

UNIVERSIDAD DE ALMERIA
ESCUELA SUPERIOR DE INGENIERÍA
**Desarrollo de un
juego para la consola
clásica Game Boy
Advance**

Curso: 2020/2021

Alumno/a:

Samuel Martín Vargas Giagnocavo

Director/es:
Juana López Redondo
Nicolás Calvo Cruz

Agradecimientos

Siempre he considerado que toda la gente que he conocido y con la que he trabajado durante los últimos años me ha ayudado de una forma u otra, tanto en mi formación profesional como en mi formación como persona. En condiciones normales esta sección del trabajo se habría limitado a un simple agradecimiento a todos los compañeros y profesores que he conocido durante la carrera y educación primaria y secundaria, incluyendo por supuesto a familiares y amigos. Sin embargo, he tenido la suerte de no vivir en “condiciones normales” y por ello me veo forzado a resaltar, con permiso de las personas que no serán mencionadas en esta sección, a aquellas que considero que han tenido un papel importante en el proceso de realizar el Trabajo Fin de Grado y la carrera de Ingeniería Informática.

Entre las personas que quiero destacar, me gustaría agradecer en especial a los dos directores de mi Trabajo Fin de Grado: Juana López Redondo y Nicolás Calvo Cruz. Agradezco enormemente la confianza, paciencia (especialmente al realizar el informe) y apoyo de mis dos directores, además de las continuas observaciones y consejos proporcionados que han mejorado la calidad del trabajo. También tengo que agradecer la oportunidad de abordar una temática distinta (y si se me permite, más interesante) a las demás, la cual no solo me ha motivado al realizar el trabajo, sino que ha despertado un interés en este área que seguramente no habría descubierto de no haber realizado el trabajo.

Además, me gustaría agradecer la asombrosa paciencia de mis padres, Antonio y Cynthia. El apoyo mostrado junto con todo lo que he adquirido de ellos, tanto conocimientos como valores, será algo que nunca podré agradecer de más.

Espero que el trabajo realizado esté a la altura del apoyo recibido de todas las personas mencionadas anteriormente. Confío también en que, si he olvidado a alguien, sepa perdonármelo. Muchas gracias a todos.

ÍNDICE GENERAL

	Página
1. Introducción	1
1.1. Game Boy Advance	2
1.2. Explicación y objetivos del proyecto	3
1.3. Justificación e Intereses	3
1.4. Planificación del proyecto	4
1.5. Recursos utilizados	8
1.5.1. Lenguajes de programación	8
1.5.2. Control de versiones	8
1.5.3. Editores	8
1.5.4. Herramientas gráficas y de audio	8
1.5.5. Herramientas de planificación	9
1.5.6. Emuladores	9
1.5.7. Hardware	9
1.6. Estructura del proyecto	10
2. Arquitectura de la Game Boy Advance	13
2.1. Procesador ARM7TDMI	16
2.2. Coprocesador Sharp SM83	20
2.3. El registro I/O	21
2.3.1. Registros LCD	21
2.3.1.1. DISPCNT	22
2.3.1.2. DISPSTAT	23
2.3.1.3. Configuración de fondos	24
2.3.1.4. Registros de efectos especiales	25
2.3.2. Registros de sonido	27
2.3.2.1. Modos analógicos	28
2.3.2.2. Modos digitales	30
2.3.3. Registros para <i>Direct Memory Access</i>	32
2.3.4. Registros de temporización	33

2.3.5. Registros de comunicación serie	34
2.3.6. Registros de los botones	36
2.3.7. Registros de interrupciones	37
2.4. Picture Processing Unit (PPU)	38
2.4.1. Sprites	42
2.4.1.1. Atributos	43
2.4.1.2. Matriz de transformación	47
2.4.2. Bitmaps	51
2.4.2.1. Modo 3	52
2.4.2.2. Modo 4	52
2.4.2.3. Modo 5	53
2.4.3. Fondos	53
2.4.4. Efectos especiales	56
2.4.4.1. Blending	56
2.4.4.2. Efectos fade	57
2.4.4.3. Modo mosaico	57
2.4.4.4. Modo ventana	58
3. Diseño y desarrollo del juego	59
3.1. Especificaciones	61
3.1.1. Requisitos generales	61
3.1.2. Casos de uso	63
3.1.3. Requisitos funcionales	63
3.1.4. Requisitos de información	67
3.1.5. Requisitos no funcionales	68
3.2. Desarrollo	69
3.2.1. Menú	70
3.2.2. Nivel del juego	70
3.3. Aspectos destacados del desarrollo	76
3.3.1. Pruebas de rendimiento	76
3.3.2. Sincronización vertical	78
3.3.3. Operaciones de punto flotante	79
3.3.4. Limitaciones de lectura y escritura	80
3.3.5. Texto	80
3.3.6. Preparación de los archivos binarios	80
4. Resultado final	83
4.1. YAP - Yet Another Platformer	83
4.2. Personajes	84

ÍNDICE GENERAL

4.3. <i>Title screens</i> , menús y niveles	86
4.4. Música	89
4.5. Indicadores in-game	89
4.6. Evaluación del juego en el hardware original	90
5. Conclusiones del proyecto	93
5.1. Trabajo futuro	93
Bibliografía	94
A. Contenido adicional	97
A.1. Estructura del proyecto	97
A.2. Personajes	98
A.3. Registros	102
A.4. Escenas	105
A.5. Automatización mediante scripts de BASH	110
A.6. Funciones de la BIOS	111

ÍNDICE DE FIGURAS

1.1.	Game Boy Advance, Game Boy Advance SP y Game Boy Advance Micro.	2
1.2.	Metodología ágil	5
1.3.	Diagrama de Gantt.	7
1.4.	Consola retrocompatible con la Game Boy Advance, la Nintendo DS Lite.	9
1.5.	Cartucho <i>Supercard</i>	10
2.1.	El circuito de la Game Boy Advance [1].	14
2.2.	Distribución de la memoria de la Game Boy Advance [1].	15
2.3.	La PPU de la Game Boy Advance.	16
2.4.	Diagrama simplificado del circuito de la Game Boy Advance [1].	16
2.5.	Registros por modo del procesador ARM7TDMI.	18
2.6.	Registros por modo del procesador ARM7TDMI cuando este utiliza el set de instrucciones THUMB.	19
2.7.	Segmentación de instrucciones de la Game Boy Advance.	19
2.8.	Distribución del EmbeddedICE-RT y controlador TAP en el diagrama de bloques del ARM7TDMI.	20
2.9.	Distribución 1D de los <i>tiles</i> en la OAM.	22
2.10.	Distribución 2D de los <i>tiles</i> en la OAM.	22
2.11.	<i>Tileset</i> que se utilizará para crear el fondo.	39
2.12.	Relleno del canvas a partir de <i>tiles</i>	39
2.13.	Creación de un fondo a partir del <i>tileset</i>	40
2.14.	Distintos <i>tiles</i> utilizados.	40
2.15.	Coordenadas de la pantalla.	41
2.16.	Los períodos HBlank y VBlank en la pantalla de la GBA. Imagen de [2].	42
2.17.	Distribución de los atributos en la OAM.	43
2.18.	Resultado de no ocultar los <i>sprites</i> no utilizados.	45
2.19.	Demostración de los <i>flags</i> para invertir un <i>sprite</i> de diversas maneras.	46
2.20.	Utilizando los mismos <i>tiles</i> se muestran diferentes <i>sprites</i> alternando la paleta de colores de cada uno.	47

2.21. Rotación de <i>sprites</i> por medio de la matriz de transformación.	49
2.22. Escalado <i>sprites</i> por medio de una matriz de transformación.	49
2.23. Rotación y escalado de <i>sprites</i> por medio de la matriz de transformación.	50
2.24. <i>Sprite</i> que sobrepasa el área permitida.	51
2.25. Demostración práctica del uso del Modo 3.	52
2.26. Ejemplo práctico del modo 5.	53
2.27. Fondo 32x32.	55
2.28. Fondo 32x32.	55
2.29. Matriz de transformación sobre el fondo de la Figura 2.28.	56
2.30. Efecto conseguido configurando el efecto <i>blending</i> de la consola.	57
2.31. Demostración del cambio de intensidad a través de BLDCNT y BLDY.	57
2.32. Sprite renderizado con efecto “mosaico” (izquierda), junto a su representación sin alterar (derecha).	58
2.33. Demostración del funcionamiento de una ventana.	58
3.1. Nidhogg, un juego para ordenador.	59
3.2. Assets para el primer prototipo del juego.	59
3.3. Funcionalidades por implementar.	60
3.4. Segundo prototipo del juego.	60
3.5. Arte escogido para el desarrollo del juego.	61
3.6. Diagrama de casos de uso.	63
3.7. Diagrama de flujo del menú principal.	70
3.8. Diagrama de flujo para cada nivel del juego.	71
3.9. Diagrama de flujo para el menú in-game.	72
3.10. Interfaz que utiliza No\$GBA para mostrar el uso de la CPU.	76
3.11. El valor del temporizador en la esquina superior izquierda.	77
3.12. Resultados de la prueba de rendimiento.	78
3.13. Desincronización vertical.	78
3.14. Rendimiento de las dos alternativas disponibles.	79
3.15. Archivos generados por el <i>script</i>	81
3.16. Regla del Makefile.	81
4.1. Versión publicada del juego.	83
4.2. Logo del juego <i>Yet Another Platformer</i>	83
4.3. Los <i>sprites</i> del personaje principal.	85
4.4. Las dos variantes del personaje principal.	85

ÍNDICE DE FIGURAS

4.5. Los <i>sprites</i> del primer enemigo.	85
4.6. Los <i>sprites</i> del segundo enemigo.	86
4.7. Enemigos de uno de los niveles.	86
4.8. <i>Title screen</i>	87
4.9. Menú principal del juego.	87
4.10. Escena de controles del juego.	88
4.11. Primera parte del nivel.	88
4.12. Segunda parte del nivel.	89
4.13. Tres corazones indicando la vida del jugador.	90
4.14. Software para meter juegos en el cartucho.	90
4.15. El juego se ejecuta en la Nintendo DS.	91
5.1. Prototipo del nuevo sistema de selección.	94
A.1. Estructura del proyecto.	97
A.2. Variante azul corriendo.	98
A.3. Variante azul iniciando la transformación.	98
A.4. Últimos fotogramas de la transformación de la variante azul a la dorada.	99
A.5. Variante dorada.	99
A.6. Ataque del personaje principal. además se observa la muerte del “monstruo verde”.	99
A.7. Variante dorada del personaje principal junto a uno de los enemigos.	100
A.8. Ataque del personaje principal. Además se observa la muerte de la “mosca”	100
A.9. El personaje principal saltando.	100
A.10. El personaje principal cayendo.	101
A.11. El personaje principal desapareciendo al no tener vida.	101
A.12. Registros LCD.	102
A.13. Registros de Sonido.	103
A.14. Registros <i>Direct Memory Access</i>	103
A.15. Registros de Temporización.	103
A.16. Registros de Comunicación Serial (1).	104
A.17. Registros de los botones.	104
A.18. Registros de Comunicación Serial (2).	104
A.19. Registros de Interrupciones.	104
A.20. Créditos iniciales (1).	105
A.21. Créditos iniciales (2).	105

A.22.Menú ingame.	106
A.23.Segundo nivel (1).	106
A.24.Segundo nivel (2).	107
A.25.Tercer nivel (1).	108
A.26.Tercer nivel (2).	109

ÍNDICE DE TABLAS

1.1.	Temporización por iteración.	6
2.1.	Mapa de memoria de la Game Boy Advance.	14
2.2.	Registro <i>Display Control</i> .	23
2.3.	Registro <i>Display Status</i> .	23
2.4.	Registro de control de fondos.	24
2.5.	Registros de cada uno de los fondos (1).	25
2.6.	Registros de cada uno de los fondos (2).	25
2.7.	Formato del valor de referencia de los fondos utilizando la matriz de transformación.	25
2.8.	Registros dedicados de los modos 1 y 2.	29
2.9.	Frecuencias disponibles para generar ruido en el modo 4.	30
2.10.	Registros fuente, destino y de control de DMA.	33
2.11.	Registros de control y datos de los temporizadores.	34
2.12.	Registros I/O para el modo multijugador.	36
2.13.	Los botones en el registro KEYINPUT.	36
2.14.	Los botones en el registro KEYCNT.	37
2.15.	Interrupciones disponibles en el sistema.	37
2.16.	Distribución de los <i>tiles</i> en la VRAM.	41
2.17.	Color de 15 bits.	42
2.18.	Tamaño del <i>sprite</i> según los valores de {F-E} en los atributos 0 y 1.	44
2.19.	Atributo 0.	45
2.20.	Atributo 1.	46
2.21.	Atributo 2.	47
2.22.	Modos bitmap de la Game Boy Advance. Tabla basada en [3]	52
2.23.	Modos para los fondos de la Game Boy Advance.	54
2.24.	Tamaños disponibles para cada uno de los fondos.	54
3.1.	Objetivo 1.	61

3.2. Objetivo 2.	62
3.3. Objetivo 3.	62
3.4. Objetivo 4.	62
3.5. Objetivo 5.	62
3.6. Perfil de Usuario 1.	63
3.7. Actor 1.	63
3.8. Requisito funcional 1.	64
3.9. Requisito funcional 2.	64
3.10. Requisito funcional 3.	65
3.11. Requisito funcional 4.	65
3.12. Requisito funcional 5.	66
3.13. Requisito funcional 6.	66
3.14. Requisito funcional 7.	67
3.15. Requisito de información 1.	67
3.16. Requisito no funcional 1.	68
3.17. Requisito no funcional 2.	68
3.18. Requisito no funcional 3.	69
3.19. Requisito no funcional 4.	69
3.20. Requisito no funcional 5.	69
A.1. Funciones de la BIOS a utilizar con la instrucción <i>swi</i> (1).	111
A.2. Funciones de la BIOS a utilizar con la instrucción <i>swi</i> (2).	112
A.3. Funciones de la BIOS a utilizar con la instrucción <i>swi</i> (3).	112

LISTADOS

3.1. Fragmento de código de la librería TONC.	73
3.2. Tomando medidas mediante temporizadores.	77
3.3. Implementación de VSync mediante dos bucles.	78
3.4. Ejemplo de un cálculo con números de coma fija.	80
A.1. Script (palettegen.sh) encargado de generar la paleta de colores para todas las imágenes proporcionadas.	110
A.2. Script (tilesgen.sh) encargado de generar un <i>tileset</i> a partir de cada grupo de imágenes proporcionado. También hay que incluir la paleta generada anteriormente.	110
A.3. Script (mapgen.sh) encargado de generar un mapa a partir de un <i>tileset</i> , paleta e imagen.	110
A.4. Script encargado de procesar todas las imágenes utilizando palettegen.sh, tilesgen.sh y mapgen.sh.	111

ABREVIATURAS

bpp	Bits per pixel
bps	Bits per second
CBG	Color Game Boy
DMG	Display Monochrome Game Boy
FIFO	First In First Out
FPU	Floating Point Unit
GBA	Game Boy Advance
OAM	Object Attribute Memory
ROM	Read-Only Memory
SIO	Serial Input/Output
VRAM	Video RAM

GLOSARIO

HBlank	Simboliza el período que se produce entre cada línea refrescada por pantalla
Homebrew	Juegos o aplicaciones creadas por aficionados para plataformas cerradas
Renderizar	Proceso que involucra generar una imagen a partir de modelos (2D o 3D) y/o imágenes base
Sprint	El tiempo transcurrido para completar una nueva versión del producto
Sprites	Describe cualquier entidad única que se pueda mover dentro de un juego. Suelen ser personajes y objetos.
Tile	Pequeña imagen (en el caso de la Game Boy Advance, de 8x8) que sirve para posteriormente referenciarla en un mapa
Transformaciones de corte	Operación geométrica donde una parte del cuerpo se desliza paralelamente con respecto a otra
VBlank	Simboliza el período que se produce entre cada fotograma refrescado
VCount	Simboliza el número de líneas verticales refrescadas en la Game Boy Advance

1 INTRODUCCIÓN

Actualmente, los usuarios que consumimos contenido multimedia contamos con una amplia y actualizada selección a nuestro alcance. Sin embargo, y cada vez más, muchas personas recurren a contenidos, formatos y diseños que hace poco se consideraban anticuados. Ejemplos de este fenómeno, que muestran la preferencia de los usuarios por lo *retro*, incluyen el resurgimiento de los discos de vinilo¹ y el deseo de numerosos aficionados a la fotografía de disfrutar de su *hobby* a través de medios analógicos². La demanda por parte del público de revisitar y reconsiderar lo “antiguo” ha incrementado considerablemente.

Una de las áreas que más se ha visto beneficiada ha sido la de los videojuegos. Hardware técnicamente desfasado y software con décadas sin soporte oficial han pasado a tener una segunda vida en manos de la comunidad de videojuegos, la cual ha facilitado toda la información y documentación necesaria para que nuevos usuarios puedan participar y jugar. Entre los movimientos que han aparecido se encuentra el *homebrew*, software creado por los aficionados para hardware propietario.

Un dato que refuerza lo que comentamos en el párrafo anterior es la respuesta a la demanda por parte de compañías como Nintendo o Sony. Ambas sacaron al mercado revisiones de consolas como las NES, SNES y PlayStation original junto con recopilaciones de juegos tradicionales de los 80 y 90. Cabe mencionar que las dos revisiones de Nintendo tuvieron un gran éxito.

Inevitablemente, la reaparición de este tipo de tecnología puede llegar a generar dudas sobre el porqué de este fenómeno. Una de las razones principales, además de contar con una comunidad dedicada [4], es la relativa facilidad para iniciarse: Mientras que para el hardware más actual es necesario invertir tiempo y dinero en adquirir (y aprender a utilizar) SDK's oficiales y frameworks, el hardware antiguo ofrece gran cantidad de documentación pública y, en la mayoría de casos, numerosos emuladores que además de facilitar el desarrollo de juegos, ofrecen al jugador diversas formas de jugar. Esto último es otra razón por la que es llamativo desarrollar para este tipo de hardware. Por ejemplo, un juego desarrollado para la NES estará disponible para una gran variedad de sistemas (cualquiera que cuente con un emulador de NES en este caso).

