "Castlevania 2 Belmont's Revenge", y han consistido en dejar durante 8 horas el entorno Eclipse ejecutando el emulador en modo profiling (la intro del juego en un bucle infinito al no pulsarse ninguna tecla) con el fin de que los porcentajes se establecieran a unos valores cercanos

a los reales.

Los datos presentados a continuación son un resumen de todos los recopilados. Se pueden consultar los ficheros html de salida del plugin de profiling en la carpeta /datos_de_profiling del cd adjunto.

5.3.1 Consumo de memoria

Respecto al consumo de memoria por parte del emulador, se obtuvieron los siguientes resultados:

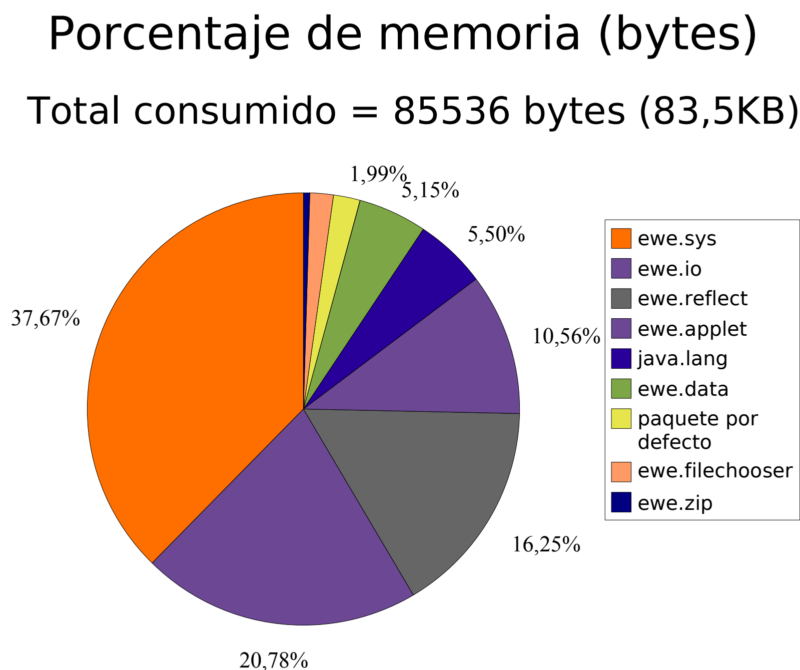


Ilustración 5.7: Consumo total de memoria en GINAGE

Los datos aquí presentados son para el consumo total de memoria, es decir, la memoria total ocupada por el emulador. Existe también el concepto de memoria activa que hace referencia a la memoria ocupada por las instancias que aún siguen vivas en memoria (no han sido eliminadas por el recolector de basura). El valor que interesa para estas estadísticas es evidentemente el primero pues es una cota superior al consumo de memoria máximo aunque se pueden encontrar los valores de los tamaños activos así como en número de instancias totales, vivas y recolectadas en la carpeta /datos_de_profiling.

El tamaño total consumido en memoria, contando el de las clases de Ewe usadas por el emulador, es de 83,5KB. Según [13], el tamaño de la máquina virtual de Ewe es de unos 550KB para una aplicación sencilla por lo, como era de esperar, el consumo de memoria es muy reducido.

5.3.2 Tiempo empleado por función

Tras realizar la medición con el profiler, se obtuvieron los siguientes resultados:

Porcentaje de tiempo

(paquete principal vs resto de clases)

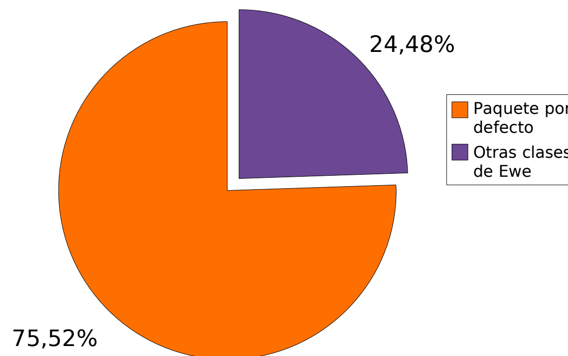


Ilustración 5.8: Porcentaje de tiempo empleado por clases propias.

Se observa que aproximadamente el 25% del tiempo el emulador se halla ejecutando métodos de otras clases ajenas a las propias, esto es, otras clases de Ewe relativas a la gestión de hilos, recolección de basura etc. Lo peor de este tiempo es que no se puede reducir ya que forma parte del propio funcionamiento interno de la librería. Ese 75% de tiempo real aprovechable por la aplicación, se distribuye de la siguiente manera entre las clases:

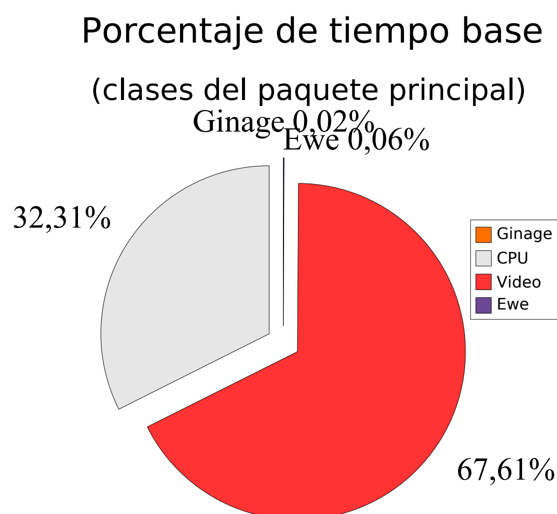


Ilustración 5.9: Porcentajes de tiempo base para clases principales

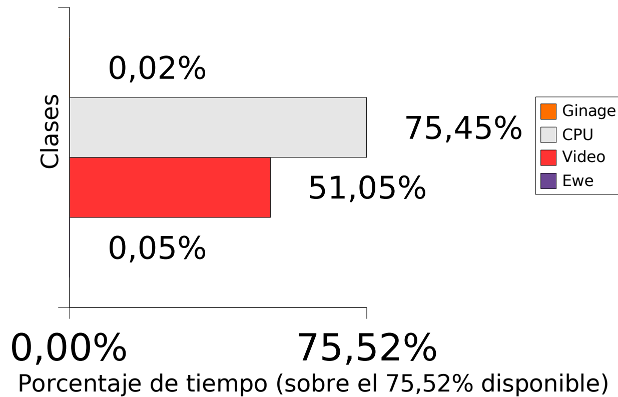


Ilustración 5.10: Porcentajes de tiempo acumulativo para clases principales

Como se ve en el diagrama circular, 1/3 del tiempo base (tiempo empleado en ejecutar el código de la propia clase, excluyendo llamadas a métodos de otras clases) es empleado por la clase CPU y 2/3 por Video. El resto de clases tienen tiempo marginales como era de esperar, ya que el bucle principal de la emulación sólo llama a métodos de CPU y Video.