Sin embargo, además del punto de vista práctico, desarrollar para máquinas *retro* también puede resultar atractivo con fines educativos. Un ejemplo de ello puede ser [5], donde los autores exploran el uso de micro-proyectos de CHIP-8, una plataforma de videojuegos

¹<https://www.xataka.com/musica/vinilo-resurge-definitivamente-34-anos-despues-logra-vender-que-formato-cd-ee-uu-espana-tiene-40-cuota>

²<https://garage.hp.com/us/en/modern-life/analog-film-photography-trend-low-tech-photos.html>

de los 80, para aplicarlos en distintas asignaturas del grado de Ingeniería Informática. Más ejemplos de aplicación didáctica pueden encontrarse en [6], donde el autor habla sobre el uso de la Game Boy Advance, la popular consola portátil de principios del 2000, para enseñar Arquitectura de Computadores y [4], donde los autores diseñan un curso alrededor de la Game Boy Advance y su sucesora, la Nintendo DS, para enseñar programación a sus alumnos. De forma adicional, en [7] se comenta, entre otras cosas, el rol que tiene el desarrollo *amateur* de videojuegos al adquirir nuevas habilidades y conocimientos. Programar para este tipo de dispositivos requiere un cierto conocimiento del hardware que en otras áreas no es necesario. Para sacarle el máximo partido al hardware (especialmente cuando se trata de hardware con especificaciones limitadas) se intenta evitar la programación a altos niveles de abstracción.

Es lo descrito en el párrafo anterior lo que nos lleva al dispositivo que utilizaremos durante este trabajo, la **Game Boy Advance**.

1.1 GAME BOY ADVANCE

Con el increíble éxito que tuvo su predecesora, la Game Boy Color, Nintendo decidió sacar una nueva versión de la consola portátil en 2001, mejorando las especificaciones técnicas con las que contaba el modelo anterior y empleando una arquitectura completamente nueva. Hasta el 2010, año en el que Nintendo descatalogó el dispositivo, se lanzaron una gran cantidad de juegos, aproximadamente 1500³. De forma adicional, se introdujeron dos modelos actualizados del dispositivo original, la Game Boy Advance SP y la Game Boy Advance Micro. Estas conservaban las especificaciones base de la consola pero mejoraban aspectos como la pantalla LCD. Las tres versiones se pueden ver en la Figura 1.1. Combinando todos los modelos, se vendieron en torno a 80 millones de unidades⁴, convirtiéndose así en una de las consolas más exitosas comercialmente.

Entre los factores que contribuyeron a su éxito se incluyen la retrocompatibilidad con la generación de consolas previa de Nintendo, su precio asequible, la calidad de los juegos que desarrollaban las *third-party*⁵ o la propia Nintendo y la relativa novedad que todavía tenía el mercado de consolas portátiles. Gracias a todos estos factores, la compañía japonesa prá-



(a) Game Boy Advance

(b) Game Boy Advance SP

(c) Game Boy Advance Micro

Figura 1.1: Game Boy Advance, Game Boy Advance SP y Game Boy Advance Micro.

³https://nintendo.fandom.com/wiki/List_of_Game_Boy_Advance_games

⁴<https://www.nintendo.co.jp/ir/pdf/2007/070426e.pdf#page=21>

⁵Se conoce como *third-party* a los juegos desarrollados por una entidad ajena a la compañía propietaria de la plataforma en la que se publican.

1.2. EXPLICACIÓN Y OBJETIVOS DEL PROYECTO

ticamente disfrutó de un monopolio en el mundo de las consolas portátiles⁶ (esto se aplica tanto para la Game Boy original como para la Advance).

1.2 EXPLICACIÓN Y OBJETIVOS DEL PROYECTO

En este Trabajo Fin de Grado se programará un **juego para la consola Game Boy Advance**, que funcionará en el hardware original⁷ y en emuladores. Es importante destacar este aspecto porque el hecho de que un juego funcione en un emulador no garantiza que funcione también en el hardware original. Comentaremos algunos ejemplos de situaciones de este tipo a lo largo del trabajo.

Para poder realizar el proyecto correctamente se precisarán **conocimientos del lenguaje de programación C aplicados a máquinas de propósito específico**. Estos deberán complementarse con un **conocimiento de la arquitectura y funciones propias de la Game Boy Advance**. De forma adicional, dado que el trabajo se especializa en un área como es la de los videojuegos, será necesario **familiarizarse con las técnicas profesionales de desarrollo de videojuegos**. Se espera poder conseguir un producto acorde con la calidad que ha ofrecido durante años la comunidad *homebrew*. Algo a destacar en este aspecto es la integración de archivos multimedia para su uso en el producto final. Entre los contenidos que se utilizarán se incluyen tanto *sprites*⁸ como pistas de audio. Finalmente, el trabajo también requerirá del **uso de métricas de rendimiento** durante el desarrollo para poder evaluar el software escrito y tomar decisiones en base a los resultados obtenidos.

En la siguiente lista se muestra un resumen de los objetivos parciales anteriormente planteados, y derivados del objetivo principal de crear un juego para la Game Boy Advance (abreviada normalmente como GBA):

- Aprendizaje de la arquitectura de la Game Boy Advance.
- Adquisición de conocimientos de desarrollo para máquinas de propósito específico.
- Ampliación práctica del conocimiento sobre programación en lenguaje C.
- Familiarización con las técnicas profesionales de desarrollo de videojuegos.
- Aplicación de técnicas de programación eficiente apoyadas en medidas de rendimiento (benchmarking) para comparar estrategias y soluciones a un mismo problema.

1.3 JUSTIFICACIÓN E INTERESES

La idea de desarrollar un juego para una consola antigua, descatalogada desde hace más de una década, puede parecer un tanto peculiar como proyecto. Sin embargo, se puede

⁶<https://www.theverge.com/2019/4/19/18507409/nintendo-game-boy-competitors-nokia-sony-bandai>

⁷Se probará con una Nintendo DS Lite, consola retrocompatible con la Game Boy Advance.

⁸Imagen asignada a un objeto/personaje de un videojuego.

justificar en base a diversos motivos. Uno de ellos es la arquitectura ARM[8] de la máquina escogida, cuya relevancia no deja de aumentar actualmente. Otra razón es el atractivo docente de la propia consola, que permite estudiar, en un mismo dispositivo, muchos conceptos presentes en los planes de estudio del Grado en Ingeniería Informática [6]. Igualmente, el trabajo no deja de precisar del esfuerzo y dedicación que cualquier proyecto de software requiere. Las labores de planteamiento, realización y gestión siguen estando presentes, e incluso se pueden llegar a dificultar dadas las peculiaridades que presenta un dispositivo de este tipo. Para poder realizar un proyecto de este estilo, el programador tendrá que involucrarse tanto en los aspectos más técnicos que trae consigo la programación de un dispositivo empotrado, como el diseño de funcionalidades con las que el usuario interactuará y que son ajena a la programación (por lo menos de forma directa), como son el diseño de menús, personajes, niveles y controles del juego.

Además, el interés no solo procede de las dificultades que presentará la realización del trabajo, sino del dispositivo en sí. A título personal, el autor de este trabajo ha estado desde su infancia en contacto con consolas del fabricante nipón. Vivió incluso el final de la época de esplendor de la GBA. Por este motivo, poder desarrollar ahora software para esta plataforma supone un importante factor motivador. De forma complementaria, el desarrollo de software para sistemas empotrados siempre ha sido otro tema de interés para el autor dada su experiencia previa con microcontroladores ESPRESSIF[9] y Arduino. De hecho, desarrolló para el primero un pequeño juego, por lo que la transición a la Game Boy Advance se podría considerar el siguiente paso a seguir para continuar afianzando los conocimientos adquiridos.

Otro aspecto que motiva la realización de este trabajo, y que puede parecer contradictorio dada la naturaleza del dispositivo, es la disponibilidad del producto final para una infinidad de plataformas. Tal y como ya se mencionó, el software desarrollado podrá ejecutarse realmente en casi cualquier dispositivo actual gracias a la popularidad de la emulación.

En base a lo expuesto, podemos concluir que la realización del proyecto supone una motivación personal para el autor. Es además una propuesta bastante original en comparación a otros trabajos de este tipo aunque tenga las mismas exigencias a efectos prácticos. Igualmente, el resultado se podrá usar sin problemas en entornos actuales a pesar de estar destinado a una máquina descatalogada. Por consiguiente, se considera una forma perfecta de certificar los conocimientos y competencias adquiridas durante el Grado en Ingeniería Informática.

1.4 PLANIFICACIÓN DEL PROYECTO

En este apartado se mostrarán las tareas a realizar y su planificación para completar los objetivos propuestos en la sección 1.2. La realización de las tareas se hará de modo que se siga una **estrategia de desarrollo** similar al **Ágil** (metodología popular en la industria de videojuegos [10, 11]), en la que existirán varias iteraciones y prototipos del juego. Cada nueva iteración tendrá como base el *feedback* y conocimientos adquiridos en la iteración anterior con el objetivo de conseguir un producto final pulido y acorde con los objetivos establecidos. Las diferentes fases de cada iteración se pueden observar en la Figura 1.2.

En su forma más simplificada, cada iteración repetirá el proceso del modelo de desarrollo en cascada, con la diferencia de que la metodología ágil no requiere acabar completamente

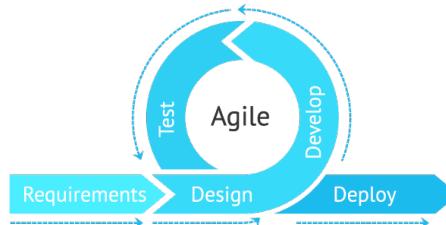


Figura 1.2: Metodología ágil. Imagen de devcom.com.

cada fase. Por lo tanto, una vez planteado el proyecto y sus especificaciones (*Requirements* en la Figura 1.2), las fases restantes se compondrán de las siguientes etapas:

1. Diseño: Teniendo en cuenta los objetivos del proyecto, se diseñarán nuevas especificaciones y funcionalidades a implementar en la iteración actual.
2. Desarrollo: Se implementará lo especificado en la fase anterior.
3. Pruebas: Se evaluará lo desarrollado en el punto 2, comprobando que funciona acorde a lo esperado, tanto a nivel de programación, como a nivel conceptual, siendo habitual la prueba por parte del usuario final.
4. Revisión: A pesar de que normalmente se incluye en la fase de Pruebas, esta fase merece una mención aparte. Teniendo en cuenta, los constantes cambios que se realizan en este tipo de proyectos, es necesario revisar lo desarrollado y decidir si el proyecto sigue acorde con la idea original. Los ajustes a realizar se identificarán en esta fase y se tendrán en cuenta para la próxima iteración.

Con una versión del producto acabada, los desarrolladores publicarán la iteración con la posibilidad de empezar a revisitar las fases descritas anteriormente para lanzar una versión mejorada o con más funcionalidades [11].

Dado que el proyecto se comenzó con un escaso conocimiento sobre las funcionalidades y arquitectura de la Game Boy Advance, se aprovecharon las iteraciones para conocer las diferentes opciones y características del dispositivo. Por ejemplo, en la primera iteración se analizaron las posibilidades a nivel gráfico que ofrece el dispositivo, mientras que durante la segunda iteración se exploraron las posibilidades que ofrece el procesador, a nivel de audio. En total, se utilizaron un total de 4 iteraciones, también denominadas *sprints*, para realizar el proyecto.

Al combinar los *sprints* junto con las demás tareas “no iterables” se obtiene un proyecto el cual ha requerido **300 horas en total**. Las tareas en cuestión se distribuyen de la siguiente manera:

1. **Estudio de la arquitectura de la Game Boy Advance (40 horas):** Estudio de la arquitectura con ayuda de la documentación recogida en la bibliografía.
2. **Asimilación de técnicas de programación específicas (18 horas):** Estudio de proyectos de código abierto para Game Boy Advance con los que aprender estrategias de desarrollo de calidad contrastada para esta plataforma.

3. **Sprint (49.25 horas):** Desarrollo del proyecto. A continuación, se desglosan y describen las tareas de cada uno de los 4 *sprints* que se realizan, para un total de 197 horas.

- **Reuniones con los tutores (20 horas):** Reuniones con el profesorado para definir los requisitos del proyecto, seguir su desarrollo y resolver posibles dudas.
- **Diseño de los niveles del juego (17 horas):** Estudio de niveles existentes en diferentes juegos para inspiración propia.
- **Diseño y creación de recursos (5 horas):** Elaboración, mediante herramientas como Adobe Photoshop o GIMP, de elementos del juego como sprites e iconos. Dada la poca familiaridad con el software de dibujo y la falta de experiencia en este campo, se propuso optar por diseños *Open Source* para realizar el juego. A pesar de trabajar con recursos ya existentes, se siguen necesitando 5 horas, dado que será necesario convertirlos a formatos que la consola pueda “entender”.
- **Implementación (105 horas):** Programación del juego deseado acorde a las especificaciones propuestas en las tareas de “Diseño”. Esta tarea es la más exigente, por lo que se le asignaron un mayor número de horas que a las restantes.
- **Evaluación del producto (50 horas):** Análisis del resultado obtenido en la tarea de “Implementación” para la identificación y solución de errores y aplicación de mejoras.

4. **Elaboración de memoria (45 horas):** Redacción de un informe que describa todo el trabajo desarrollado en las distintas facetas.

La temporización de las tareas mostradas se puede observar en la Tabla 1.1. Por su parte, el diagrama de Gantt, que refleja visualmente los datos mostrados en la Tabla 1.1, quedaría como se muestra en la Figura 1.3. La Tabla se ha calculado estimando unas 2 horas por día dedicadas al proyecto.

Tarea	Duración	Inicio	Fin
Estudio arquitectura GBA	40 horas	Lunes 08/02/21	Sábado 27/02/21
Estudio técnicas de programación	18 horas	Domingo 28/02/21	Lunes 08/03/21
Sprint 1	49.25 horas	Martes 09/03/21	Sábado 03/04/21
Sprint 2	49.25 horas	Sábado 03/04/21	Miércoles 28/04/21
Sprint 3	49.25 horas	Miércoles 28/04/21	Domingo 23/05/21
Sprint 4	49.25 horas	Domingo 23/05/21	Jueves 17/06/21
Elaboración del informe	45 horas	Jueves 17/06/21	Viernes 09/07/21
Entrega	-		Viernes 09/07/21

Tabla 1.1: Temporización por iteración.

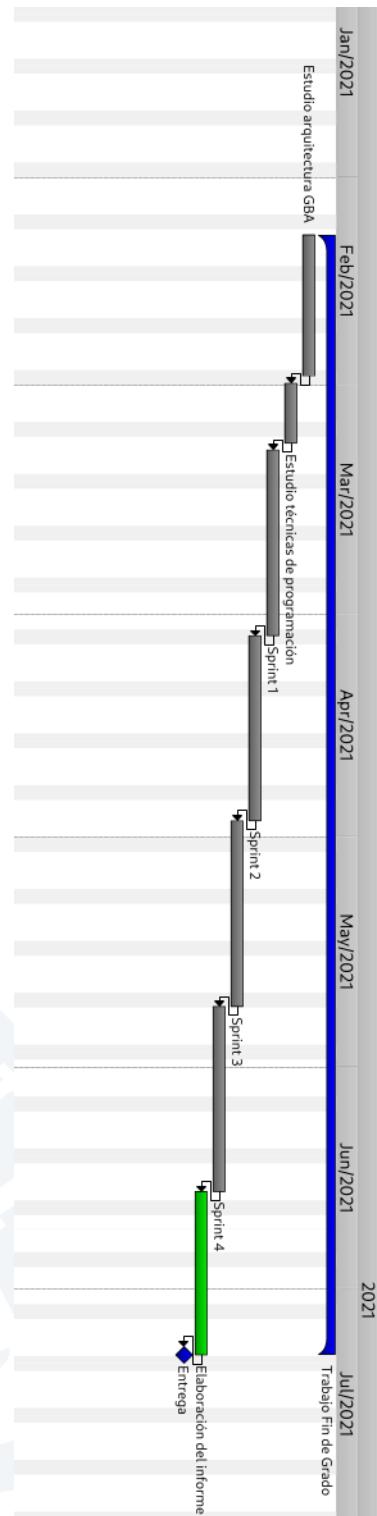


Figura 1.3: Diagrama de Gantt.

1.5 RECURSOS UTILIZADOS

En esta sección se describirán las herramientas utilizadas para poder realizar satisfactoriamente el proyecto. Se ha considerado oportuno clasificarlas en los subapartados de Lenguajes de programación, Control de versiones, Editores, Herramientas gráficas y de audio, Herramientas de planificación, Emuladores y Hardware.

1.5.1 Lenguajes de programación

Como se mencionó en el punto 1.2, para llevar a cabo el proyecto se va a **programar en C**. Aunque actualmente existen varias formas de desarrollar un juego para la Game Boy Advance (a partir de C++ y Rust por ejemplo), se seguirá apostando por C, dada la documentación y su uso extendido en la comunidad *homebrew*. El estándar utilizado para el proyecto será el “GNU17”, el que usa GCC por defecto⁹, que es también el compilador empleado en este proyecto. Además, para simplificar el proceso de compilación, se hará uso de la utilidad make.

1.5.2 Control de versiones

Para gestionar el código se hará uso del popular software de control de versiones **Git**. Además, para alojar el proyecto se utilizará un servidor privado (de modalidad gratuita) de **GitHub**.

1.5.3 Editores

Dada la necesidad de programar y escribir el informe final en diferentes máquinas, se optará por utilizar editores multiplataforma. Para programar se hará uso de **Vim** y **Visual Studio Code**. El primero es adecuado para editar rápidamente archivos y automatizar ciertas tareas, mientras que el segundo es muy eficaz para trabajar simultáneamente con diversos archivos.