En el diagrama de barras se disponen los tiempos acumulativos (tiempo empleado en ejecutar el código de una clase, incluyendo las llamadas a métodos de otras clases). Se ve que CPU, GINAGE y Ewe copan el total disponible para la aplicación (75,52%). La CPU emplea un 75,45% (prácticamente el total) ya que sus métodos llaman a la clase Video periódicamente.

Un análisis más detallado de las clases principales muestra los métodos que más tiempo consumen. La clase CPU presenta los siguientes tiempos:

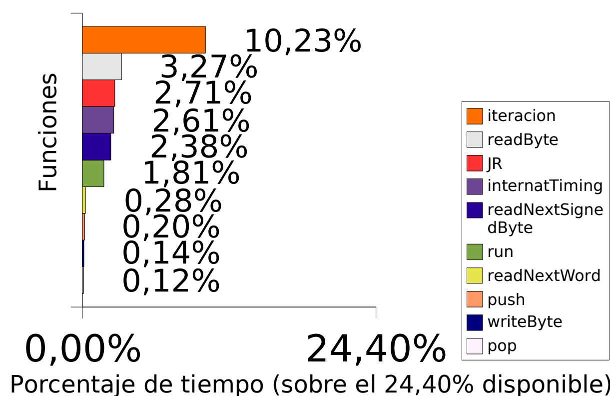


Ilustración 5.11: Porcentajes de tiempo base para funciones de la clase CPU

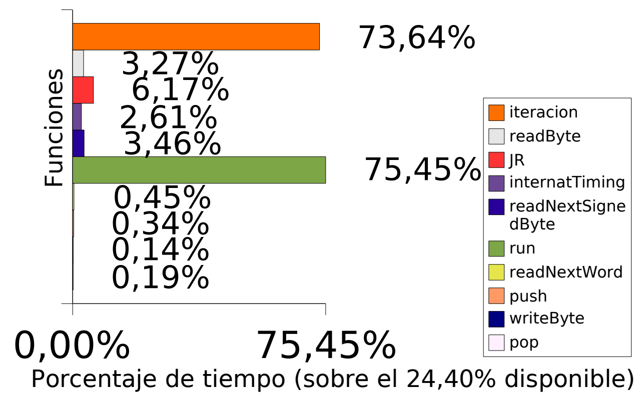


Ilustración 5.12: Porcentajes de tiempo acumulativo para funciones de la clase CPU

La función iteración, como era de esperar, es la que más tiempo consume pues es la función principal encargada de atender interrupciones, cambiar el modo del LCD (si procede, pintando la pantalla y calculando la linea actual), generar interrupciones (si procede), decodificar y realizar la operación del opcode actual, modificar el estado interno, etc. Esta función llama a varias otras de la clase Video (para refrescar la pantalla y calcular la linea actual por ejemplo) por lo que su tiempo acumulado se dispara.

ReadByte es la segunda función que más tiempo gasta (y una de las más llamadas como después se verá) pues leer un byte de memoria es algo común a prácticamente todas las instrucciones por lo que llamar a esta función es extremadamente frecuente. Como no llama a ninguna otra función, su tiempo base y acumulativo coinciden.

JR (salto relativo al PC) también es llamada con mucha frecuencia y emplea un tiempo considerable. No es de extrañar ya que es un opcode común a muchas ROMs (cualquier bucle de ensamblador necesita una instrucción de tipo JR condicional a un flag).

InternalTiming es extremadamente llamada, al menos una vez por cada opcode pues es la encargada de avanzar el timing interno del emulador por cada instrucción ejecutada un número de ciclos de reloj de Game Boy. Al ser tan llamada, es normal que consuma bastante tiempo.

ReadNextSignedByte lee con signo el siguiente byte de memoria. Es llamada por el opcode JR (entre otros) por lo que es al menos igual de frecuente que éste. De ahí sus tiempos.

Run (se recuerda de CPU es implementada como un hilo) consiste únicamente en un bucle while que llama a iteración si una variable booleana es igual a true, su tiempo base es muy pequeño pero su tiempo acumulativo es el total del gastado por la clase CPU.

El resto de funciones corresponden a otras funciones llamadas con relativa frecuencia

como son:

- readNextWord: leer palabra de 16 bits de memoria.

- push: Operación de introducir una dirección en la pila. Usada internamente por los opcodes CALL y RST (interrupciones) además de por el propio opcode PUSH. Las operaciones de pila son relativamente comunes.

- writeByte: Escritura de un byte en memoria. Muchos opcodes necesitan escribir en memoria (aunque es infinitamente menos frecuente que readByte por supuesto).

- pop: Operación de introducir una dirección en la pila. Usada internamente por los opcodes RET y RETI (vuelta de interrupción) además de por el propio opcode POP. Las operaciones de pila son relativamente comunes.

La clase Video aún poseyendo varias funciones, presenta un patrón de consumo de tiempos mucho más simple. Destacan 3 funciones por encima del resto (el porcentaje de tiempo de las demás es insignificante):

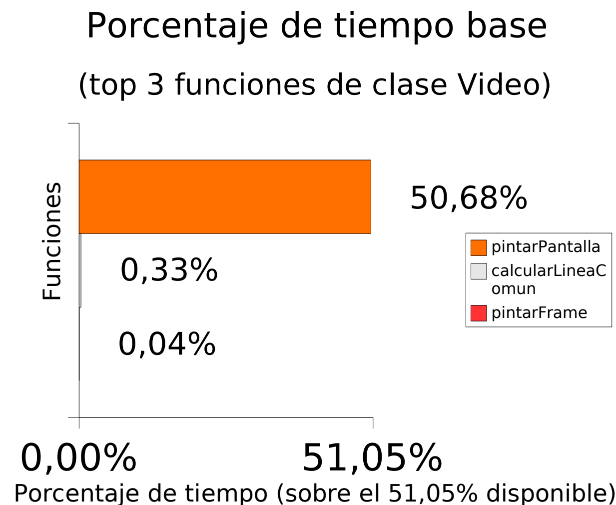


Ilustración 5.13: Porcentajes de tiempo base para funciones de la clase Video

El tiempo consumido en la función pintarPantalla es alarmante: Más del 50% del tiempo del emulador es acaparado por la función que vuelca el contenido de un array con los valores RGB de cada pixel a pantalla. Si se tiene en cuenta además que para las pruebas de tiempo no se ha usado el escalado de imagen se desprende claramente que Ewe tiene un cuello de botella con el volcado del buffer de imágenes a pantalla.

calcularLineaComun es la función encargada de obtener los valores RGB para la linea actual (indicada por el registro LY). El cálculo de los valores RGB es complejo pues depende del modo de funcionamiento del emulador (GB o GBC) y se tienen en cuenta múltiples prioridades entre la capa BG, WIN y SPRITE, así como las propiedades de los tiles que influyen en el dibujado como la posición, el tamaño (8x8 o 8x16), si están invertidos o no en alguno de los ejes, etc. Ésta función es llamada desde la función iteración de la clase CPU cada vez que se entra al modo 3 de LCD (144 veces por frame).

Por último, la función pintarFrame devuelve true o false dependiendo de si el frame actual ha de pintarse (y calcularse) o no, dependiente del valor de frameskip seleccionado en las opciones. Esta función es llamada desde la clase CPU como condición antes de llamar también a calcularLineaComun.

5.3.3 Conclusiones

Tras comprobar los tiempos consumidos por la propia librería Ewe se deduce que la misma no está pensada para aplicaciones de consumo excesivo de recursos. Una aplicación GUI sencilla puede funcionar muy bien ya que la CPU esta ociosa la mayor parte del tiempo y las llamadas a los métodos sleep() y wait() que hace Ewe internamente permiten atender eventos rápidamente. En el caso de un emulador, la CPU está al ocupada constantemente ejecutando opcodes, calculando lineas de la pantalla, etc. Los propios eventos de teclado no se atienden debido a ésto (único hilo no interrumpible por la MV) por lo que se tiene que recurrir a parar periódicamente el hilo de usuario para "darles tiempo" a ser tratados por la función que los atiende.

Por otro lado, es inaceptable los tiempos que usa Ewe para renderizar una imagen. Incluso el tiempo para hacer el escalado de la imagen mediante las funciones de Ewe es peor que la implementación manual del algoritmo. Definitivamente ésto es un problema intrínseco a la implementación de las librerías por lo que no se puede hacer gran cosa, sopena de arriesgarse a romper la compatibilidad con las MV nativas a las otras plataformas.

Por lo demás, el consumo de tiempos presentado en las demás clases sigue lo esperado: La función iteración es la que consume todo el tiempo acumulativo del que se dispone (cerca del 75%). ReadByte() JR() y internatTiming() son llamadas muy frecuentemente como era de esperar de la lectura de bytes de memoria, los saltos relativos al PC y el avance del timing interno (llamado por cada opcode). El resto de funciones las componen otras que también son bastante comunes como las operaciones de push y pop de la pila o las escrituras de bytes en memoria. En general, dependiendo de la ROM que se pruebe para el test los resultados variarán debido a que los opcodes usados por cada juego tienen diferente frecuencia de aparición pero las principales funciones siempre ocuparan los puestos de cabeza. Se puede obtener una información detallada de los tiempos, número de llamadas y consumo de memoria en la carpeta documentación/resultados-_de_pruebas_de_rendimiento/datos_de_profiling del cd adjunto.

5.4 Análisis de una posible explotación comercial

5.4.1 Estado del mercado

En el mercado PC es difícil obtener dinero con un emulador. Existen muchos emuladores gratuitos, la mayoría libres, donde los desarrolladores obtienen dinero a base de donaciones principalmente.

Hace 15 años, la realidad era radicalmente distinta. La Game Boy era una consola emulada con escaso éxito y los pocos emuladores con una compatibilidad decente, eran de pago y privativos. El emulador gnuboy surgió en el 2000 y abrió el camino a los posteriores emuladores libres que vendrían hasta la actualidad.

El mercado de los dispositivos móviles sin embargo es bastante más fructífero en términos económicos. La escasez de emuladores gratuitos y de calidad y la proliferación de compañías de descargas de juegos para el móvil, hacen que este mercado sea extremadamente productivo. Ha de tenerse presente que muchos juegos clásicos que se venden promocionándose como tales por estas compañías, no son más que emuladores con la propia ROM del juego embebida.

5.4.2 Costes de amortización

Existen múltiples formas mediante las cuales GINAGE podría generar beneficios. Muchas de ellas son formas tradicionalmente empleadas por muchos proyectos de software libre para autofinanciarse.

- **Publicidad tradicional**: Es una de las prácticas más comunes. Consisten en alguna parte de la página web imágenes publicitarias (banners en ingles) anunciando algún producto o servicio de manera que al hacer click sobre el anuncio el navegador es direccionado a la pagina web del anunciante. Ya que la web alberga un emulador de Game Boy para PC y PDA, los anunciantes más interesados por anunciarse en principio podrían ser:
  - Vendedores de PDAs y software de entretenimiento para las mismas
  - Páginas de descargas de programas para PDAs.
  - Páginas de temática sobre videojuegos o emuladores
  - Páginas dedicadas a la venta especifica de accesorios de importación para consolas

- **Google Adsense**: Es un sistema ofertado por Google que permite recibir dinero por publicidad incrustada en la propia web. Los anuncios ofertados se corresponden con la temática de la propia página. El dinero ganado de ésta forma es proporcional al número de clicks que los visitantes hagan en los banners desde la página.

En ambas forma de publicidad es preciso mantener el número de visitas a base de contenidos interesantes y cambios frecuentes. El desarrollo del emulador debería ser constante para asegurarse suficientes novedades para atraer visitantes.
La creación de unos foros públicos donde los usuarios se pudieran relacionar, dar sus impresiones y exponer sus problemas con el emulador, ayudaría mucho a mantener un núcleo de visitas fijo.