En cuanto a la escritura del informe, se hará uso de **TeXmaker**, que proporciona una interfaz simple y cómoda para personas que no cuenten con mucha experiencia escribiendo en **LATEX**.

1.5.4 Herramientas gráficas y de audio

Teniendo en cuenta la naturaleza del dispositivo para el que se programará, será necesario convertir los diferentes recursos gráficos a un formato especial. **Grit** y **SuperFamicomv**, herramientas creadas específicamente para la conversión de imágenes a un formato reconocible por la Game Boy Advance, se utilizarán para adaptar recursos conocidos como *sprite sheets* y *tile sets* (los cuales se explicarán y se verán con más detenimiento en el punto 2.4).

Independientemente de si los recursos gráficos son de terceros o propios, la mayoría de las veces será necesario retocar las imágenes antes de convertirlas. Para ello será necesario **Pixelorama**, editor específicamente diseñado para trabajar con *sprites*.

⁹<https://gcc.gnu.org/onlinedocs/gcc-11.1.0/gcc/Standards.html>

1.5. RECURSOS UTILIZADOS

Para generar los fondos que utilizará el juego, se hará uso de **Tiled**, uno de los editores de niveles *open source* más populares. Su uso se mostrará en el punto 2.4.

Por último, para convertir música utilizada para el juego en un formato reconocible para el dispositivo se utilizará **Audacity**.

1.5.5 Herramientas de planificación

Para planificar cada una de las tareas del proyecto así como calcular su distribución se utilizará **Plan**, de la suite de aplicaciones Calibra. Por otro lado, **Trello** se utilizará para clasificar las tareas del proyecto en cada uno de los *sprints* a realizar.

1.5.6 Emuladores

Tras explicar las herramientas necesarias para poder desarrollar el proyecto, pasamos a aquellas que nos permitirán probarlo y ejecutarlo. Las escogidas para este proyecto son los emuladores mGBA y No\$GBA. Del primero se destaca su activo desarrollo y el ser multiplataforma, mientras que del segundo se pueden resaltar las opciones de depuración que ofrece al usuario.

1.5.7 Hardware

Finalmente, pasamos a los recursos que ejecutarán las versiones finales de nuestro proyecto (también las versiones de desarrollo, pero con menor frecuencia). En este caso, al no disponer de una consola Game Boy Advance original, se hará uso de una consola retrocompatible, la **Nintendo DS Lite** (véase la Figura 1.4).

Lamentablemente, la consola no admite cargar directamente software *homebrew*. Por este motivo, se utilizará un cartucho **Supercard** (véase la Figura 1.5), que permitirá, mediante una tarjeta microSD, ejecutar el software generado.



Figura 1.4: Consola retrocompatible con la Game Boy Advance, la Nintendo DS Lite.



Figura 1.5: Cartucho *Supercard*.

1.6 ESTRUCTURA DEL PROYECTO

Este documento tiene como finalidad mostrar la solución desarrollada al problema propuesto, comentando a su vez las diversas características que tiene la Game Boy Advance. Dado que será necesario conocer varios conceptos antes de comentar el desarrollo del juego, la estructura del proyecto comenzará describiendo la arquitectura de la GBA y las opciones que ofrece.

En el Capítulo 2 se comentarán brevemente las características de los dos procesadores que utiliza la consola nipona y el papel que tiene cada uno en el dispositivo.

Seguidamente, se expondrán los registros principales y las funciones de cada uno. Entre ellas se encuentran, por ejemplo, la configuración de la pantalla y del sonido, las transferencias de acceso a directo a memoria y la comunicación con otros dispositivos.

Teniendo una idea de los registros que el desarrollador necesitará conocer para programar en el dispositivo, se explicarán los modos que ofrece la unidad de procesamiento de imágenes de la consola (*PPU* ó *Picture Processing Unit*). Se combinará la explicación de los diferentes modos con pequeñas demostraciones para facilitar la comprensión. Además, se introducirán al lector algunos conceptos matemáticos necesarios para comprender el funcionamiento de algunas de las funciones de la *PPU*.

Partiendo de la base previa, en el Capítulo 3, se detallarán los requisitos y especificaciones del proyecto, entre los que se incluirán varios diagramas. De forma adicional, se resaltarán algunos aspectos del desarrollo que distinguen a la Game Boy Advance con respecto a otros dispositivos.

En la fase final del informe, en concreto en el Capítulo 4, se presentará la solución desarrollada para el proyecto, describiéndose el funcionamiento del juego. Se mostrará también cómo se ha probado el software en el hardware real, detallando el procedimiento a seguir.

Una vez descrito el estado actual del juego, en el Capítulo 5 se expondrán las conclusiones detallando los conocimientos adquiridos y la experiencia de desarrollar un juego para la Game Boy Advance. Adicionalmente, se comentarán varias de las funcionalidades y mejoras

1.6. ESTRUCTURA DEL PROYECTO

descartadas para el proyecto en la fase de prototipado (de cada iteración) que podrán servir como líneas de trabajo en un futuro.

Finalmente, en los apéndices se incluirán información adicional de los registros y funciones del sistema, estructura del proyecto a nivel de código, varias imágenes mostrando el juego en funcionamiento y varios *scripts* utilizados para facilitar el desarrollo del producto final.

2 ARQUITECTURA DE LA GAME BOY ADVANCE

Con un procesador RISC como corazón del sistema, la Game Boy Advance tiene unas especificaciones modestas, similares a lo que veríamos en los dispositivos embebidos de hoy en día. El dispositivo cuenta con un unidad ARM7TDMI de 32 bits como procesador principal. Tiene también un coprocesador el cual fue utilizado en la iteración anterior de la consola, el Sharp SM83 (basado en el set de instrucciones 8080 de 8 bits). Esto permite a la consola jugar, de forma nativa, a juegos desarrollados para la Game Boy original [1].

Estos dos procesadores residen en el mismo SoC (véase en la Figura 2.1) y permiten a la consola ejecutarse en varios modos. Podemos distinguir los siguientes:

- El **modo ARM**, que hace que el procesador funcione a 16.78 MHz, manejando instrucciones de 32 bits.
- El **modo THUMB**, reservado para optimizar partes del código. En este modo la consola también funciona a una frecuencia de 16.78 MHz, pero solo acepta instrucciones de 16 bits.
- El **modo CGB** (Color Game Boy), que utiliza el coprocesador mencionado anteriormente para ejecutar código escrito para la Game Boy Color original (la frecuencia puede variar entre 4.2 MHz y 8.4MHz).
- El **modo DMG** (Dot Matrix Game), similar el modo anterior, aunque esta vez limitado a 4.2 MHz. Está reservado para su uso en juegos programados para la Game Boy con pantalla monocromática.

Es importante mencionar que ambos procesadores no pueden ejercer la función de procesador central simultáneamente. Por ende, el programador puede cambiar del modo ARM al THUMB y viceversa, pero no de un modo que use el procesador ARM7 a otro que utilice el Sharp SM83 [12].

La consola cuenta con el mapa de memoria detallado en la Tabla 2.1 que, dependiendo a que sección se accede, tiene buses de 32, 16 u 8 bits. La práctica de tener buses de diferentes tamaños es común en consolas, incluso de sobremesa como la PlayStation 2 [13].

Los espacios que aparecen en la tabla anterior, entre la ROM del sistema y la RAM externa (en concreto, 0x00004000-0x01FFFFFF), son regiones que no se utilizan o en las que el programador no tiene que intervenir de ninguna manera [2].

¹A pesar de tener disponible 256 KB, la Game Boy reserva en los últimos 256 bytes datos que gestiona internamente, como el vector de interrupciones.

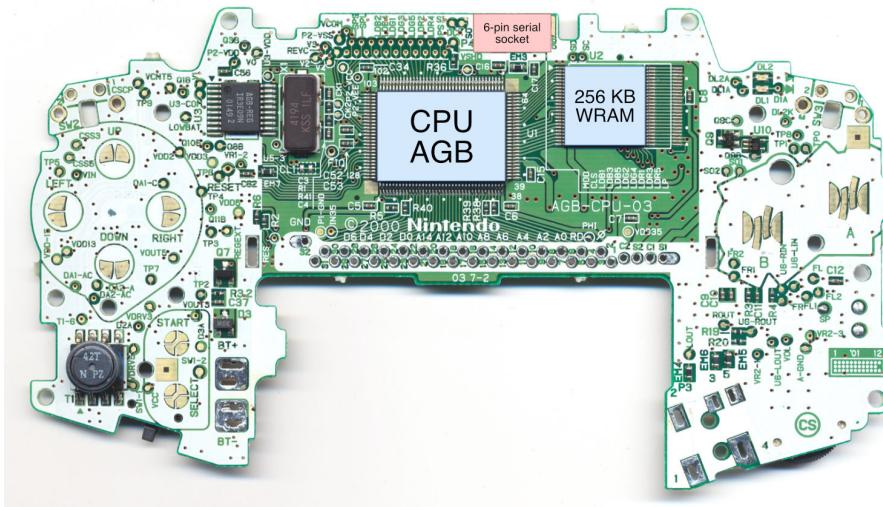


Figura 2.1: El circuito de la Game Boy Advance [1].

Nombre	Sección	Tamaño	Bus
ROM del sistema	0x00000000-0x00003FFF	16 KB	32 bits
RAM externa (On-board RAM)	0x02000000-0x0203FFFF	256 KB ¹	16 bits
RAM interna (On-chip RAM)	0x03000000-0x03007FFF	32 KB	32 bits
RAM de IO	0x04000000-0x040003FE	-	32 bits
RAM de paleta	0x05000000-0x050003FF	1 KB	16 bits
VRAM	0x06000000-0x06017FFF	96 KB	16 bits
Memoria de objetos	0x07000000-0x070003FF	1 KB	32 bits
ROM y flash del cartucho	0x08000000-0x0DFFFFFF	≤ 96 MB	16 bits
SRAM del cartucho	0x0E000000-0x0E00FFFF	≤ 64 KB	8 bits

Tabla 2.1: Mapa de memoria de la Game Boy Advance.

A continuación, comentaremos brevemente la función que se le da a cada sección.

- **ROM del sistema:** Encabezando el mapa de memoria de la consola encontramos la ROM del sistema, la cual guarda, como su nombre indica, código inmutable del sistema. Contiene por ejemplo la BIOS y rutinas que el programador puede utilizar en su juego. Entre las rutinas incluidas se pueden distinguir algunas para calcular operaciones aritméticas complejas y transformaciones afines, y otras para llevar a cabo funciones de descompresión, sonido, copia de memoria e interacción con otras consolas al conectarlas mediante el cable *link*² (función conocida como *multi-boot*).
- **RAM:** A continuación, después de la ROM, se encuentra la RAM de la consola (también conocida como EWRAM o External Work RAM), en la que se distinguen dos tipos. La primera región, posicionada en la dirección 0x02000000, reside fuera del SoC (véase la Figura 2.1). Cuenta con un tamaño de 256 KB y un bus de 16 bits por el cual podemos considerar esta sección como la RAM “lenta”. La siguiente región que encontramos en el mapa de memoria, la RAM interna, ofrece un bus completo de 32 bits, pero con un

²La Game Boy Advance permitía jugar con hasta 4 personas mediante una conexión por cable.

tamaño reducido (comparado con la RAM externa) de 32 KB. En la Figura 2.2 se muestra la distribución y localización de la IWRAM y EWRAM.

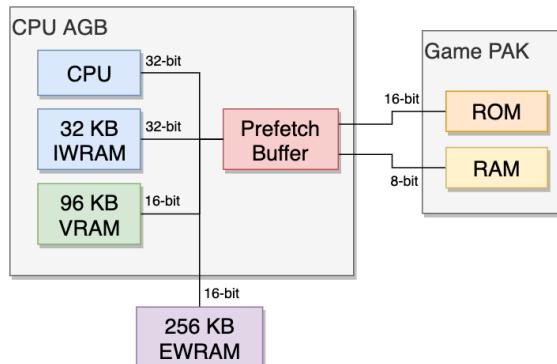


Figura 2.2: Distribución de la memoria de la Game Boy Advance [1].

- **Registro de control de entrada y salida:** En la siguiente sección aparece “mapeado” el registro de control de entrada y salida en el que el programador puede consultar información (como las teclas que el usuario presiona) y modificar la configuración de la consola. Por ejemplo, en esta región se encuentran los valores que determinarán el tipo de gráficos que utilizará el juego.
- **RAM de paleta:** La RAM de paleta, o “palette RAM” en inglés, es la sección de memoria que cubre el espacio necesario para albergar las paletas de colores de los diferentes modos de vídeo y gráficos disponibles.
- **VRAM:** En la VRAM se almacenan los índices de la paleta de colores para referenciarlos. Además de esta función en algunos modos de la Game Boy Advance, la VRAM también contiene los objetos o píxeles que aparecerán en pantalla. Más adelante se darán más detalles.
- **Memoria de Objetos:** La memoria de objetos (OAM, de "Object Attribute Memory" en inglés), al igual que la RAM de paleta, se utiliza en unos modos concretos y se encarga de guardar los datos de objetos a mostrar, informalmente nombrados por el anglicismo *sprites*. Estos “objetos” son, normalmente, el texto y personajes que aparecen en los juegos 2D. Algunos de los atributos que maneja esta región son la posición, rotación, escala y tamaño.

Las tres secciones descritas, RAM de paleta, VRAM y OAM, forman parte de la PPU (*Picture Processing Unit*) de la Game Boy Advance. Esta se encarga de combinar y procesar la información almacenada en dichas regiones (qué información procesa y cómo dependerá del modo de vídeo utilizado) para crear una imagen que el usuario pueda ver. La distribución de la PPU se observa en la Figura 2.3.

- **ROM y SRAM del cartucho:** La última región en el mapa de memoria es la vinculada con el cartucho del juego, que se introduce en la ranura que ofrece la consola y funciona mediante una conexión paralela. La ejecución del juego empieza en esta dirección de

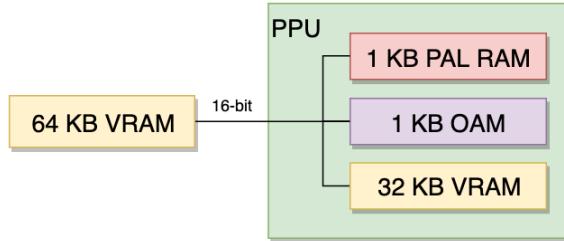


Figura 2.3: La PPU de la Game Boy Advance.

memoria³, la ROM del cartucho (0x08000000). Para que los jugadores puedan guardar sus partidas se dedican los últimas 64 KB, sección denominada SRAM [3, 2, 12].

Todas las zonas de memoria anteriores se pueden ver distribuidas en el diagrama del circuito de la consola, en la Figura 2.4.

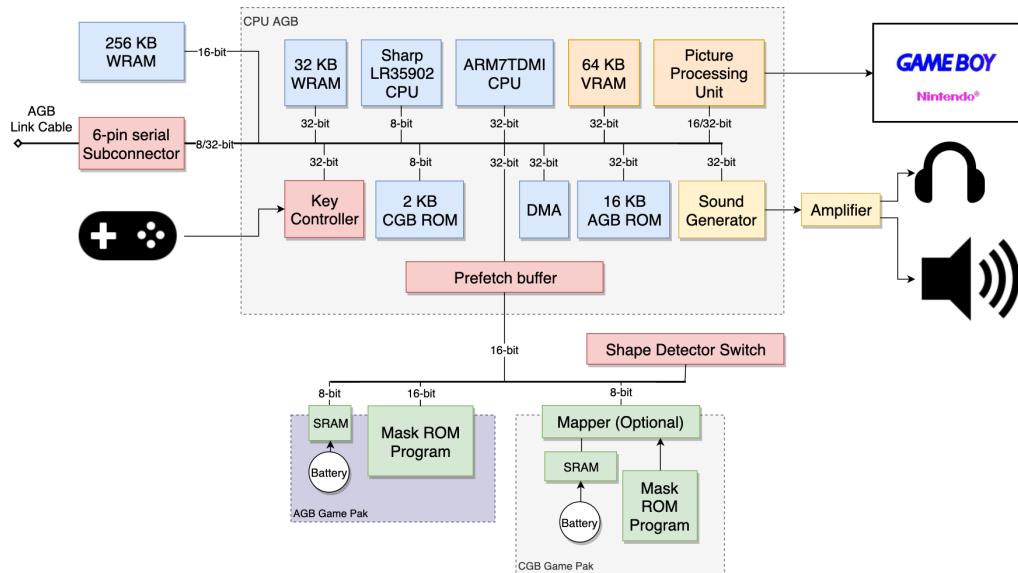


Figura 2.4: Diagrama simplificado del circuito de la Game Boy Advance [1].

2.1 PROCESADOR ARM7TDMI

En este punto se incluyen los aspectos más importantes del procesador principal de la Game Boy Advance. Toda la información se puede encontrar en [14] (el documento de referencia del procesador), complementada con detalles adicionales sobre el juego de instrucciones utilizado en [15].

Para cumplir con las exigencias de potencia computacional de la nueva generación de consolas portátiles, Nintendo, decidió diseñar un procesador basado en ARMv4, un juego de instrucciones (ISA o *Instruction Set Architecture* en inglés) RISC. Las especificaciones del ISA nos dan una idea de las características del dispositivo:

³Salvo en juegos multijugador, en ese caso la ejecución del juego comienza en las instrucciones localizadas en la RAM externa (0x02000000).

- **ARMv4** tiene un total de 37 registros de 32 bits, de los cuales 31 son generales y 6 son de estado. En la Game Boy existen diferentes modos. Cada uno de ellos tiene acceso a 17, o menos, de los registros disponibles. El acceso según el modo del procesador a los registros del sistema se describe en la Figura 2.5. Es importante resaltar que, tal y como pasa con otros procesadores ARM, algunos de los 37 registros no son *banked registers*, lo que significa que varios modos compartirán registros y, por lo tanto, los valores de estos pueden ser sobrescritos.