- **Donaciones**: Algunos proyectos de software libre como Blender se financian en su mayor parte por medio de donaciones anónimas. En este sentido, a los tradicionales métodos de ingreso en número de cuenta, giro postal, etc ha surgido desde hace unos años PayPal. Este servicio permite a cualquier persona con una cuenta PayPal pagar sin necesidad de dar sus datos bancarios en Internet. Este sistema ha demostrado ser el más eficaz por su comodidad: sólo se necesita conocer la dirección de correo de la persona a la que se le quiere transferir dinero.

- **Relicenciar el código de forma dual**: Se podrían crear dos versiones del emulador, una libre y gratuita y la otra de pago. La de pago podría tener características añadidas cómo por ejemplo un depurador integrado de calidad. También podría licenciarse una única versión con dos licencias dependiendo de si se va a usar con ánimo de lucro o no. Así, por ejemplo, si alguna compañía decidiese portar directamente el código para sus dispositivos móviles y venderlo se les podría pedir un porcentaje sobre los beneficios.

- **Modificación del software a medida**: El proyecto podría modificarse para adaptarse a nuevos requisitos o portarse a nuevas plataformas. Por ejemplo, una empresa de juegos para móviles podría interesarse en portarlo para teléfonos J2ME o añadirle algunas características nuevas (soporte de sonido, multijugador, etc).

- **Venta del código fuente**: Se ha estimado una média de 20 horas semanáles decicadas al desarrollo. La duración del mismo ha sido de aproximadamente 6 meses. Para un coste de 12€ por hora de programación, se éstima que el código fuente de GINAGE ha costado entre 5700 y 6200€. La venta del mismo podría hacerse por una cantidad superior.

Capítulo 6. CONCLUSIONES Y PROPUESTAS

6.1 Objetivos alcanzados (consecución de los objetivos)

GINAGE partía con el objetivo de funcionar en plataformas móviles y PC. Aunque llega a ejecutarse en MV nativas de Ewe para PDA, su lentidud lo hace difícilmente utilizable para su propósito principal. La elección de Ewe no fue la más adecuada para realizar el código. Como ya se ha visto en el capítulo 5, su pobre rendimiento en el pintado de gráficos junto a los problemas derivados de la programación con un sólo hilo no interrumpible y los problemas con los eventos KEY_RELEASE han hecho que finalmente utilizar Ewe haya supuesto más trabas que las que en principio se podría esperar de una librería. No obstante la elección por Ewe fue correcta en el sentido de que éstos problemas no se conocían en un principio y se eligió principalmente por funcionar en un rango de dispositivos amplio, por ser mejor opción que Superwaba y, por último, por ser software libre.

A la luz de los resultados la mejor solución hubiera sido utilizar algún lenguaje como C o C++ junto a alguna librería gráfica multiplataforma como SLD con su port de Windows CE. También existen ports para SymbianOS y Dreamcast aunque no son oficiales y podrían implementar un subconjunto de las librerias. Quizas podría haberse orientado el proyecto a un emulador multiplataforma PC, Dreamcast o similar.

6.2 Propuestas de desarrollo futuras:

Aunque los resultados obtenidos son funcionales en PC, se podría pensar en mejorar GINAGE en los siguientes puntos:

- **Aumento de compatibilidad**: El 86% y 45% actual para juegos de GB y GBC puede crecer mucho más si se depura el código para las ROMs problemáticas. Un flag puesto a 0 en lugar de 1 o un timing incorrecto de una instrucción por poner un ejemplo pueden provocar que varios títulos no funcionen correctamente. En las pruebas se han observado problemas de capas con algunos juegos de GBC que posiblemente se deban a un error de implementación interno. Buscar patrones en común entre los juegos que fallan y analizar la traza del depurador podrían dar con el problema. Para mejorar la rapidez en la corrección de bugs, se propone el siguiente punto como mejora.

- **Mejora del depurador integrado**: El depurador usado en las primeras versiones no deja de ser una herramienta rudimentaria y para desarrollo interno. La reescritura del depurador, con una interfaz gráfica y soporte para puntos de ruptura (breakpoints) así como modificación en caliente del estado del procesador podrían acelerar la corrección de bugs. Los puntos de ruptura podrían implementarse como un opcode más aprovechando que algunos valores no tienen opcode asociado. Introducir un breakpoint equivaldría a insertar un byte de esa instrucción ficticia en una posición de memoria determinada. La modificación en caliente de registros, flags y estado del MBC es sencilla de hacer pues simplemente consiste en cambiar los valores de las variables que los representan.

- **Optimización de la emulación:** Aún quedan posibles puntos de optimización. Podrían utilizarse interfaces y escribirse 2 clases CPU y 2 clases Video, cada una de ellas específica para GB y GBC. Los juegos de GBC instanciarían un par de clases y los de GB otras. Ésto permitiría ahorrar condiciones if para el caso de GB y GBC que actualmente están en el código consumiendo tiempo.
  También se puede implementar un modo de emulación pantalla a pantalla en lugar de linea a linea, esto es, ejecutar un número de ciclos de CPU igual a los empleados en refrescar una vez la pantalla y después generar la pantalla en lugar de ejecutar un número de ciclos de CPU igual a los empleados en pintar una línea y después generar la línea. La emulación pantalla a pantalla es más eficiente pero mucho menos precisa pues algunos efectos gráficos realizados durante la interrupción LCD_STAT como el uso de más de 56 colores en pantalla intercambiando las paletas en la linea LYC son imposibles de emular.

- **Soporte de Sonido**: Aunque Ewe no proporciona soporte para implementar sonido debido a las escasas posibilidades de muchas PDAs, se podría implementar en una versión de PC. Como Ewe posee clases wraper de la mayoría de sus clases que facilitan la portabilidad desde Java con clases de API idéntica, lo más sencillo sería reescribir el código en Java que sería prácticamente inmediato. J2SE si ofrece buen soporte para generar tonos y variar la frecuencia de éstos por lo que es factible de implementarse el sonido, al menos, en PC. Utilizando la Java Sound API y la clase DataLine que permite escribir buffers de bytes a un canal de audio mediante el método write se podría componer el buffer a partir de los canales de audio de la GB.