Ignorando los modos privilegiados mostrados en la Figura 2.5 (*FIQ, Supervisor, Abort, IRQ, Undefined*), los usos de los registros en el modo general (primera columna de la figura, *System and User*) que controlará el programador son:

- Generales: Los registros considerados como generales son R0-R12. Sin embargo, mientras que el modo ARM tiene libertad total para acceder a cada uno de los 13, el modo THUMB solo puede acceder de R8 a R12 utilizando instrucciones específicas.
- *Stack Pointer*: El registro R13 se utiliza para guardar la dirección de la pila, esto se realiza por convención en el modo ARM, dado que también se puede utilizar como un registro de uso general. No obstante, en el caso del modo THUMB debe utilizarse como *Stack Pointer*.
- *Link Register y Program Counter*: R14 se reserva para guardar la dirección de memoria de R15 (*Program Counter*) al realizar una instrucción de salto, también conocida como instrucción de *branch*. No obstante, como ocurre con el registro R13, en caso de no usar este tipo de instrucciones, el modo ARM permite utilizar el registro para uso general. Otra diferencia a tener en cuenta entre los dos modos es el valor del contador del programa, el registro R15, que almacena el valor del contador en los bits 2-31 en el modo ARM y 1-31 en el modo THUMB.
- CPSR y SPSR: El registro CPSR (*Current Program Status Register*), accesible en todos los modos, almacena datos como el estado del procesador e información de control como el resultado de instrucciones CMP. En los modos privilegiados encontramos un registro adicional además de los 17 registros explicados en el punto anterior, denominado SPSR (*Saved Program Status Register*), en el que se guardan los valores del registro CPSR en caso de excepción.

Las limitaciones que se comentaron anteriormente con el modo THUMB del procesador y los efectos que tiene en el acceso de los registros quedan representados en la Figura 2.6.

- La consola utiliza un juego de instrucciones **load-store** en el cual todas las operaciones aritméticas que operan en memoria tienen que realizarse mediante instrucciones load. Este ISA permite también el uso de constantes de 8, 12 y 24 bits para minimizar el uso de instrucciones.

Una vez que se han comentado las características del conjunto de instrucciones, se pasa a describir las características específicas del ARM7TDMI. Entre las más destacables se encuentran:

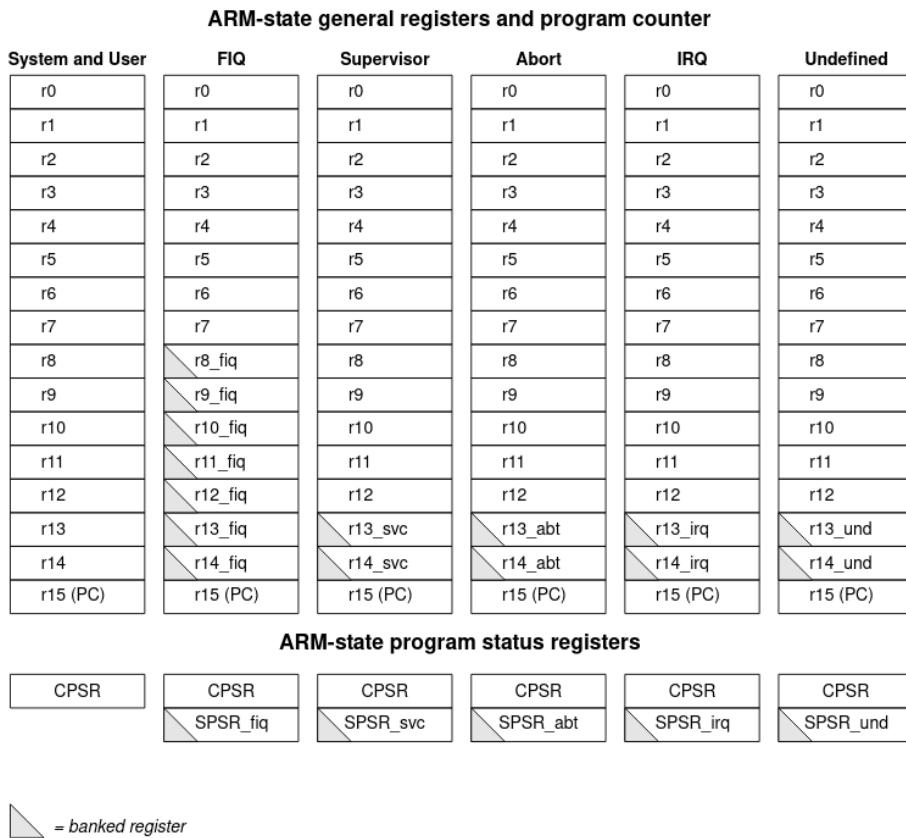


Figura 2.5: Registros por modo del procesador ARM7TDMI.

1. Un camino de datos segmentado de tres etapas: *Fetch*, *Decode* y *Execute* (véase la Figura 2.7). En concreto, esta implementación de la segmentación para el ARM7TDMI permite ejecutar hasta tres instrucciones de forma concurrente.
2. Una ALU de 32 bits, mejorando de forma sustancial las capacidades de modelos anteriores. De esta forma, la consola podía operar con números de 32 bits sin tener que consumir ciclos adicionales del procesador.
3. Un procesador con extensiones específicas, las cuales aparecen en el propio nombre, **ARM7TDMI**. A continuación se detallan:
 - **Thumb:** Tal y como se indicó previamente, el procesador soporta un modo que opera con instrucciones de 16 bits. Al ejecutarse en el hardware, las instrucciones se descomprimen a instrucciones de 32 bits sin pérdida de rendimiento. Según las estimaciones de ARM, programar en el modo THUMB reduce el tamaño del programa un 35% (comparado con el modo ARM) y proporciona un rendimiento del 160% al utilizarse en sistemas que trabajan en 16 bits [14]. Todas las mejoras mencionadas anteriormente son posibles siempre y cuando las limitaciones que implica el modo (el limitado acceso a los registros superiores, R8-R12, por ejemplo) no supongan un inconveniente al programar.
 - Extensiones de **Depuración mediante JTAG**: *Joint Test Access Group* o *Standard Test Access Port and Boundary-Scan Architecture*, siendo este último el nombre

2.1. PROCESADOR ARM7TDMI

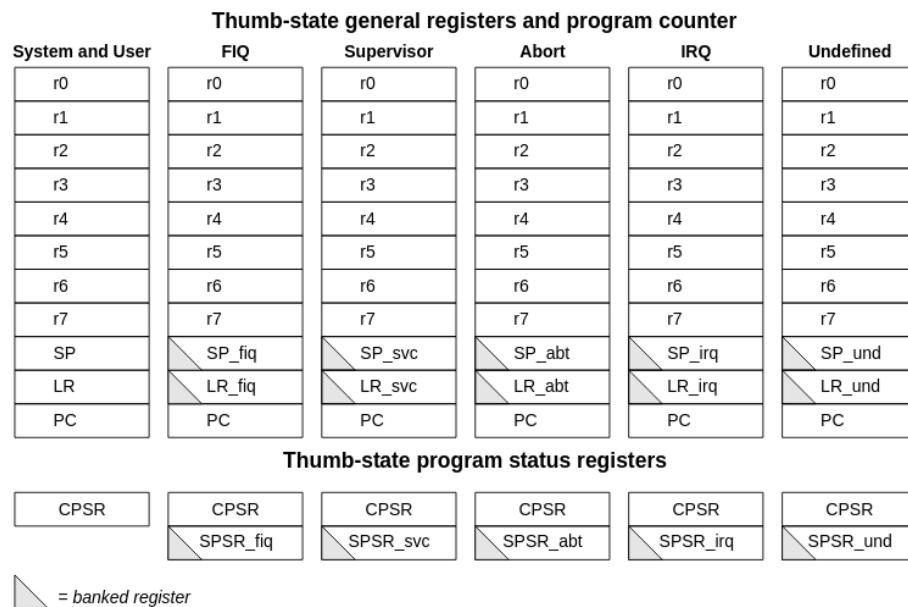


Figura 2.6: Registros por modo del procesador ARM7TDMI cuando este utiliza el set de instrucciones THUMB.

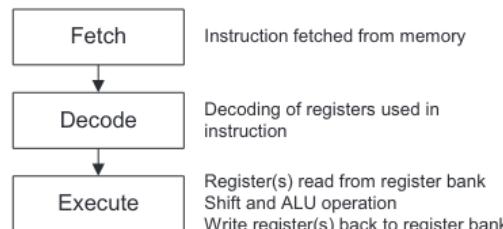


Figura 2.7: Segmentación de instrucciones de la Game Boy Advance.

oficial dado por la IEEE. Se trata de una especificación que permite a partir de una conexión, depurar un microcontrolador directamente en el hardware.

- **Multiplicador mejorado:** Los ciclos necesarios para completar multiplicaciones con números de 32 bits se reducen de forma considerable con respecto a las versiones anteriores del procesador.
- **EmbeddedICE macrocell:** Con ICE, *In-circuit emulation* en inglés, la consola añade la posibilidad de manipular directamente el hardware del dispositivo para poder realizar una depuración a fondo del programa.

La distribución de la lógica necesaria para poder incluir las extensiones de depuración mediante JTAG y EmbeddedICE macrocell se puede observar en la Figura 2.8. Aquí se puede ver también la adición de un controlador llamado “TAP”. Este sirve de intermediario entre la interfaz serial JTAG y las *scan chains*, que habilitan la comunicación entre la interfaz ICE y el procesador de la consola.

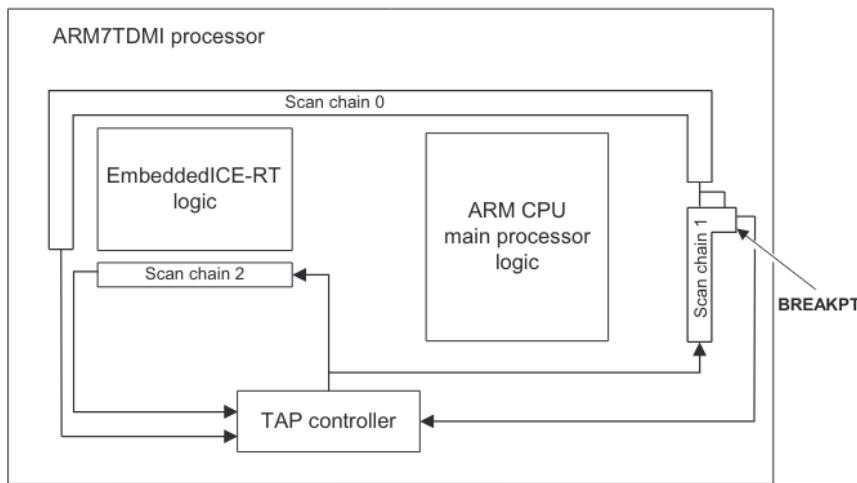


Figura 2.8: Distribución del EmbeddedICE-RT y controlador TAP en el diagrama de bloques del ARM7TDMI.

El lector interesado puede consultar [14] para obtener más información sobre el procesador ARM7TDMI.

2.2 COPROCESADOR SHARP SM83

Para mantener la retrocompatibilidad de modelos anteriores con la Game Boy Advance, Nintendo incluyó un coprocesador que se encargaría de llevar a cabo esta labor. Como era de esperar, la compañía incluyó el mismo procesador utilizado para la Game Boy original, un procesador Sharp SM83 dentro del SoC Sharp LR35902. El procesador, dependiendo del dispositivo, funcionaba a una frecuencia de 4.19 MHz u 8.38 MHz. Por aquel entonces, incluir el hardware de dispositivos anteriores era una práctica convencional, puesto que el hardware no era lo suficientemente potente como para emular generaciones pasadas. La situación hoy en día es diferente según el caso. Por ejemplo, la Xbox One ya incluye un emulador para poder ejecutar juegos de Xbox 360⁴.

La información disponible sobre el procesador Sharp es escasa. No obstante, se sabe que su diseño se basa, fundamentalmente, en los procesadores **Intel 8080** y **Zilog Z80**. A continuación se detallarán las características heredadas de cada uno de ellos.

La primera característica, que se hace evidente al saber en los dos procesadores en los que se basa, es que el **Sharp SM83 es un procesador de 8 bits**.

Sharp, al diseñar el procesador, combinó características de cada uno. Una de ellas sería la sintaxis utilizada al programar en ensamblador, en concreto, la sintaxis utilizada por el Z80. Esta se escogería seguramente por el *copyright* que tenía Intel en su día sobre la sintaxis utilizada en el 8080. Por lo tanto, las diferencias encontradas son menores. Los únicos cambios son el de los nombres de instrucciones abreviadas que utilizaba Intel, que se reemplazaron por otras que usaban el nombre completo (por ejemplo, “LOAD” en vez de “LD”).

⁴<https://www.extremetech.com/gaming/257851-microsoft-built-xbox-360-xbox-compatibility-xbox-one>

2.3. EL REGISTRO I/O

En cuanto a registros se refiere, se tomó prestado el diseño utilizado en el Intel 8080, pero con ligeras modificaciones que destacaremos a continuación. En concreto, los **10 registros** que utiliza el procesador Sharp son:

- El registro A de 8 bits, reservado para el acumulador.
- El registro F de 8 bits, reservado para guardar el estado del procesador.
- El resto de registros (el B, C, D, E, H, y L) de 8 bits, diseñados para un uso general por parte del programador.
- Los registros PC y SP, de 16 bits cada uno, se reservaban para su uso como contador de programa y puntero de pila, respectivamente.

Dado que algunas instrucciones lo permitían, se podían utilizar los registros en pares de la siguiente manera: AF, BC, DE y HL.

A pesar de tomar prestado el diseño de los registros, lo que **no heredó fue el esquema I/O del Intel 8080**, que permitía acceder a cada puerto mediante una instrucción específica. Se optó por el mismo sistema que utiliza la Game Boy Advance, “mapeando” cualquier puerto a una zona de memoria.

Otra especificación que copió del Intel 8080 fue el **bus de datos de 8 bits y direccionamiento de 16 bits**, con el cual se podían acceder a 64 KB de memoria.

2.3 EL REGISTRO I/O

La Game Boy Advance “mapea” los registros I/O de control directamente en memoria, utilizando parte del espacio de memoria disponible para el dispositivo. A pesar de disminuir el espacio de memoria disponible, Nintendo consigue de esta forma reducir costes y simplificar el diseño interno comparado con la alternativa, un diseño I/O independiente.

En este punto se incluyen las principales regiones utilizadas para configurar las funciones comentadas en el informe y requeridas en el código del proyecto (véase el Capítulo 3). Todos los registros, incluyendo los no mencionados en esta sección, se podrán consultar en el Apéndice A.3, en el que se describe brevemente cada uno junto con su nombre y tamaño. Toda la información del apéndice ha sido extraída de [12].

El mapa I/O de la consola está conformado por 7 secciones: Los **registros LCD, registros de sonido, canales DMA, registros de temporización, comunicación serie, botones del dispositivo y control de interrupciones** [12]. En los siguientes puntos se describirán cada una de ellas. Es importante mencionar que la extensión de cada uno de los puntos será proporcional a la importancia que tenga esa sección en particular para el proyecto. Toda la información mostrada en este punto, al tener una naturaleza puramente técnica con las especificaciones de Nintendo, tendrá como base [2] y [12].

2.3.1 Registros LCD

En los registros LCD se incluyen todos los parámetros utilizados para configurar la imagen que se muestra por pantalla.

2.3.1.1. DISPCNT

El primer registro que se encuentra en esta región es el denominado *Display Control* (o como se referencia en [12], DISPCNT), situado en la dirección de memoria 0x04000000. Entre los parámetros que se pueden modificar se encuentran:

- Bit {F}: El modo “ventana” reservado para sprites.
- Bits {E-D}: Los bits para activar los dos modos “ventana” reservados para los 4 fondos diferentes.
- Bit {C}: Activa el uso de sprites.
- Bits {B-8}: Cada uno de los bits en este rango activa los 4 fondos que puede tener la GBA en un momento dado. Los bits {B-8} activan los fondos 3, 2, 1 y 0 respectivamente.
- Bit {7}: Provoca un “reset” de la pantalla para que se muestre una imagen completamente blanca ignorando cualquier contenido almacenado en la VRAM y OAM.
- Bit {6}: Especifica la manera en la que están distribuidos los tiles en la OAM. Para 0, la distribución será de dos dimensiones mientras que para 1, la distribución será unidimensional. Véase la Figura 2.9 para una configuración 1D de los *tiles* en memoria y la Figura 2.10 para observar una configuración 2D de los *tiles*.



Figura 2.9: Distribución 1D de los *tiles* en la OAM.

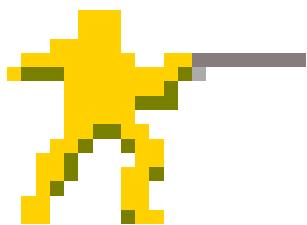


Figura 2.10: Distribución 2D de los *tiles* en la OAM.

- Bit {5}: Al activar este bit, el programador tendrá acceso a la OAM durante HBlank.
- Bit {4}: Permite la selección del buffer que se muestra por pantalla para los modos 4 y 5, los cuales manejan más de un buffer. Para más información consultar el punto 2.4.2.2 y 2.4.2.3.
- Bit {3}: La función que desempeña el tercer bit del registro DISPCNT es mostrar si el juego se está ejecutando en una Game Boy Color. El bit no se puede modificar.
- Bits {2-0}: Modo del fondo utilizado. Entre los valores permitidos están el 0, 1 y 2, definidos para las tres maneras que tiene la GBA de procesar fondos. Los modos 3, 4 y 5 están reservados para los fondos bitmap. Los valores 6 y 7 no están permitidos. Las diferencias entre los modos se verán en las secciones 2.4.2 y 2.4.3.

En la Tabla 2.2 se hace un resumen de los parámetros del registro DISPCNT.