- **Implementación de MBC 7 y otros**: Aunque se han implementado los MBCs más comunes, algunos juegos específicos usan otros MBCs (por ejemplo Kirby Tilt 'n' Tumble). Una vez alcanzada una compatibilidad decente para los otros MBCs se podría pensar en dar soporte a estos MBCs exóticos. No existe información de su funcionamiento interno en

Internet salvo por el código fuente de algún emulador que los soporta (p.e. GEST). En cualquier caso, ésta mejora sería la última en hacerse dado el escaso número de títulos que usan estos MBCs (ver 6.2.1.1 Tipos de controladores de banco).

- **Soporte para trucos**: GameGenie y GameShark eran dos dispositivos situados entre el cartucho original y la ranura de cartuchos. Al encender la consola, se le presentaba una pantalla al usuario para que introdujese unos códigos en hexadecimal. Cada código introducido consiste en un entero que es leído en lugar de la posición de memoria correspondiente, por lo que el código del juego a todos los efectos queda parcheado posibilitando el uso de trucos como vidas y tiempo infinitos, desbloqueo de pantallas ocultas, etc. La implementación del soporte de los códigos de estos dos dispositivos es sencilla siendo una característica ampliamente implementada por muchos emuladores. Únicamente hay que conocer qué representan los números introducidos para modificar la posición de memoria correspondiente.

- **Soporte de otros periféricos**: Principalmente los accesorios Gamelink y Super Game Boy para dotar de multijugador, color y otras mejoras a los juegos originales. El cable Gamelink, aún estando su funcionamiento perfectamente documentado, es difícil de emular principalmente debido a la velocidad de transmisión del mismo (ver Taxonomía de emuladores de PC).
  El Super Game Boy se comunica con el chip de SNES mediante comandos enviados por el puerto 0xFF00 (joypad). Existe documentación del mismo en Internet y es una característica común a muchos emuladores por lo que es factible de ser reproducida.

Bibliografía

[AFF+98] P. Anthrox, M. Fayzullin,P. Felber, P. Robson, M. Korth.Game Boy CPU Manual. 1998.

[FFRK01] M. Fayzullin, P. Felber, P. Robson, M. Korth.Pan docs. 2001.

[FFRK98] M. Fayzullin, P. Felber, P. Robson, M. Korth.GBSpec. 1998.

[Frohwein01] Jeff Frohwein.GB/GBC Opcode Summary. 2001.

[Lloyd00] Justin Lloyd.Game Boy Cribsheet. 2000.

[MF01] Victor Moya del Barrio y Agustin Fernandez.Study of the techniques for emulation programming. 2001.

[Rol06] Ángel Roldán.Comparativa entre Superwaba y Ewe. 2006.

Referencias web

[1] *Wikipedia : Emulator article*. http://en.wikipedia.org/wiki/Emulator

[2] *Wikipedia : Crusoe*. http://en.wikipedia.org/wiki/Crusoe

[3] *Emulador MAME*. http://www.mamedev.org

[4] *emulador ZSNES*. http://www.zsnes.com

[5] *emulador ScummVM*. http://www.scummvm.org

[6] *Reiner Ziegler*. http://www.reinerziegler.de/readplus.htm

[7] *Wikipedia : Game Boy line*. http://en.wikipedia.org/wiki/Game_Boy_line

[8] *Historia de los videojuegos*. http://www.elotrolado.net/wiki/Historia_de_los_videojuegos

[9] *Web de GINAGE*. https://dev.oreto.inf-cr.uclm.es/www/ginage/

[10] *Wikipedia: ROM hacking article*. http://en.wikipedia.org/wiki/Rom_Hacking

[11] *Página oficial de descarga de Java SUN*. http://www.java.com/es/download

[12] *Eclipse Test & Performance Tools Platform Project*. http://www.eclipse.org/tptp/

[13] *Ewe vs PersonalJava*. http://www.ewesoft.com/EweDetails/VersusPersonalJava.htm

[14] *Proguard web*. http://proguard.sourceforge.net

Anexo A: Diagramas a color

Bytes	*Game Boy*	*Bytes*	*Game Boy Color*
0000-3FFF	16KB ROM banco 0	0000-3FFF	16KB ROM banco 0
4000-7FFF	16KB ROM bancos 1-n	4000-7FFF	16KB ROM bancos 1-n
8000-9FFF	8KB RAM de Vídeo (VRAM)	8000-9FFF	8KB VRAM banco 0-1 en modo GBC
A000-BFFF	8KB RAM externa o de cartucho(puede tener bancos o no existir)	A000-BFFF	8KB RAM externa o de cartucho(puede tener bancos o no existir)
C000-DFFF	8KB RAM interna o de trabajo	C000-CFFF	4KB RAM interna o de trabajo banco 0
		D000-DFFF	4KB RAM interna o de trabajo bancos 1-7 en modo GBC
E000-FDFF	Echo RAM (mismos valores que C000-DDFF)	E000-FDFF	Echo RAM (mismos valores que C000-DDFF)
FE00-FE9F	Sprite Attribute Table (OAM)	FE00-FE9F	Sprite Attribute Table (OAM)
FFA0-FEFF	No usable	FFA0-FEFF	No usable
FFOO-FF7F	Puertos (registros) de E/S	FFOO-FF7F	Puertos (registros) de E/S
FF80-FFFE	RAM interna (Hight RAM)	FF80-FFFE	RAM interna (Hight RAM)
FFFF	Interrupt Enable Register	FFFF	Interrupt Enable Register