2.3. EL REGISTRO I/O

Bits	Función
F	Modo “ventana” para <i>sprites</i> .
E-D	Modo “ventana” para fondos.
C	Activa el uso de <i>sprites</i> .
B-8	Activa cada uno de los 4 fondos disponibles.
7	“Reset” de la pantalla.
6	Especifica la distribución de los tiles en memoria.
5	Activa el acceso a la OAM durante HBlank.
4	Selecciona el buffer adicional en los modos 4 y 5.
3	Permite consultar si un juego se está ejecutando en una Game Boy Color.
2-0	Selecciona el modo que utiliza la Game Boy Advance para los fondos.

Tabla 2.2: Registro *Display Control*.

2.3.1.2. DISPSTAT

El siguiente registro, el *Display Status* (conocido también como el registro DISPSTAT), se sitúa en la dirección de memoria 0x04000004 y permite al programador consultar la información relacionada con el VBlank, HBlank y VCount. VBlank es el período que hay entre cada fotograma refrescado, y HBlank se refiere al período que hay entre cada línea refrescada por pantalla. Por otro lado, VCount es el número de líneas, las filas de píxeles horizontales, refrescadas. Los bits que contiene son los siguientes:

- Bits {F-8}: El programador especifica un valor VCount determinado, para invocar una interrupción en el bit 5 o que el cambio se vea reflejado en el bit 2.
- Bit {5}: Activa la interrupción de VCount si este coincide con el valor especificado en F-8.
- Bits {4-3}: Activa la interrupción de HBlank y VBlank respectivamente.
- Bit {2}: Este bit tendrá valor si VCount coincide con el valor especificado en F-8.
- Bits {1-0}: Permiten saber si el dispositivo se encuentra en HBlank, si el valor del bit 0 es 1, o en VBlank, si el valor del bit 1 es 1.

En la Tabla 2.3 se hace un resumen de los parámetros del registro DISPSTAT.

Bits	Función
F-8	VCount objetivo.
5	Activa una interrupción al alcanzar VCount objetivo.
4-3	Activa la interrupción de HBlank y VBlank.
2	Bit activo en caso de alcanzar VCount objetivo.
1-0	Estado de HBlank y VBlank.

Tabla 2.3: Registro *Display Status*.

El valor de VCount también se puede consultar en la dirección de memoria 0x04000006. Dado que el valor oscila entre 0 y 227, solo se tienen en cuenta los primeros 8 bits.

2.3.1.3. Configuración de fondos

A continuación, se detallan los registros para controlar los distintos parámetros que pueden tener los fondos de la Game Boy Advance. Estos registros sirven, entre otras cosas, para cambiar el tamaño, color y posicionamiento de los fondos.

- Bits {F-E}: Tamaño del fondo. Para más información consultar la sección 2.4.3.
- Bit D: Por defecto, los fondos que no utilizan la matriz de transformación, se repiten en caso de que el *offset* provoque que el fondo llegue a su fin. Este campo permite que los fondos que utilicen la matriz de transformación también se vean repetidos. Para más información, consultar el punto 2.4.3.
- Bits {C-8}: El índice del grupo de *tiles* específico que se utiliza.
- Bit 7: Especifica el formato del color a utilizar. Con un 0 se consiguen 16 colores disponibles (4 bpp) y con un 1 se consiguen 256 colores (8 bpp).
- Bit 6: Activa el modo mosaico. Más información en el punto 2.4.1.
- Bits {3-2}: Especifica el índice del bloque de grupo de *tiles* a utilizar.
- Bits {1-0}: Prioridad de renderizado del fondo. Cuanto mayor sea el valor antes se renderizará en pantalla. En caso de que haya fondos con la misma prioridad se recurre al orden natural, siendo el primer fondo (BG0) el que mayor prioridad tiene y el último fondo el que menor prioridad tiene (BG3).

En la Tabla 2.4 se hace un resumen de los parámetros del registro de control de fondos.

Bits	Función
D	Activar la función “wrap” en las matrices que utilicen la matriz de transformación.
C-8	El índice del grupo de <i>tiles</i> específico que se utilizará.
7	Rango de colores utilizado.
6	Modo mosaico.
3-2	Especifica el índice del bloque de grupo de <i>tiles</i> que se utilizará.
1-0	Prioridad de renderizado.

Tabla 2.4: Registro de control de fondos.

Cada uno de los fondos cuenta con un registro de control específico. La ubicación de los registros para cada fondo se puede observar en la Tabla 2.5. Además, en la tabla se incluye el *offset X e Y* mencionado anteriormente.

En caso de que el programador quiera renderizar los fondos a través de una matriz de transformación dada (más información en el punto 2.4), tiene que configurar los registros de transformación y los registros de referencia. Este último sustituye a los parámetros de *offset* vistos anteriormente para los fondos “normales” (aquellos que no utilicen la matriz de transformación).

2.3. EL REGISTRO I/O

Fondos	Registro de Control	Registro de Offset X	Registro de Offset Y
0	0x04000008	0x04000010	0x04000012
1	0x0400000A	0x04000014	0x04000016
2	0x0400000C	0x04000018	0x0400001A
3	0x0400000E	0x0400001C	0x0400001E

Tabla 2.5: Registros de cada uno de los fondos (1).

El registro de transformación, como su nombre indica, se limita a almacenar los valores de la matriz de transformación. Los valores de la matriz 2x2 se almacenan consecutivamente, ocupando 2 bytes cada uno, empezando por 0x04000020 para el fondo 2 y 0x04000030 para el fondo 3 (véase la Tabla 2.6). Estos valores son números *fixed point* desplazados por 8 bits.

Fondos	Registro de Transformación	Registro de Referencia
0	-	-
1	-	-
2	0x04000028-0x0400002E	0x04000020-0x04000026
3	0x04000038-0x0400003E	0x04000030-0x04000036

Tabla 2.6: Registros de cada uno de los fondos (2).

El otro registro a tener en cuenta, es el registro de referencia. Este replica el comportamiento del registro de *offset* pero con números *fixed point* de 28 bits. La distribución de los valores es la mostrada en la Tabla 2.7.

27	26–8	7–0
Signo	Valor entero	Valor decimal

Tabla 2.7: Formato del valor de referencia de los fondos utilizando la matriz de transformación.

Los puntos de referencia X e Y para el fondo 2 se almacenan en las direcciones de memoria 0x04000028 y 0x0400002E respectivamente, mientras que los puntos de referencia X e Y para el fondo 3 se localizan en las direcciones de memoria 0x04000038 y 0x0400003E respectivamente (véase la Tabla 2.6).

Como se puede apreciar en la Tabla 2.6, los fondos 0 y 1 no cuentan con la posibilidad de poder utilizar la matriz de transformación.

2.3.1.4. Registros de efectos especiales

Finalmente, en la sección de registros LCD, las tres secciones que quedan son las del modo ventana, modo mosaico y otros efectos especiales. Las demostraciones de estas tres funciones se pueden ver en la Sección 2.4.4.

Dado que se pueden crear dos ventanas diferentes, existen dos registros distintos para cada uno, donde se especifican las dimensiones horizontales y verticales del rectángulo creado. Las dimensiones horizontales se indican en las direcciones 0x04000040 y 0x04000042 para las ventanas 0 y 1 respectivamente. De los 2 bytes que conforman cada valor, se utiliza 1 byte para cada una de las dos coordenadas X que representan las esquinas inferior izquierda y superior derecha del rectángulo. El formato para las coordenadas verticales es

exactamente el mismo pero en las direcciones 0x04000044 para la ventana 0 y 0x04000046 para la ventana 1. En caso de que los valores superen la resolución del dispositivo (240x160), el valor introducido se interpreta como el máximo permitido. En el caso de la anchura, 240, y en el caso de la altura, 160.

Sin embargo, antes de poder configurar las posiciones de las ventanas es necesario especificar cómo funcionan. Para ello se tiene que definir el comportamiento dentro de las ventanas, en el registro WININ en la dirección de memoria 0x04000048, y el comportamiento fuera de estas, en el registro WINOUT en la dirección de memoria 0x0400004A. Los parámetros para WININ son:

- Bit {D}: Activa efectos especiales como el *blending* y los efectos *fade in* y *fade out* para la ventana 1.
- Bit {C}: Activa el renderizado de *sprites* para la ventana 1.
- Bits {B-8}: Activa los fondos 3, 2, 1 y 0, respectivamente, para la ventana 1.
- Bit {5}: Activa efectos especiales como el *blending* y los efectos *fade in* y *fade out* para la ventana 0.
- Bit {4}: Activa el renderizado de *sprites* para la ventana 0.
- Bits {3-0}: Activa los fondos 3, 2, 1 y 0, respectivamente, para la ventana 0.

Por otro lado, para controlar lo que se renderizó fuera de la ventana se configuran los siguientes parámetros en WINOUT:

- Bit {D}: Activa efectos especiales fuera de las ventanas de *sprites*.
- Bit {C}: Activa el renderizado de *sprites* fuera de una ventana de *sprites*.
- Bits {B-8}: Activa el renderizado de los fondos 3, 2, 1 y 0, respectivamente, fuera de una ventana de *sprites*.
- Bit {5}: Activa efectos especiales fuera de las ventanas de fondo.
- Bit {4}: Activa el renderizado de *sprites* fuera de una ventana de fondo.
- Bits {3-0}: Activa el renderizado de los fondos 3, 2, 1 y 0, respectivamente, fuera de las ventanas de fondo.

El registro para la función mosaico se encuentra en la dirección de memoria 0x0400004C. El efecto distorsiona (o “pixela”) la imagen dependiendo del valor que especifique el programador. Para una demostración del efecto consultar la Sección 2.4.4. Los bits que contiene son los siguientes:

- Bits {F-C}: Distorsión en el eje vertical del *sprite*.
 - Bits {B-8}: Distorsión en el eje horizontal del *sprite*.
-

2.3. EL REGISTRO I/O

- Bits {7-4}: Distorsión en el eje vertical del fondo.
- Bits {3-0}: Distorsión en el eje horizontal del fondo.

La última región de los registros LCD son aquellos dedicados al renderizado de efectos especiales. En concreto, los que permiten la mezcla de colores (o transparencia) y cambio de intensidad en la representación de los colores⁵.

El principal registro que gestiona toda la configuración relacionada con los efectos especiales es el registro posicionado en 0x04000050, denominado en [12] como BLDCNT. Los bits que contiene son los siguientes:

- Bits {D-8}: Selección de las capas posteriores a mezclar. Cada bit activa el “backdrop”⁶, *sprites* y fondos 3, 2, 1 y 0, respectivamente.
- Bits {7-6}: Selección del efecto a procesar. Siendo 3, se traduce en un incremento en la intensidad de la capa frontal. Si es 2, se hace una disminución en la intensidad de la capa frontal. Cuando es 1, se hace una mezcla de colores de las capas frontales con las posteriores seleccionadas. Finalmente, el 0 significa que no se escoge ningún efecto. Para que la mezcla de colores funcione como se espera es necesario que la capa frontal se encuentre, como su nombre indica, por encima de la capa (o capas) posteriores.
- Bits {5-0}: Selección de las capas frontales a mezclar. Cada bit activa el “backdrop”, *sprites* y fondos 3, 2, 1 y 0, respectivamente.

Para poder procesar los efectos tal y como desea el programador, este tiene que especificar los valores adecuados en los registros BLDALPHA para configurar el nivel de mezcla entre las capas frontales y posteriores, en 0x0400052. De forma adicional, para controlar el nivel de intensidad en las capas seleccionadas, tendrá que modificar el valor de BLDY en 0x04000054.

En el caso del registro BLDALPHA, que ocupa 2 bytes, los bits {C-8} denotan el coeficiente que tendrán las capas posteriores mientras que los bits {4-0} indican el coeficiente que tendrán las capas frontales. De forma similar a lo que ocurre con los registros de los fondos que utilizan las matrices de transformación, los 4 bits inferiores de cada coeficiente simbolizan incrementos fraccionales de 1/16, teniendo cada coeficiente un rango entre 0 y 1. En el caso del registro BLDY, ocurre lo mismo pero solo utilizando los bits {4-0} para especificar un valor entre 0 y 1.

2.3.2 Registros de sonido

La Game Boy Advance ofrece 6 modos diferentes de sonido, 4 analógicos y 2 digitales (canales A-B). Hay un registro para controlar cada uno de ellos además de varios generales que controlan el volumen.

⁵Un incremento en la intensidad se traduce en una imagen más “blanca” mientras que una imagen con menos intensidad se traduce en una imagen más oscura.

⁶El “backdrop” de la GBA consiste en un fondo completamente negro.

El término técnico para referirse a las 6 opciones de la consola para ofrecer audio es “canal”. Sin embargo, para no confundirlo con los canales de izquierda y derecha, a lo largo del trabajo, se hará referencia a los 6 canales como “modos”.

La información contenida en esta sección se basa en la documentación oficial [2] y en el material adicional disponible en [16].

2.3.2.1. Modos analógicos

Los dos primeros modos que ofrece el hardware de la consola, los canales 1 y 2, permiten la reproducción de pulsos con la posibilidad de modificar el volumen de cada onda. De forma adicional, el primer modo permite cambiar de dinámicamente la frecuencia de las ondas. Para ello, ambos modos harán uso de registros para modificar el tono y la duración de la onda, teniendo el modo 1 un registro adicional para controlar la frecuencia.

El registro encargado de la duración, volumen y ciclo de la onda utiliza el siguiente formato para los modos 1 y 2:

- Bits {F-C}: Volumen inicial del sonido.
- Bit {B}: Modificación del volumen. 0 para disminuirlo conforme pasa el tiempo y 1 para incrementarlo conforme pasa el tiempo.
- Bits {A-8}: El período con el que cambia el volumen. El valor especificado resulta en $n/64$ segundos de período.
- Bits {7-6}: El ciclo que toma el pulso. 0 para un ciclo del 12.5 %, 1 para un ciclo del 25 %, 2 para un ciclo del 50 % y 3 para un ciclo del 75 %.
- Bits {5-0}: La duración del sonido. El valor especificado resulta en $(64 - n)/256$ segundos.

El registro encargado de la frecuencia de la onda utiliza el siguiente formato para los modos 1 y 2:

- Bit {F}: En caso de estar activo el bit, “resetea” el sonido.
- Bit {E}: En caso de ser 0, el sonido es reproducido de forma constante. Si es 1, el sonido es reproducido con la duración especificada en los bits {5-0} del registro anterior.
- Bits {A-0}: Frecuencia utilizada para el tono. El valor especificado resulta en $131072/(2048 - n)$ Hz.

Para indicar un cambio dinámico en la frecuencia, el modo 1 cuenta con el registro “sweep”, que tiene los siguientes bits:

- Bits {6-4}: El tiempo que pasa entre cambios de frecuencia. El valor especificado resulta en $n/128$ segundos.

2.3. EL REGISTRO I/O

- Bit {3}: Cómo se modifica la frecuencia. 0 para el incremento con el paso del tiempo y 1 para la disminución.
- Bits {2-0}: Denota el cambio en la frecuencia. El valor especificado resulta en un cambio en el periodo de $T = T \pm T/2^n$. La suma o resta dependerá del valor del bit 3.

En la Tabla 2.8 se incluyen los registros de los modos 1 y 2.

Canal	Registro de duración	Registro de frecuencia	Registro “sweep”
1	0x04000062	0x04000064	0x04000060
2	0x04000068	0x0400006C	-

Tabla 2.8: Registros dedicados de los modos 1 y 2.

El modo 3, permite reproducir ondas almacenadas en el espacio de memoria localizado entre 0x04000090 y 0x0400009F. En este modo se pueden diferenciar los registros de control para la selección, duración y frecuencia.

El primer registro, reservado para seleccionar la muestra localizada en [0x04000090, 0x0400009F] y utilizado para activar el modo, tiene el siguiente formato:

- Bit {7}: Activa el modo.
- Bit {6}: Selecciona el grupo a utilizar. Complementa al bit 5.
- Bit {5}: En caso de que el valor sea 0, la Game Boy Advance dividirá las ondas almacenadas en el rango de memoria especificado anteriormente en dos grupos diferentes de 32 muestras cada una. En caso de ser 1, gestionará toda la información almacenada en el rango como un solo grupo de 64 muestras.

El segundo registro, es el encargado de modificar la duración y el volumen del modo 3:

- Bits {F-D}: Volumen del modo. 0 para 0 %, 1 para 100 %, 4 para 75 %, 2 para 50 % y 3 para el 25 %.
- Bits {7-0}: La duración del sonido. El valor especificado resulta en $n/256$ segundos.

Finalmente, el tercer registro necesario para utilizar el modo 3 es el encargado de configurar la frecuencia y repetición del sonido. Tiene el mismo formato descrito para los modos 1 y 2, y cuenta con los siguientes bits:

- Bit {F}: En caso de estar activo el bit, “resetea” el sonido.
- Bit {E}: Si es 0, reproduce el sonido de forma constante. Si es 1, reproduce el sonido con la duración especificada en los bits {5-0} del registro anterior.
- Bits {A-0}: Frecuencia utilizada para el tono. El valor especificado resulta en $131072/(2048-n)$ Hz.

El último modo analógico, el modo 4, tiene como función generar ruido como añadido a los demás modos. De los dos registros reservados para el modo, el que controla la duración y el volumen sigue el mismo formato presentado anteriormente. El registro diferente que se introduce en este caso, es el registro de ruido, que tiene la siguiente estructura:

- Bit {F}: En caso de estar activo el bit, “resetea” el sonido.
- Bit {E}: Si es 0, reproduce el sonido de forma constante. Si es 1, reproduce el sonido con la duración especificada en los bits {5-0} del registro anterior.
- Bits {7-4}: Denota la función a utilizar con la frecuencia especificada en los bits {2-0}. La salida de la función (*o pre-scaler divider* como se llama en [16]) será utilizada para generar el ruido. El efecto del valor del campo sobre la función se puede observar en la ecuación 2.1.

$$Q/2^{\text{valor}+1} \quad (2.1)$$

- Bit {3}: Controla el período del ruido generado. Si es 0, el periodo es de 127 ciclos y si es de 1, el periodo es de 32767 ciclos.
- Bits {2-0}: Frecuencia utilizada en conjunto con la función denotada en [7-4] para generar ruido. Las frecuencias vinculadas a los 8 valores disponibles se pueden observar en la Tabla 2.9.

Valor	Frecuencia (en MHz)
0	4.19304/2
1	4.19304/8
2	4.19304/16
3	4.19304/24
4	4.19304/32
5	4.19304/40
6	4.19304/48
7	4.19304/56

Tabla 2.9: Frecuencias disponibles para generar ruido en el modo 4.

2.3.2.2. Modos digitales

Se introdujeron dos nuevos modos con respecto al sistema de audio de la Game Boy Color. Estos modos, denominados A y B, hacen uso de los dos DACs (Conversor Digital Analógico) de 8 bits incorporados en la consola. Para utilizarlos se distinguen dos tipos de registros, aquellos reservados para indicar los datos a usar y otros para configurar los modos, los registros de control.

Los registros de datos de los modos A y B, se encuentran en las regiones 0x040000A0 y 0x040000A4, respectivamente. El sistema que implementa la Game Boy Advance, denominado Direct Sound, coge los primeros datos que encuentra en el buffer (FIFO), es decir, los 8

2.3. EL REGISTRO I/O

bits menos significativos del registro de 32 bits, que serán los primeros en reproducirse. Para activar la salida de sonido de estos modos, y también los modos 1-4, se utilizan tres registros de control. El primero está en 0x04000080 y tiene como función activar y controlar de forma global el volumen de los modos 1-4. Tiene el siguiente formato:

- Bits {F-C}: Cada uno de los 4 bits activa el canal izquierdo para los modos 1-4.
- Bits {B-8}: Cada uno de los 4 bits activa el canal derecho para los modos 1-4.
- Bits {6-4}: Volumen en el canal izquierdo de los primeros 4 modos.
- Bits {2-0}: Volumen en el canal derecho de los primeros 4 modos.

El segundo registro de control utilizado es el posicionado en la dirección de memoria 0x04000082. Este modifica el volumen global de los modos 1-4 para los dos canales, además de configurar parámetros relacionados con los modos A y B. Tiene el siguiente formato:

- Bit {F}: Restablece el buffer de datos FIFO del modo B.
- Bit {E}: Para poder reproducir sonido en los modos A y B, es necesario utilizar el temporizador 0 o 1. En este campo se especifica cual de los dos temporizadores se utilizará para el modo B.
- Bits {D-C}: Activa los canales izquierdo (2) y/o derecho (1) del modo B.
- Bit {B}: Restablece el buffer de datos FIFO del modo A.
- Bit {A}: Al igual que en el campo E se especifica el temporizador a utilizar para el modo A, número 0 o 1.
- Bits {9-8}: Activa los canales izquierdo (2) y/o derecho (1) del modo A.
- Bit {3}: Volumen global del modo B.
- Bit {2}: Volumen global del modo A.
- Bits {1-0}: Volumen global de los primeros 4 modos.

Finalmente, el registro denominado el “master switch” se posiciona en la dirección de memoria 0x04000084. El registro también puede servir para consultar el estado actual en el que se encuentran los modos 1-4. Es importante mencionar que los bits destinados a consultar el estado actual de un modo no pueden ser sobrescritos por el programador.

- Bit {7}: Permite desactivar todos los modos que ofrece la Game Boy Advance al igualar el campo a 0.
 - Bits {3-0}: Especifica el estado de los modos 4, 3, 2 y 1, respectivamente. Un valor igual a 1 denota que el estado está activo, mientras que el 0 indica lo contrario.
-

2.3.3 Registros para *Direct Memory Access*

La GBA ofrece la posibilidad de copiar datos rápidamente de una dirección de memoria a otra mediante el uso de *Direct Memory Access* (DMA) [17].

La consola cuenta con 4 variantes de DMA con diferentes prioridades y, por ende, finalidades. El canal número 0 permite realizar transferencias con la máxima prioridad disponible. Los canales 1 y 2 se reservan para las transferencias de sonido a los buffers adecuados. Por último, el canal 3 se reserva para transferencias generales.

Los registros de 32 bits que permiten llevar a cabo estas operaciones son los registros fuente, destino y de control DMA. Estos tres registros se repiten para cada uno de los 4 canales. En concreto, el registro de control DMA sigue el formato descrito a continuación:

- Bit {1F}: Activa una transferencia mediante DMA.
- Bit {1E}: Activa una interrupción en caso de haberse completado la operación.
- Bits {1D-1C}: Especifica cuándo debe comenzar la transferencia: 0 de manera inmediata, 1 al entrar en el estado VBlank, 2 al entrar en el estado HBlank y el comportamiento al igualar a 3 dependerá del canal utilizado. Para los canales reservados para audio se empieza la transferencia una vez que se hayan leído los buffers correspondientes. Para el canal 3 se realiza una transferencia antes de que la PPU renderice cada línea horizontal de la pantalla [3].
- Bit {1A}: Especifica el tamaño del dato a copiar. 0 para 16 bits y 1 para 32 bits.
- Bit {19}: Si se ha seleccionado el modo 1 en los bits {1D-1C}, se repite la transferencia cada vez que el dispositivo entre en el estado VBlank. En el caso de haber seleccionado el modo 2 en los bits {1D-1C}, se repite la transferencia cada vez que el dispositivo entre en el estado HBlank.
- Bits {18-17}: Definen la modificación de la dirección fuente que se lleva a cabo al completar una copia: 0 para incrementar la dirección fuente, 1 para decrementar la dirección fuente, 2 para dejar la dirección de destino fija y 3 para incrementar la dirección fuente, restableciendo el valor inicial una vez acabada la operación.
- Bits {16-15}: La finalidad de estos dos bits es similar a la mostrada en los bits {18-17} pero para la dirección destino.
- Bits {0F-00}: El número de transferencias realizadas en la operación.

Los registros de fuente y destino restantes se configuran simplemente igualando el valor del registro a la dirección de memoria correspondiente. La dirección de memoria para canal quedaría tal y como se muestra en la Tabla 2.10.

Canal	Registro fuente	Registro destino	Registro de control
0	0x040000B0	0x040000B4	0x040000B8
1	0x040000BC	0x040000C0	0x040000C4
2	0x040000C8	0x040000CC	0x040000D0
3	0x040000D4	0x040000D8	0x040000DC

Tabla 2.10: Registros fuente, destino y de control de DMA.

Las transferencias que utilizan *DMA*, a pesar de ser más rápidas que las funciones *memcpy* y *memset* convencionales, traen consigo problemas adicionales que pueden causar un comportamiento inesperado. Por ejemplo, al poner a la CPU en un estado “inerte”, cualquier tipo de interrupción configurada se puede ver afectada [3].

Finalmente, es importante comentar que el canal 0 únicamente puede copiar desde la RAM interna del dispositivo.

2.3.4 Registros de temporización

La Game Boy Advance ofrece la posibilidad de configurar hasta 4 contadores independientes que permitan un seguimiento del tiempo. Los dos tipos de registros que se diferencian en esta sección, son los registros de control y de datos. Cada uno ocupa 2 bytes [3].

El registro de control tiene el siguiente formato:

- Bit {7}: Activa el temporizador.
- Bit {6}: Activa la interrupción cuando el temporizador sobrepasa el límite.
- Bit {2}: Hace que los “overflows” de temporizadores posicionados previamente en memoria incrementen el valor del temporizador. Por ejemplo, si el valor objetivo se alcanza en el temporizador 2, el temporizador 3 incrementará su valor.
- Bits {1-0}: Define la frecuencia con la que se actualiza el contador: 0 para una frecuencia de 16.78 MHz, 1 para 262.21 KHz, 2 para 65.536 KHz y 3 para 16.384 KHz. El programador puede conseguir de manera exacta la frecuencia con la que se actualizan los valores especificando un “offset” en el registro de datos [3].

El registro de datos complementa al registro de control y permite tanto la lectura del valor actual del contador como la escritura del valor inicial del contador. El valor que provocará el “overflow” viene dado por la ecuación 2.2. Dado que el programador puede especificar un valor inicial, no siempre se producirá un “overflow” a los 65536 incrementos.

$$65536 - n_{inc} = 0 \quad (2.2)$$

El formato de ambos registros se utiliza repetidamente para cubrir cada uno de los 4 temporizadores. Las direcciones de memoria de los registros de control y de datos para cada temporizador se pueden observar en la Tabla 2.11.

Temporizador	Registro de Control	Registro de Datos
0	0x04000102	0x04000100
1	0x04000106	0x04000104
2	0x0400010A	0x04000108
3	0x0400010E	0x0400010C

Tabla 2.11: Registros de control y datos de los temporizadores.

2.3.5 Registros de comunicación serie

La Game Boy Advance permite el intercambio de información con otras GBAs a partir del puerto serie que trae cada una de sus tres variantes. En concreto, la consola ofrece 6 modos diferentes en los que otro dispositivo puede interactuar con la consola. Sin embargo, en este apartado solo se comentarán brevemente el modo normal, que permite la comunicación bidireccional entre dos consolas y unidireccional entre varias GBA; y el modo multijugador, el cual permite la comunicación bidireccional entre un máximo de 4 consolas Game Boy Advance.

Tanto el modo normal como el multijugador precisa del registro de control general de comunicación (llamado RCNT según [12]) y el registro de control de entrada y salida serie (SIOCNT).

En los dos modos es necesario inicializar el registro RCNT (posicionado en la dirección de memoria 0x04000134) a los valores mostrados a continuación:

- Bit {F}: Su valor debe ser 0 para poder utilizar cualquiera de los dos modos de comunicación.
- Bit {E}: A pesar de no tener un uso (por lo menos documentado) su valor debe ser 0.
- Bits {8-4}: A pesar de no tener un uso (por lo menos documentado) su valor debe ser 0.
- Bits {3-0}: Es irrelevante para los modos descritos en esta sección.

El registro SIOCNT, en el que sí hay diferencias entre los dos modos, se ubica en la dirección de memoria 0x04000128. Para el modo normal, el registro sigue el siguiente formato:

- Bit {E}: Activa la interrupción de comunicación serie en caso de haberse completado una transferencia.
- Bit {D}: Se hace 0 para activar la comunicación en modo normal y 1 para activar la comunicación en modo multijugador.
- Bit {C}: El campo configura el tamaño de la transferencia, 0 para 8 bits y 1 para 32 bits.
- Bit {7}: Especifica el estado actual de la comunicación, 0 denota un estado inactivo y 1 un estado activo.
- Bit {3}: Estado de salida de datos.
- Bit {2}: Estado de entrada de datos. Se traduce en el estado del bit {3} de los demás dispositivos.

2.3. EL REGISTRO I/O

- Bit {1}: Define la frecuencia a la que funciona la comunicación. Es 0 para 256 KHz y 1 para 2 MHz. Para conseguir estabilidad en la transferencia de datos se recomienda una frecuencia de 256 KHz dado que la transferencia a 2 MHz se reserva para hardware especial [12].
- Bit {0}: Indica el reloj utilizado como referencia para la transferencia. Es 0 para el reloj externo y 1 para el interno.

El modo multijugador sigue un formato parecido excepto por los valores posicionados en el primer byte del registro. Concretamente, se sigue el siguiente formato:

- Bit {E}: Activa la interrupción de la comunicación serie en caso de haberse completado una transferencia.
- Bit {D}: Se hace 0 para activar la comunicación en modo normal y 1 para activar la comunicación en modo multijugador.
- Bit {C}: Para poder utilizar el modo multijugador el valor debe ser 0.
- Bit {7}: Especifica el estado actual de la comunicación. Se pone a 0 para indicar un estado inactivo y a 1 para un estado activo. En el caso del modo multijugador, este bit no se puede modificar en los dispositivos esclavos conectados a la GBA principal.
- Bit {6}: El valor es 1 en caso de haberse producido un error.
- Bits {5-4}: El valor define el rol que toma el dispositivo en la comunicación. Es 0 para el dispositivo principal, y 1, 2 y 3 para los dispositivos esclavos.
- Bit {3}: Estado de la conexión. El valor 0 denota un fallo en la conexión, mientras que el 1 indica que todos los demás dispositivos están listos.
- Bit {2}: Es similar a los bits {5-4}. Denota si el dispositivo tiene el rol maestro (1) o de esclavo (0).
- Bits {1-0}: Indica la tasa de baudios a la que se envían los datos. Es 0 para 9600, 1 para 38400, 2 para 57600 y 3 para 115200 bps.

Los siguientes registros a tener en cuenta son los encargados de enviar los datos a los diferentes dispositivos.

En el caso del modo normal se pueden distinguir dos registros, uno dedicado a las transferencias de 8 bits (en la dirección de memoria 0x0400012A) y otro para las transferencias de 32 bits (en la dirección de memoria 0x04000122). En el registro de 8 bits, únicamente se almacenarán los 8 bit menos significativos. Para garantizar el correcto funcionamiento de la comunicación, los dos registros deben contener la información a enviar antes de activar la comunicación serie. Al terminar, la información recibida acaba sustituyendo los datos iniciales escritos en ambos registros.

Por otro lado, el modo multijugador, cuenta con un registro específico para el envío de datos, que reutiliza el registro destinado a transferencias de 8 bits en el modo normal, pero

esta vez utilizando completamente los 16 bits del registro. Además, se incluyen registros para recibir los datos de cada uno de los roles en una comunicación serie multijugador. La distribución de estos se puede ver en la Tabla 2.12.

Dispositivo	Registro I/O para el modo multijugador
0 (Máster)	0x04000120
1 (Esclavo)	0x04000122
2 (Esclavo)	0x04000124
3 (Esclavo)	0x04000126

Tabla 2.12: Registros I/O para el modo multijugador.

2.3.6 Registros de los botones

El dispositivo ofrece dos formas de obtener la entrada del usuario:

- La primera es comprobar directamente el registro KEYINPUT, posicionado en la dirección 0x04000130. Este registro ocupa 2 bytes, aunque solo son útiles los primeros 10 bits. La distribución de los botones de la consola se puede observar en la Tabla 2.13. Al consultar los valores del registro es necesario tener en cuenta que un 0 en un bit determinado significa que el botón ha sido pulsado, mientras que un 1 indica lo contrario.

Bit	Botón
9	L
8	R
7	Abajo (D-pad)
6	Arriba (D-pad)
5	Izquierda (D-pad)
4	Derecha (D-pad)
3	Start
2	Select
1	B
0	A

Tabla 2.13: Los botones en el registro KEYINPUT.

- La segunda forma, únicamente recomendada para hacer que el dispositivo salga de un estado de bajo consumo, se basa en el uso de interrupciones. Para ello, el programador tendrá que configurar el registro KEYCNT (véase la Tabla 2.14), ubicado en la dirección de memoria 0x04000132.

Tabla 2.14: Los botones en el registro KEYCNT.

Bit	Función
9	Se tiene en cuenta el botón L para la interrupción.
8	Se tiene en cuenta el botón R para la interrupción.
7	Se tiene en cuenta el botón de abajo (D-pad) para la interrupción.
6	Se tiene en cuenta el botón de arriba (D-pad) para la interrupción.
5	Se tiene en cuenta el botón de izquierda (D-pad) para la interrupción.
4	Se tiene en cuenta el botón de derecha (D-pad) para la interrupción.
3	Se tiene en cuenta el botón Start para la interrupción.
2	Se tiene en cuenta el botón Select para la interrupción.
1	Se tiene en cuenta el botón B para la interrupción.
0	Se tiene en cuenta el botón A para la interrupción.

2.3.7 Registros de interrupciones

Para activar y utilizar las interrupciones en el dispositivo, el programador precisará de tres registros: IME, IE e IF.

El primero, el registro IME (*Interrupt Master Enable Register*), es el encargado de activar el uso de interrupciones en el sistema. De los 4 bytes que ocupa el registro, solo se utiliza el primer bit, para denotar si se utilizan interrupciones o no.

En caso de activar el uso de interrupciones, el programador tendrá que especificar cuáles hay que utilizar. Para ello tendrá que modificar los valores del registro IE (*Interrupt Enable Register*) para indicar cuáles de las 14 interrupciones disponibles serán utilizadas (véase la Tabla 2.15).

Tabla 2.15: Interrupciones disponibles en el sistema.

Bit	Función
D	Activa la interrupción al retirar el cartucho de la consola.
C	Activa la interrupción para la detección de los botones especificada en el registro KEYCNT.
Continúa en la siguiente página	

Tabla 2.15

Bit	Función
B	Activa la interrupción al completar la transferencia número 3 a través de DMA.
A	Activa la interrupción al completar la transferencia número 2 a través de DMA.
9	Activa la interrupción al completar la transferencia número 1 a través de DMA.
8	Activa la interrupción al completar la transferencia número 0 a través de DMA.
7	Activa la interrupción al completar una transferencia por comunicación serie.
6	Activa la interrupción cuando se sobrepasa el temporizador número 3.
5	Activa la interrupción cuando se sobrepasa el temporizador número 2.
4	Activa la interrupción cuando se sobrepasa el temporizador número 1.
3	Activa la interrupción cuando se sobrepasa el temporizador número 0.
2	Activa la interrupción cuando el valor de VCount objetivo especificado en el registro DISPSTAT es alcanzado.
1	Activa la interrupción cuando se entra en la región HBlank de la pantalla.
0	Activa la interrupción cuando se entra en la región VBlank de la pantalla.

Sin embargo, una vez provocada la interrupción, el programador tendrá que hacerle saber a la máquina que la interrupción ha sido gestionada exitosamente. El registro donde se realiza esta acción es el IF (*Interrupt Request Flags*), en la dirección de memoria 0x04000202. Siguiendo el mismo formato mostrado en la Tabla 2.15, se tendrá que escribir un 1 en el bit de la interrupción gestionada. Todo esto se hace desde la función que el programador tendrá que especificar en la dirección de memoria 0x03007FFC.