Ilustración A.1: Mapa de memoria en la Game Boy y Game Boy Color

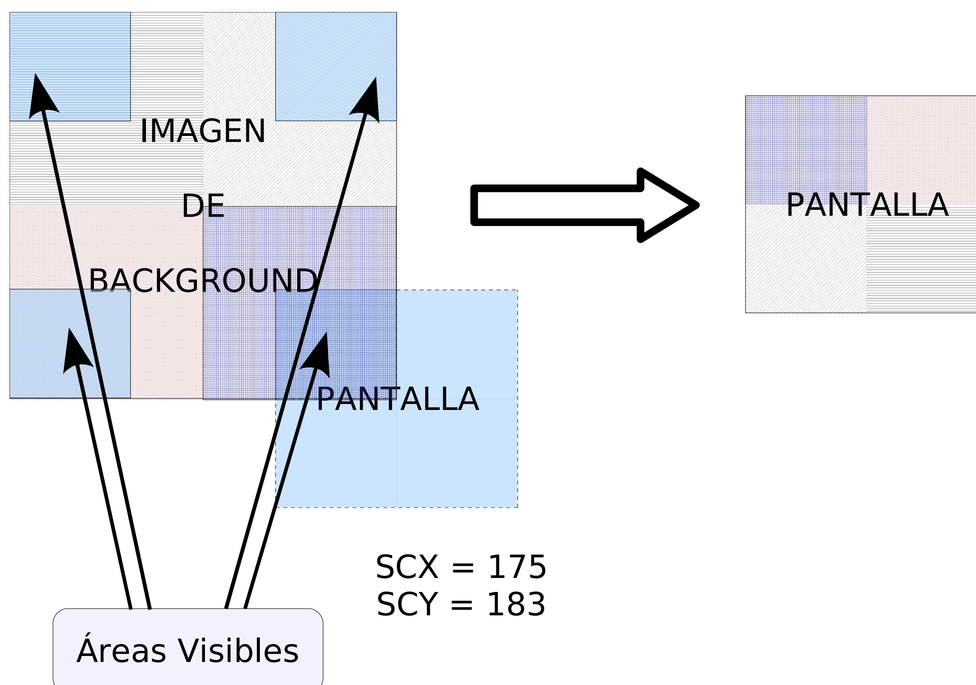


Ilustración A.2: Efecto wrap en capa Background

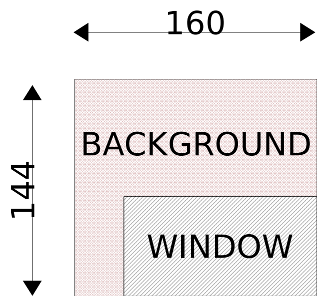
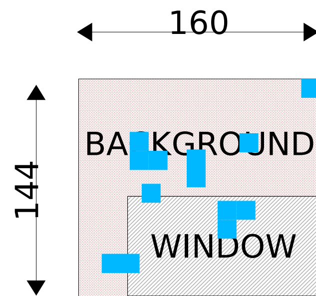


Ilustración A.3: Capa Window sobre capa Background



Sprites =

Ilustración A.4: Capa Sprite sobre Window y Background)

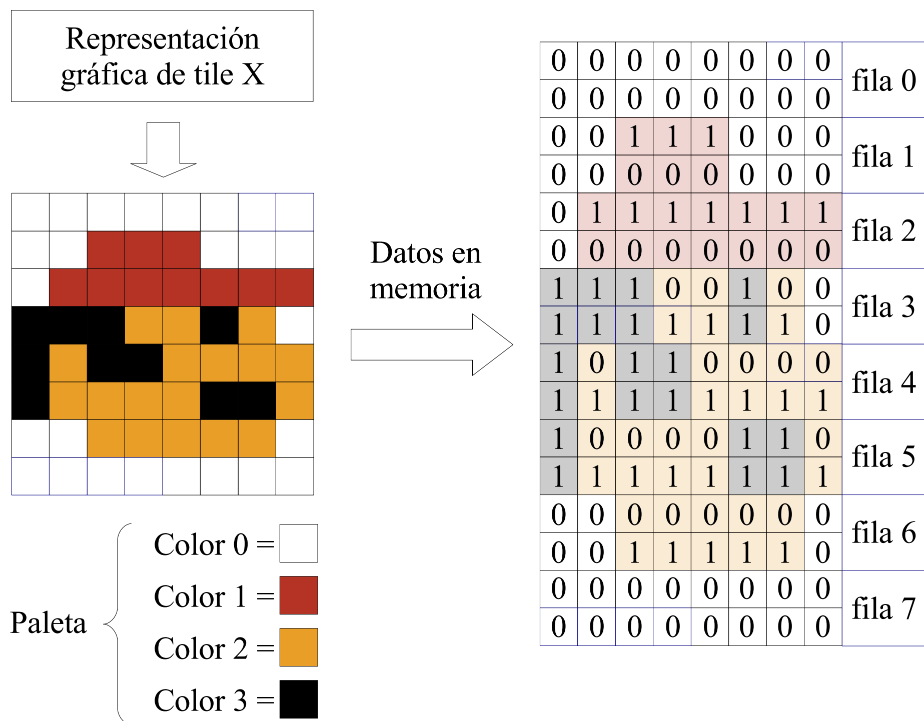


Ilustración A.3: Representación de tile en memoria.

El color de cada pixel es codificado con 2 bits en 2 bytes consecutivos en memoria

Anexo B: Manual de usuario

Requisitos previos:

GINAGE requiere un JRE 1.4 de Sun o superior para funcionar. GINAGE puede funcionar con versiones 1.2 y superiores del JRE pero necesitará recompilar los fuentes usted mismo. Si no sabe lo que es compilar, descárguese la versión compilada e instálese un JRE 1.4 o superior (si es que aún no tiene uno). Puede descargarse la última versión del JRE de Sun de [11].

Iniciar el programa:

GINAGE se distribuye como un único fichero jar donde se incluyen la propia aplicación y el runtime de las librerías Ewe para java. Para ejecutar el emulador, haga click sobre el fichero. Si su ordenador no está configurado para ejecutar los ficheros jar directamente, sitúese en el directorio donde ha copiado el jar desde el linea de comandos y tecle la orden “java -jar ginage.jar”.

La interfaz de usuario:

La interfaz principal del programa presenta el siguiente aspecto:

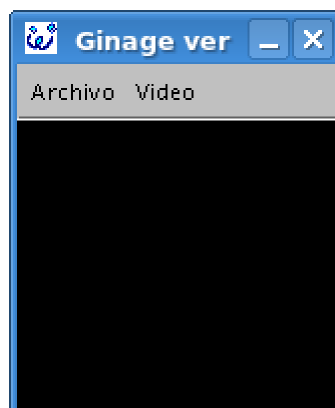


Ilustración B.1: Pantalla principal de GINAGE

Puede verse la pantalla principal donde se dibujan los gráficos junto a los menús Archivo y Video justo encima. A continuación se detallan las opciones del menú Archivo:

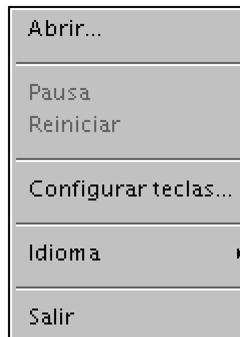


Ilustración B.2: El menú Archivo

- Abrir...: Carga una nueva ROM de Game Boy o Game Boy Color. Los ficheros han de tener las extensión .gb o .gbc. También se pueden cargar ficheros comprimidos en .zip y, en algunos casos, en .gz