2.4 Picture Processing Unit (PPU)

La unidad de procesamiento de imágenes de la Game Boy Advance procesa la información de las secciones OAM, VRAM y RAM de paleta, mostradas en la Tabla 2.1, para producir una imagen que el jugador pueda ver. En la Sección 2 se detallaron los modos existentes según la PPU trata los datos. Diferenciamos tres: **Sprites**, **Bitmaps** y **Fondos** [2]. En las próximas secciones se comentarán cada uno de estos modos y sus diferentes variaciones y configuraciones.

2.4. PICTURE PROCESSING UNIT (PPU)

Sin embargo, antes de indagar en cada uno de los modos hace falta entender conceptos como los *tiles*. Estos son necesarios al hablar de los modos de *sprites* y *fondos*, la forma en la que el programador posiciona los píxeles y objetos en pantalla, y el formato utilizado para representar colores en la consola. Todos son conceptos indispensables para cualquiera de los modos que ofrece la consola de Nintendo.

Los *tiles* denotan unas pequeñas imágenes (normalmente cuadradas) que sirven como base para crear escenarios y personajes. Para facilitar la comprensión de este proceso, se muestra a continuación un ejemplo en *Tiled* (véase sección 1.5.4) donde se crea un escenario (o fondo) a partir de imágenes base de un tamaño de 32 x 32 píxeles. Al conjunto de imágenes base (*tiles*) se le denomina *tileset*. En la Figura 2.11 se muestra un *tileset* en el que se puede distinguir cada unidad por la separación que hace *Tiled*.

A partir de este conjunto de imágenes base, se crea la imagen final. El proceso se puede observar en las Figuras 2.12 y 2.13.

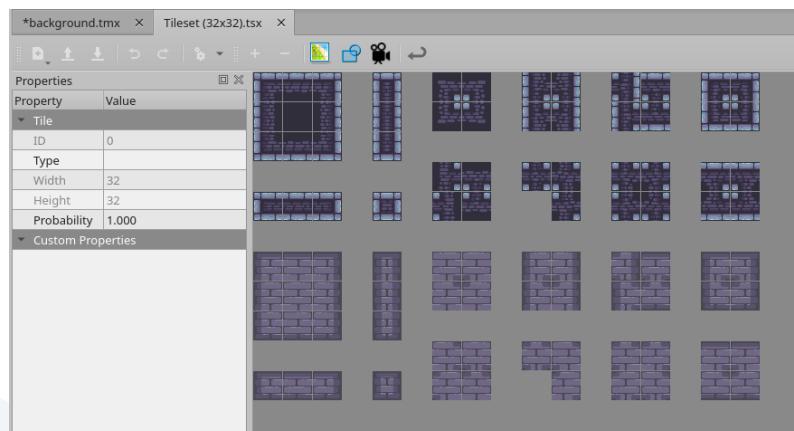


Figura 2.11: Tileset que se utilizará para crear el fondo.

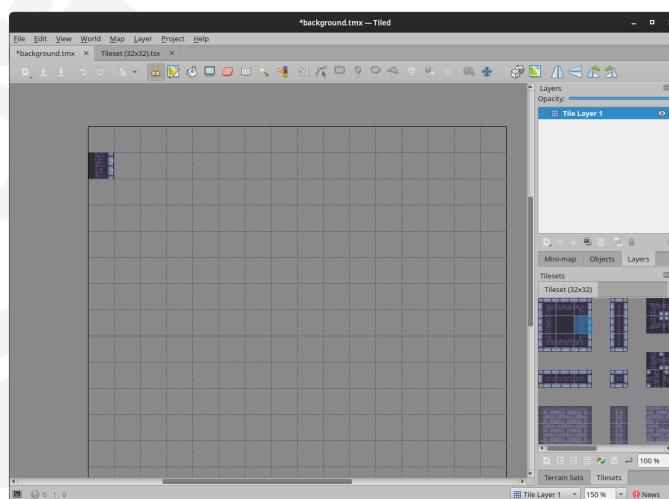


Figura 2.12: Relleno del canvas a partir de *tiles*.

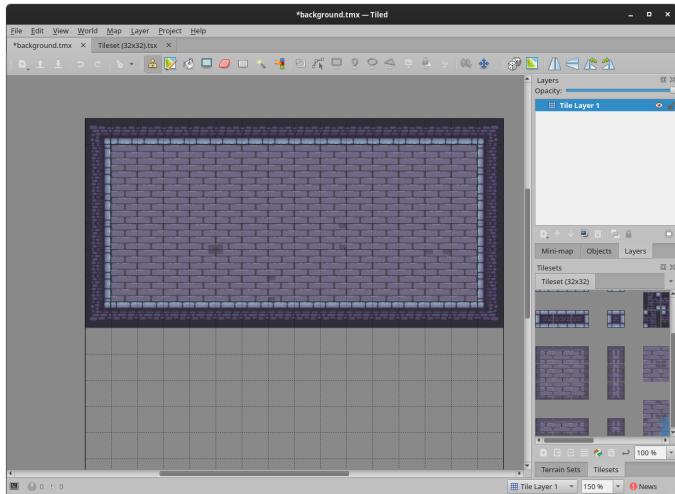


Figura 2.13: Creación de un fondo a partir del *tileset*.

Esta metodología facilita la reutilización de recursos. No obstante, utilizar esta técnica requiere crear una tabla de índices adicional (o mapa) para poder especificar qué *tile* se está utilizando [3].

La reutilización se hace evidente si se consulta de nuevo la Figura 2.13 y se intenta identificar el número total de *tiles* utilizados. En dicha Figura se utilizan 16 *tiles* diferentes. En el caso de la Game Boy Advance este número se reduciría a 9, ya que en la tabla o mapa se puede especificar la orientación (invertir el eje horizontal o vertical) del *tile*. La Figura 2.14 muestra el fondo junto con el índice de cada *tile*.

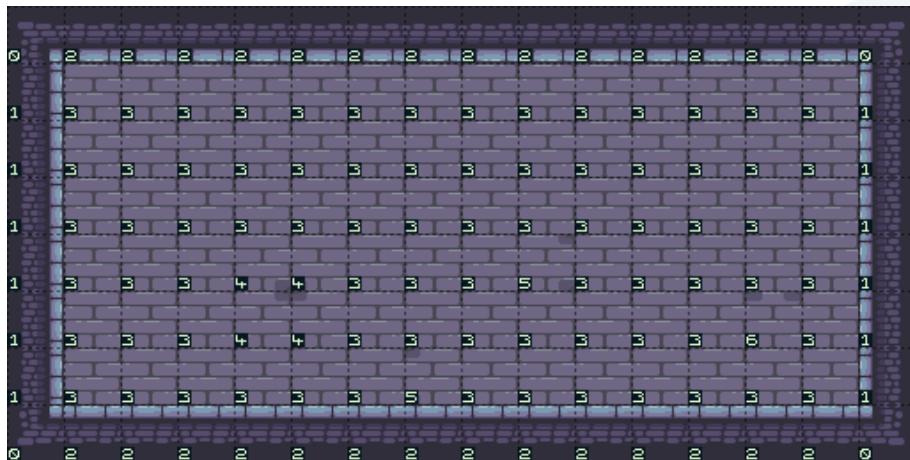


Figura 2.14: Distintos *tiles* utilizados.

En varias de las funciones que ofrece la *PPU* es necesario especificar el posicionamiento de los *tiles* a utilizar para poder renderizar la imagen correctamente. Es por esto que los *tiles* se distribuyen en 32 grupos de 2048 bytes cada uno (conocidos como “screenblocks” en [3]). Estos grupos, a su vez, se organizan en 4 bloques todavía más abstractos de 16 KB cada uno (conocidos como “charblocks” en [3]). La distribución de los grupos descritos anteriormente sobre la VRAM se puede observar en la Tabla 2.16.

2.4. PICTURE PROCESSING UNIT (PPU)

Memoria	0x06000000	0x06004000	0x06008000	0x0600C000
“charblock”	0	1	2	3
“screenblock”	{0-7}	{8-15}	{16-23}	{24-31}

Tabla 2.16: Distribución de los *tiles* en la VRAM [3].

Para poder programar para el dispositivo, es necesario conocer también cómo trabaja la Game Boy Advance al posicionar píxeles y objetos en pantalla, además de cómo y cuándo se refresca la imagen.

Como se puede deducir de la distribución utilizada para los *tiles*, la Game Boy Advance trata a la imagen como una matriz cuyas filas conforman el eje Y y las columnas el eje X. El punto de partida para los ejes, en el que su valor es 0, se posiciona en la esquina superior izquierda de la pantalla (véase la Figura 2.15). Esto será de utilidad no solo en los modos *bitmap* donde el programador manipulará la imagen a nivel de píxeles, sino también al trabajar con objetos y fondos, dado que será necesario indicar unas coordenadas para moverlos y posicionarlos.



Figura 2.15: Coordenadas de la pantalla.

Al contrario de lo que pueda parecer, la forma en la que la pantalla de la Game Boy Advance se refresca influye en la manera en la que se programa para el dispositivo. Cada fotograma nuevo es actualizado no de “golpe”, sino línea a línea (cada una de estas líneas recibe el nombre de *scanline*). Entre cada línea refreshada existe un período llamado HBlank, y entre cada imagen refreshada existe un período llamado VBlank. Ambos ofrecen al programador un rango de tiempo en el que modificar ciertos parámetros sin distorsionar la imagen⁷. En total, la GBA cuenta con 228 *scanlines* (el número de píxeles verticales más 68) de una longitud de 308 (el número de píxeles horizontales más 68). La Figura 2.16 muestra los conceptos comentados anteriormente.

⁷En el caso de HBlank, esto no es del todo cierto en todos los casos dado que si se actualiza un *sprite* durante HBlank es posible que se vea distorsionado.

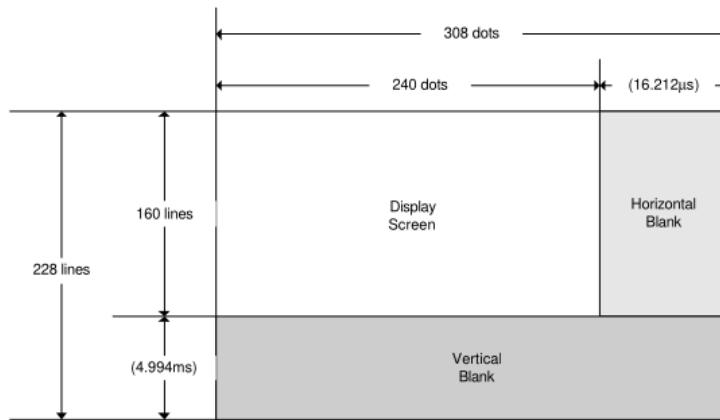


Figura 2.16: Los períodos HBlank y VBlank en la pantalla de la GBA. Imagen de [2].

Finalmente, queda mencionar que la consola utiliza un formato de 15 bits para representar colores. La distribución del rojo, verde y azul en este formato aparece representada en la Tabla 2.17. Como puede observarse, el bit más significativo no se usa [2].

ø	B	B	B	B	B	G	G	G	G	R	R	R	R	R
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Tabla 2.17: Color de 15 bits.

Partiendo de los conceptos previos, se comentará a continuación el uso y funcionamiento de los mismos aplicados a los *sprites*, fondos y *bitmaps* de la Game Boy Advance.

2.4.1 Sprites

La Game Boy Advance implementa a nivel de hardware un modo para manejar y mostrar personajes u objetos dentro del juego. *Sprite* es el término que normalmente se utiliza para describir cualquier entidad única que se pueda mover dentro de un juego. El programador define los parámetros correspondientes y la *PPU* se encargará de mostrar la imagen con los *tiles* y colores que el programador haya posicionado en la *VRAM* y *RAM* de paleta, respectivamente.

Por suerte para el programador, las tareas que se pueden considerar “complejas” las realiza la *PPU* (por ejemplo, organizar los *tiles*, tener en cuenta los bits por píxel y mostrarlos por pantalla). Aún así, tendrá que realizar los siguientes pasos para acabar mostrando el *sprite* deseado:

1. Separar los colores utilizados en una imagen, como un personaje o un objeto dentro del juego. Como resultado obtendremos una imagen con los índices apuntando a la paleta de colores.
2. Dividir la imagen en bloques de 8x8, que es el tamaño base con el que trabaja la Game Boy Advance. Es en este punto donde se hace evidente la utilidad de herramientas como Grit, que automatiza la mayor parte de este proceso.
3. Especificar las propiedades que tendrá el sprite en la OAM. Para ello, previamente, se han tenido que copiar los colores utilizados al espacio de memoria reservado para la

2.4. PICTURE PROCESSING UNIT (PPU)

paleta de colores y la distribución de los *tiles* (los índices) a la VRAM. Cada objeto almacenado en este espacio de memoria, que soporta hasta 128 *sprites*, cuenta con tres atributos de 16 bits para definir sus propiedades.

4. Indicar los valores que tendrán las matrices de transformación de 2×2 que se usen para rotar y escalar los *sprites*. Este punto es opcional y se detallará más en profundidad en el apartado 2.4.1.2).

La distribución de cada uno de estos atributos, junto con los valores de la matriz de transformación, siguen el formato expuesto en la Figura 2.17 [3]. En dicha figura se utiliza p para denotar al personaje u objeto, $Attr$ para cada uno de sus atributos y a para referirse a los valores de la matriz. Es necesario recalcar que, mientras que la información de un *sprite* es almacenada de forma contigua, los valores de la matriz de afinidad se encuentran dispersos, con 6 bytes de diferencia entre cada uno.

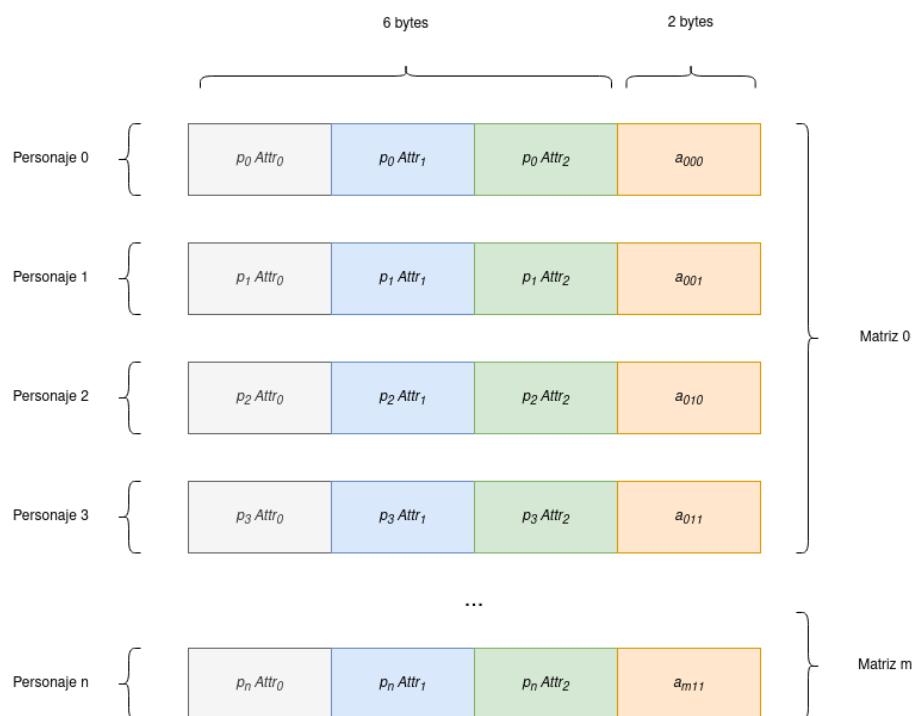


Figura 2.17: Distribución de los atributos en la OAM.

2.4.1.1. Atributos

Como se ha mencionado, las propiedades de los sprites se definen mediante tres atributos conocidos como 0, 1 y 2, que se explican a continuación. No obstante, para más información se recomienda consultar [2, 3]. Además, para las funciones que se puedan apreciar visualmente, se incluye un ejemplo mostrando su funcionamiento en esta sección y en la 2.4.4.

Atributo 0

Este atributo alberga las siguientes propiedades:

- Bits {F-E}: Junto con los bits {F-E} del atributo 1, definen el tamaño de la imagen del objeto o personaje. Los tamaños posibles se pueden ver en la Tabla 2.18.

Atr. 0\1	0	1	2	3
0	8x8	16x16	32x32	64x64
1	16x8	32x8	32x16	64x32
2	8x16	8x32	16x32	32x64

Tabla 2.18: Tamaño del *sprite* según los valores de {F-E} en los atributos 0 y 1.

- Bit {D}: Bits por píxel. Este valor hay que ajustarlo dependiendo de la cantidad de colores que utilicen los *tiles*. Si se iguala a 0, se utilizarán 4 bpp (bits por píxel), admitiendo una selección de 16 colores diferentes. En caso de igualarlo a 1, se utilizarán 8 bpp, ofreciendo una selección considerablemente mayor, con 256 colores diferentes. La desventaja es que cada *tile* ocupará más espacio y, por lo tanto, habrá menos disponibles.
- Bit {C}: Bit de activación del efecto “mosaico”. Este es uno de los efectos gráficos que ofrece la *PPU* de la consola. Su función es “pixelar” todavía más el *sprite* (o fondo, como se verá más adelante) en cuestión. Para utilizarlo es necesario activar este bit además de configurar el registro mostrado en la sección 2.3.1. Para más información consultar 2.4.4.3.
- Bits {B-A}: Efectos adicionales. Están disponibles el *blending* (que se consigue igualando este parámetro a 1) y el renderizado de máscara (que se consigue igualando este parámetro a 2). En caso de no especificar ningún valor (0), la *PPU* renderizará el *sprite* de forma normal.

El primer modo, como su nombre sugiere, mezcla el *sprite* (o fondo) con cualquier cosa que se encuentre detrás. El nivel de transparencia dependerá de los valores que se utilicen en los tres registros que usa, que se detallan posteriormente.

El segundo no mostrará el *sprite*, pero su área será utilizada para controlar cuales de los objetos o fondos que se encuentran dentro se renderizan.