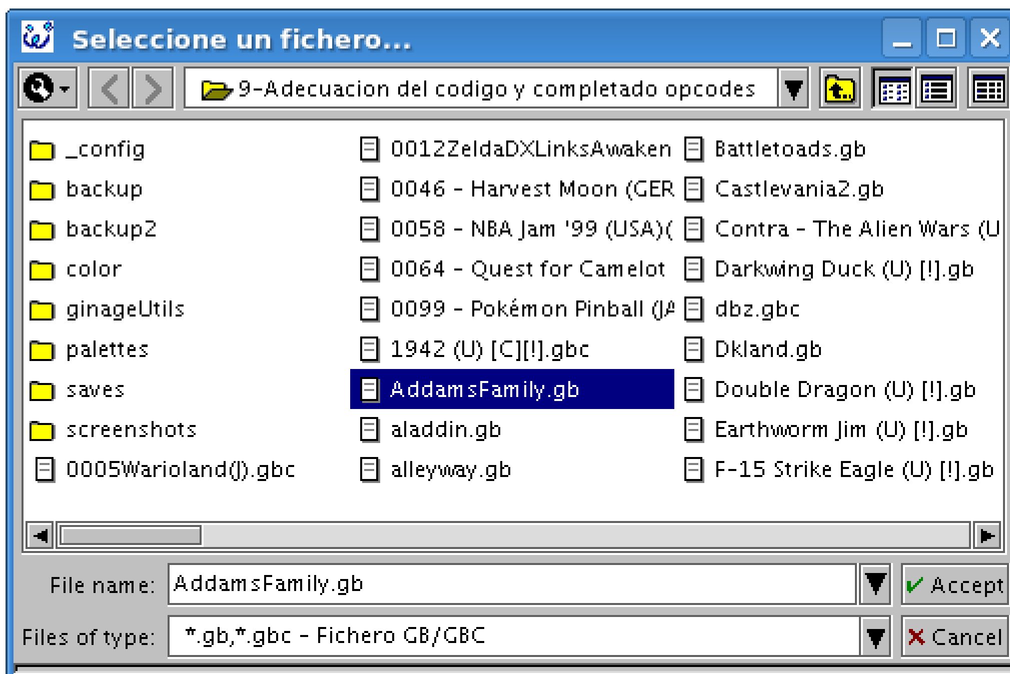


Ilustración B.3: El cuadro de diálogo abrir...

- Pausa: Permite para la emulación por un tiempo indefinido. Para reanudarla basta con volver a pulsarla. NOTA: La emulación es detenida temporalmente siempre que se muestra un menú a una ventana emergente.
- Reiniciar: Permite reiniciar la ROM actual como si se hubiera apagado y encendido de nuevo la Game Boy con el juego dentro (reinicio hardware).
- Configurar teclas...: Permite modificar las teclas asociadas del teclado asociadas a las de la Game Boy.

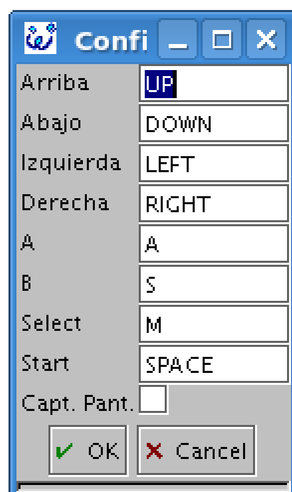


Ilustración B.4: El cuadro de dialogo para configurar teclas...

La tecla que se está modificando actualmente es mostrada en azul indicando que la siguiente pulsación de una tecla asignará ésta como equivalente a esa acción. La opción Capturas de Pantalla permite habilitar o deshabilitar la tecla F2 para capturar la pantalla en un instante dado.

- Idioma: Permite cambiar el idioma de la interfaz. Actualmente se admiten los idiomas Español o Ingles.
- Salir: Cierra el emulador, salvando el estado de la partida actual. También se puede cerrar la aplicación pulsando el la cruz de la ventana con el mismo efecto.

El menú video lo componen las siguientes opciones:

Ilustración B.5: El menú Video

- Tamaño de pantalla: Permite cambiar entre tamaño original (1x) y tamaño 2x.
- Frameskip: Permite habilitar o deshabilitar diferentes grados de frameskip (salto de frames) para mejorar la velocidad del emulador. El frameskip puede oscilar desde 0 (deshabilitado) hasta 5.
- Modo de color: Permite establecer el color de la imagen mostrada en 24 bits por pixel o en

256 colores indexados.

- Limitador de velocidad: Permite habilitar/deshabilitar los frames por segundo (FPS) máximos del emulador.
- Contador de frames: Permite habilitar/deshabilitar los frames por segundo (FPS) por pantalla.
- Editar paletas...: Ésta opción solo está disponible para juegos de Game Boy. Permite establecer los colores para las 3 paletas, hasta un total de 12 colores diferentes.

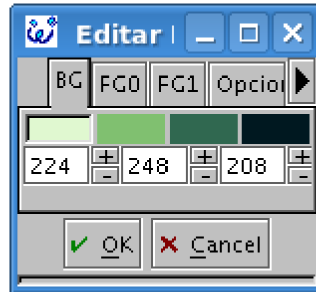


Ilustración B.6: El cuadro de dialogo para Editar paletas...

Cada pestaña selecciona una paleta y cada uno de los 4 recuadros de color permite seleccionar un color. Los 3 valores numéricos permiten modificar el color seleccionado para la paleta seleccionada. La cuarta pestaña permite establecer las paletas a los valores de la Game Boy original (tonos de verde) o la Game Boy Pocket (escala de grises).

Preguntas frecuentes (FAQ):

- P:Al ejecutar el emulador obtengo un error de tipo "java es un comando desconocido".
  R: Necesitas una MV de Java 1.4 o superior. Instala el JRE de Sun.

- P:Al ejecutar el emulador obtengo un error de mayor minor version mismatch.
  R: Tienes una MV anterior a la 1.4. Instala un JRE 1.4 o superior.

- P: ¿Como puedo hacer que el emulador se ejecute más rápido?.
- R: Habilita el tamaño de pantalla a 1x y prueba ambos modos de color. En algunos sistemas Windows la opción 256 colores da mejores resultados. También puedes cambiar la opción de frameskip para aumentar la velocidad (a costa de movimientos más bruscos).

- P: El juego X no funciona
  R: El emulador no es compatible con todos los juegos. La compatibilidades aproximadas son de un 86% para Game Boy y un 45% para Game Boy Color.

- P: Al cargar un .zip o .gz obtengo una ZipException por la linea de órdenes
- R: Ewe no es capaz de descomprimir correctamente algunos ficheros comprimidos. Prueba a descomprimir el juego y cargar el fichero .gb o .gbc directamente.

Anexo C: Resumen de códigos de operación del LR35902

Se puede encontrar un resumen de los opcodes en [Lloyd00] y en [Frohwein01]. Una descripción completa de los mismos, incluyendo los flags afectados y los ciclos de reloj empleados en cada una, está disponible en [AFF+98].

Tipo de instrucción	*Operación*
ADC x	A = A + x + CY
ADD x,y	x = x + y
AND x	A = A & x
BIT b,x	Z = (x[b] == 0)
CALL c,x	if(c) CALL x
CALL x	Salto a subrutina en PC + x: (PUSH PC, PC = PC + x)
CCF	CY = ~CY
CP x	Comparar A con X: if(A – X == 0) changeFlags()
CPL	Complemento a 1 de A (A = ~A)
DAA	Ajuste decimal de A (después de suma/resta de datos BCD)
DEC x	x = x - 1
DI	Deshabilitar interrupciones
EI	Habilitar interrupciones
HALT	Detener CPU hasta interrupción
INC x	x = x +1
JP c,d	if(c) JP d
JP d	PC = d
JR c,x	if(c) JR x
JR x	PC = PC + x
LD x,y	x = y
LDD x,y	x = y, HL = HL - 1
LDI x,y	x = y, HL = HL + 1
NOP	Operación nula
OR x	A = A | x
POP x	Sacar x de la cima de la pila actualizando SP: xl = (SP), xh = (SP + 1), SP = SP + 2
PUSH x	Poner x en la cima de la pila actualizando SP: (SP – 1) = xh,(SP – 2) = xl, SP = SP - 2
RES b,x	x[b] = 0
RET	Volver de interrupción: Pcl = (SP), Pch (SP + 1), SP = SP + 2
RET c	if(c) RET
RETI	RET, EI

Tipo de instrucción	*Operación*
RST x	Salto a subrutina en x: (PUSH PC, PC = x)
SBC x	A = A – x - CY
SCF	
SET b,x	x[b] = 1
STOP	Detener CPU hasta interrupción de joypad
SUB x	A = A - x
XOR x	A = A ^ x
SWAP x	
RL x	
RLC x	
RR c	
RRC x	
SLA x	
SRA x	
SRL x	

Tabla C.1: Resumen del ISA de la Game Boy

La CPU de la Game Boy aún presentando puntos en común con el procesador Z80, tiene algunos cambios en el ISA en parte propiciados por la eliminación del segundo juego de registros y parte de los flags. A continuación se resumen los opcodes que cambian entre un procesador y otro. Cualquier intento por parte del LR35902 de ejecutar los opcodes sin equivalente “-” provocan el cuelgue de la CPU en una Game Boy real.

Opcode	*Z80*	*LR35902*
08	EX AF,AF	LD (nn),SP
10	DJNZ PC+dd	STOP
22	LD (nn),HL	LDI (HL),A
2A	LD HL,(nn)	LDI A,(HL)
32	LD (nn),A	LDD (HL),A
3A	LD A,(nn)	LDD A,(HL)
D3	OUT (n),A	-
D9	EXX	RETI
DB		-
DD	<IX>	-
E0	RET PO	LD (FF00+n),A
E2	JP PO,nn	LD (FF00+C),A
E3	EX (SP),HL	-
E4	CALL P0,nn	-
E8	RET PE	ADD SP,dd
EA	JP PE,nn	LD (nn),A
EB	EX DE,HL	-
EC	CALL PE,nn	-

Opcode	Z80	LR35902
ED	<pref>	-
F0	RET P	LD A,(FF00+n)
F2	JP P,nn	LD A,(FF00+C)
F4	CALL P,nn	-
F8	RET M	LD HL,SP+dd
FA	JP M,nn	LD A,(nn)
FC	CALL M,nn	-
FD	<IY>	-
CB3X	SLL r/(HL)	SWAP r/(HL)

Tabla C.2: Comparativa de opcodes entre el Z80 y el LR35902

Anexo D: Estructura del cd

Dada la extensión del código fuente, se ha optado por incluir únicamente su versión electrónica en el cd adjunto. En el mismo puede encontrarse también la documentación utilizada durante la realización de éste proyecto. Para mayor comodidad, está organizado en directorios cuyo contenido es el siguiente:

- Código:
  - Versiones de desarrollo: Los resultados de cada iteración de la metodología evolutiva orientada a prototipos. Permite observar el seguimiento de las funcionalidades añadidas en cada versión. Las subcarpetas se encuentran numeradas en orden creciente de las versiones más antiguas a las más recientes. Cada subcarpeta contiene un fichero léeme que describe los problemas detectados y resueltos y las nuevas mejoras introducidas.
  - Versión Final: La versión final en jar junto a las librerías de Ewe modificadas y optimizado para java 1.4.
  - Versión de pruebas de rendimiento: La versión que se usó para las pruebas de rendimiento. Entre otras cosas, muestra al cerrarse los milisegundos por frame.
  - Modificaciones de librería Ewe:

- Documentación:
  - PFC: Contiene el presente documento en formatos pdf y odt
  - resultados de pruebas de rendimiento: Los resultados completos de las pruebas de rendimiento:
    - datos_de_profiling: Los resultados en ficheros html obtenidos del plugin de profiling para Eclipse tanto para el consumo de memoria como el consumo de tiempo por función
    - comparativa de eficiencia: El documento donde se ha recopilado la información de varios betatesters que ayudaron en las pruebas de eficiencia con sus ordenadores.

- Javadoc: Contiene la documentación de las clases generada con javadoc.

- Documentación de apoyo:
  - Principal: La documentación más usada durante el desarrollo, relativa al funcionamiento interno de la GB o de otros documentos sobre emuladores.
  - Otra documentación: El resto de documentación también útil pero consultada en menor medida.