- Bits {9-8}: En estos 2 bits, el programador podrá configurar cómo renderizar el *sprite*. En concreto, si el objeto se rotará o cambiará de tamaño (igualando su valor a 1 o 3, respectivamente), si se ocultará (igualando a 2) o si su representación será la básica (igualando a 0), mostrando el objeto sin ningún cambio adicional.

Hay que resaltar que, dado que la Game Boy Advance inicializa la memoria a 0, el programador tendrá que asegurarse de que este parámetro sea 2 para todos los objetos no usados, aunque no utilice los 128 sprites disponibles. En caso contrario, en la esquina superior izquierda se mostrará el primer tile encontrado en la VRAM. Este problema se ilustra en la Figura 2.18.

2.4. PICTURE PROCESSING UNIT (PPU)



Figura 2.18: Resultado de no ocultar los *sprites* no utilizados.

- Bits {7-0}: Las coordenadas Y del objeto siguiendo el formato especificado en la Figura 2.15.

La vista general de lo descrito anteriormente se puede observar en la Tabla 2.19. Para modificar las propiedades del atributo, el programador tiene que utilizar operadores a nivel de bit.

Bits	Descripción
F-E	Tamaño del <i>sprite</i> .
D	Color utilizado. 0 para 4 bpp y 1 para 8 bpp.
C	Efecto mosaico.
B-A	Efectos especiales.
9-8	Modo de afinidad del objeto.
7-0	Coordenada Y del objeto.

Tabla 2.19: Atributo 0.

Atributo 1

El atributo 1 incluye parámetros para invertir horizontal y verticalmente el *sprite*, y otros que complementan los vistos en el atributo 0. La distribución es la mostrada a continuación:

- Bits {F-E}: Junto con los bits {F-E} del atributo 0, definen el tamaño de la imagen utilizada para representar el objeto o personaje. Véase la Tabla 2.18.
- Bits {D-C}: Estos dos bits permiten al programador invertir tanto horizontal como verticalmente la imagen utilizada para representar el *sprite*. Este recurso favorece la reutilización de los *tiles* del *sprite*, haciendo posible al programador invertir el objeto o personaje sin tener que guardar en memoria la imagen invertida. En la Figura 2.19 se muestra el efecto de todos los posibles valores que puede tener este campo. El primer *sprite* aparece representado en su forma original, con el valor 0. El segundo *sprite* se muestra invertido horizontalmente, con el valor 1. El tercer *sprite* aparece invertido verticalmente, con el valor 2. Finalmente, el cuarto *sprite* se representa invertido tanto horizontal como verticalmente, con el valor 3.

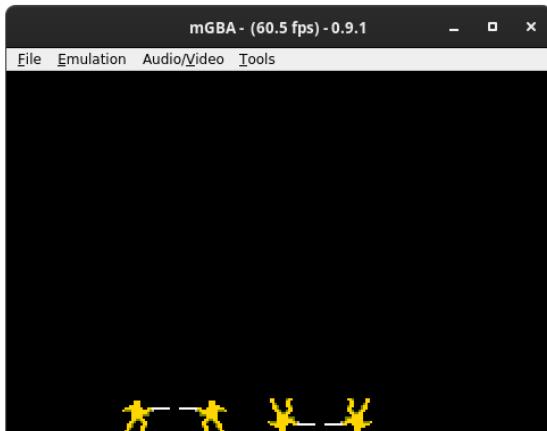


Figura 2.19: Demostración de los *flags* para invertir un *sprite* de diversas maneras.

- Bits {D-9}: En el caso de renderizar el *sprite* a partir de una matriz, igualando el campo {D-9} del atributo 0 a 1, se especificará el índice de la matriz de afinidad a utilizar. Este valor no se podrá utilizar conjuntamente con los *flags* de inversión mencionados anteriormente, dado que se superponen. No obstante, no es necesario teniendo en cuenta que se puede invertir mediante la matriz de transformación.
- Bits {8-0}: Define la coordenada X (se usará junto con la coordenada Y definida en el atributo 0).

La Tabla 2.20 muestra, de manera resumida, los bits del atributo 1 junto con una breve descripción de su uso.

Bits	Descripción
F-E	Tamaño del <i>sprite</i> .
D-C	<i>Flags</i> para invertir el <i>sprite</i> .
D-9	Índice de la matriz de transformación.
8-0	Coordenada X del objeto.

Tabla 2.20: Atributo 1.

Atributo 2

El atributo 2 especifica parámetros relacionados con el renderizado de la imagen, como a partir de qué *tile* se encuentra la imagen del objeto, la prioridad de renderizado y la paleta de colores a utilizar.

- Bits {F-C}: Permiten indicar la paleta de colores a utilizar (el índice). Al igual que los *flags* utilizados para invertir los *sprites*, este es otro parámetro que puede ser de gran utilidad, dado que puede servir para generar personajes “diferentes” minimizando el uso de recursos. Un ejemplo de ello lo podemos ver en la Figura 2.20.

2.4. PICTURE PROCESSING UNIT (PPU)



Figura 2.20: Utilizando los mismos *tiles* se muestran diferentes *sprites* alternando la paleta de colores de cada uno.

- Bits {B-A}: Este valor determina el orden con el que se renderizan los *sprites* por pantalla. Cuanto mayor sea el número, antes se “pintará” y, por lo tanto, el objeto quedará en segundo plano al superponerse con otro de menor prioridad.
- Bits {9-0}: En estos 10 bits se especifica el primer *tile* del objeto a “pintar”. Dado que en parámetros previos se indican el tamaño de la imagen y los colores utilizados, la *PPU* es capaz de saber cuántos *tiles* tiene que procesar simplemente a partir del primer índice. Este valor no solo sirve para mostrar diferentes *sprites* por pantalla, sino que facilita la realización las animaciones de *sprites* al programador. Este solo tendría que cambiar dicho valor cada cierto tiempo.

La Tabla 2.21 resume los bits utilizados, junto con una breve descripción, del atributo 2.

Bits	Descripción
F-C	Especifica la paleta a utilizar.
B-A	Prioridad de renderizado.
9-0	Índice del primer <i>tile</i> del objeto.

Tabla 2.21: Atributo 2.

2.4.1.2. Matriz de transformación

Como se ha mencionado en los parámetros de los atributos 0 y 1, la *PPU* permite al programador rotar, escalar y recortar el sprite mediante transformaciones afines. Este tipo de transformaciones, en el caso de la Game Boy Advance, se calculan con una matriz 2x2 al tratarse de un espacio 2D.

El concepto es relativamente simple. Se tiene un punto (o conjunto de puntos) definido por la ecuación (2.3) que se desea procesar. Para hacerlo, se multiplicará por la matriz de transformación, que tendrá la estructura descrita en la ecuación (2.4). Dependiendo de la operación que se quiera realizar, los valores a_{00} , a_{01} , a_{10} y a_{11} irán variando. El punto resultante, p , queda definido según la ecuación (2.5) [18]:

$$p = \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.3)$$

$$A = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \quad (2.4)$$

El problema que complica ligeramente la situación al programador es que la GBA “mapea” los valores desde la pantalla a las texturas. Dado que el cálculo descrito en la ecuación (2.5) es para convertir una coordenada del espacio de texturas a la pantalla, se tendrá que invertir la matriz de transformación para conseguir el resultado deseado [19]. La ecuación final quedaría como se puede observar en la ecuación (2.6).

$$p' = Ap \quad (2.5)$$

$$p = A^{-1}p' \quad (2.6)$$

En cuanto al cálculo de la matriz inversa se refiere, al tratarse de una matriz 2×2 , se puede utilizar la fórmula mostrada en (2.7) siguiendo uno de los teoremas explicados en [19]. Aplicada a la ecuación (2.4), quedaría como se observa en la ecuación (2.8).

$$A^{-1} = \frac{1}{|A|} adj(A)^t \quad (2.7)$$

$$\begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix}^{-1} = \frac{1}{a_{00}a_{11} - a_{01}a_{10}} \begin{bmatrix} a_{11} & -a_{01} \\ -a_{10} & a_{00} \end{bmatrix} \quad (2.8)$$

A continuación, se van a mostrar varios ejemplos junto con la matriz de transformación utilizada. Se omitirá la matriz de identidad al no modificar la imagen resultante.

Aprovechando que este recurso permite una mayor flexibilidad que los *flags* de inversión del atributo 1, se probarán con ángulos que una simple inversión no puede proporcionar. El primero de ellos es un ángulo de 45° , cuya matriz, al ser de rotación, utiliza funciones coseno (\cos) y seno (\sin) [18]. La matriz quedaría como se puede observar en la ecuación (2.9). Al hacer el cálculo, se obtiene que la matriz inversa es idéntica a la original.

$$A = A^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (2.9)$$

2.4. PICTURE PROCESSING UNIT (PPU)

La Figura 2.21 incluye el resultado de usar esta matriz con los ejemplos anteriores.

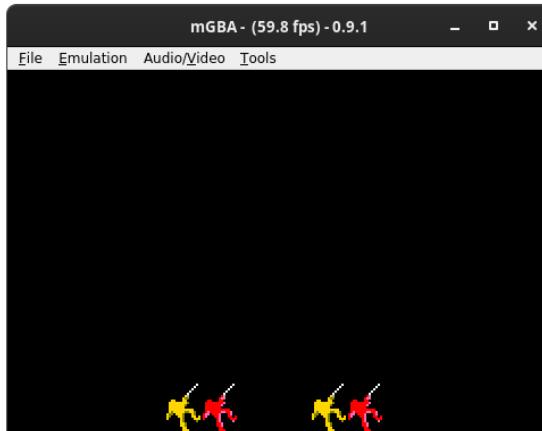


Figura 2.21: Rotación de *sprites* por medio de la matriz de transformación.

Otra matriz utilizada a menudo es la de escalado. Como su nombre indica, permite aumentar el tamaño del *sprite* tanto en el eje X como en el Y [18]. La matriz utilizada en este ejemplo se muestra en la ecuación (2.9), siendo a el factor de escalado en el eje X y d el factor de escalado en Y. La inversa quedaría como se muestra en la ecuación (2.11).

$$A = \begin{bmatrix} a & 0 \\ 0 & d \end{bmatrix} \quad (2.10)$$

$$A^{-1} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/d \end{bmatrix} \quad (2.11)$$

El resultado de multiplicar por 2 tanto el eje X como Y en el programa de demostración es el que se muestra en la Figura 2.22.

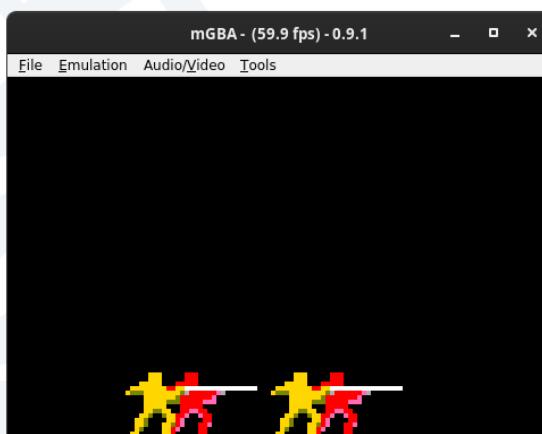


Figura 2.22: Escalado *sprites* por medio de una matriz de transformación.

En caso de querer rotar y escalar el *sprite* a la vez, esto se puede realizar combinando las fórmulas mostradas anteriormente (ecuaciones (2.12) y (2.13)), tal y como se muestra en la ecuación (2.14).

$$A_e^{-1} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/d \end{bmatrix} \quad (2.12)$$

$$A_r^{-1} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (2.13)$$

$$A_{re}^{-1} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/d \end{bmatrix} \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} = \begin{bmatrix} \cos(\theta)/a & -\sin(\theta)/a \\ \sin(\theta)/d & \cos(\theta)/d \end{bmatrix} \quad (2.14)$$

Esto da lugar a la Figura 2.23, en la que se combina una rotación de 30º con un factor de escala de 1.5 en los dos ejes.



Figura 2.23: Rotación y escalado de *sprites* por medio de la matriz de transformación.

Lamentablemente, este sistema trae consigo varias limitaciones y problemas. El primero de ellos es la imposibilidad de rotar, escalar y realizar transformaciones de corte si el *sprite* resultante sobrepasa el área original establecida (el tamaño original del *sprite*, véase la Tabla 2.18). Esto se puede resolver igualando el parámetro {9-8} del atributo 0 a 3, duplicando el área original del objeto. Sin embargo, si el *sprite* sobrepasa esa área, también se verá cortado (véase la Figura 2.24). Además, el programador deberá tener en cuenta el cambio en las coordenadas X e Y al posicionar el *sprite* en pantalla.



Figura 2.24: Sprite que sobrepasa el área permitida.

La segunda limitación es el uso, en varios casos, de operaciones en punto flotante y funciones *cos* y *sin*. El hecho de que la consola no cuente con una *FPU* (*Floating Point Unit*) ralentiza de manera considerable el cálculo. Por lo tanto, habrá que evitar este tipo de operaciones no solo al calcular la matriz de transformación, sino en otros aspectos del programa también. De hecho, el dispositivo no acepta *floats* como valores de la matriz de transformación, solo permite números de coma fija (*fixed point numbers* en inglés). El uso de este tipo de números se verá con más detenimiento en la Sección 3.2.

2.4.2 Bitmaps

A diferencia de sus predecesoras, la Game Boy Advance proporciona al programador una forma directa de manipular los píxeles que aparecen por pantalla, lo que se conoce como *bitmap*. Este modo, considerado como el más simple de los que ofrece el dispositivo, trata el contenido de la sección de memoria VRAM como los píxeles de la pantalla. Por lo tanto, con escribir en esta dirección de memoria, la pantalla inmediatamente representará los cambios reflejados sin tener que cambiar o configurar ningún registro o atributo adicional.

Este modo no es del todo práctico ni eficiente, dado que se tiene que suministrar el valor de cada uno de los píxeles que aparecen por pantalla. Sin embargo, sigue siendo válido para mostrar imágenes estáticas, realizar juegos relativamente simples (como *Pong*) o 3D como *Doom*⁸.

Concretamente, existen tres modos *bitmap* para manipular directamente los píxeles. Se diferencian en aspectos como el tamaño del *canvas*, si se recurre a una paleta de colores o si se cuenta con un buffer adicional [2].

La Tabla 2.22 resume las diferencias principales de los modos, de los que se dan más detalles en las siguientes subsecciones.

⁸https://doom.fandom.com/wiki/Game_Boy_Advance

Modo	Ancho	Alto	bpp	Uso de paleta	Buffer adicional
3	240	160	16	No	No
4	240	160	8	Sí	Sí
5	160	128	16	No	Sí

Tabla 2.22: Modos bitmap de la Game Boy Advance. Tabla basada en [3]

2.4.2.1. Modo 3

El modo 3 ofrece al programador un buffer de 240x160 de 16 bits por píxel, resolución que coincide con la pantalla de la Game Boy Advance. De los 16 bits, se utilizan 15 para cada color como se mostró en la Tabla 2.17. En total, este buffer ocupa 75 KB, desde la dirección 0x06000000 (VRAM) [2].

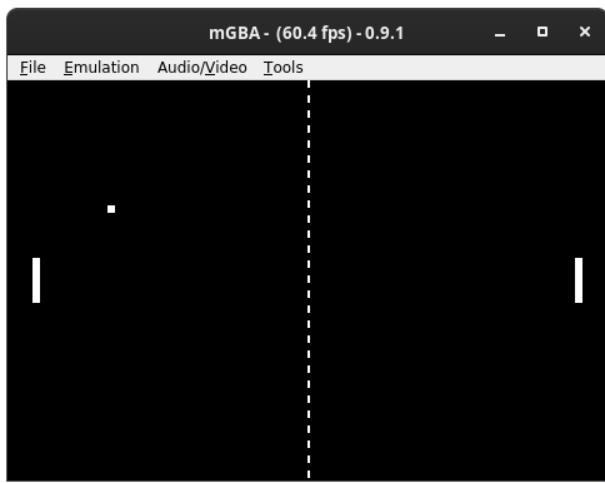


Figura 2.25: Demostración práctica del uso del Modo 3.

En la Figura 2.25 se puede observar un ejemplo de un juego simple que utiliza el modo 3 de la PPU.

Como dato adicional, es posible utilizar conjuntamente los modos *bitmaps* y el modo de *sprites*. Sin embargo, en el caso del modo 3 se disminuye el espacio destinado a *tiles* un 50 % [3].

2.4.2.2. Modo 4

El modo 4, a pesar de utilizar un *canvas* idéntico al modo 3, de 240x160 también, incluye varias opciones interesantes para el programador.

Para empezar, en lugar de guardar los valores de los píxeles directamente en la VRAM, guarda los índices de la paleta de colores que se encuentra en la RAM de paleta. A la hora de modificar el color de una gran cantidad de píxeles, esto puede suponer una ventaja en términos de rendimiento con respecto al modo 3. Dada la inherente inefficiencia que supone modificar cada uno de los píxeles que aparecen por pantalla, tener índices apuntando a una paleta de colores permite al programador cambiar un gran número de píxeles simplemente cambiando uno de los valores de la paleta.

2.4. PICTURE PROCESSING UNIT (PPU)

Además, al utilizar índices, que ocupan 8 bits, el espacio dedicado a la imagen se reduce a la mitad. Este espacio “extra” lo aprovecha un buffer que ofrece el modo 4. Este buffer adicional se encuentra en la dirección de memoria 0x0600A000. El buffer principal del modo 4 acaba en 0x06009600, por lo que no interfiere con el segundo buffer disponible. La idea es que el programador escriba el próximo “frame” en el buffer no activo para posteriormente modificar el buffer activo. Así se consigue que los cambios realizados no hagan que la imagen final sufra *tearing*, es decir, que no se superpongan imágenes de forma indebida [2, 3].

2.4.2.3. Modo 5

El modo 5 ofrece tanto la posibilidad de escribir directamente en VRAM sin tener que utilizar una paleta con color de 15 bits, como un segundo buffer donde preparar el siguiente frame. Aunque, en detrimento del jugador, usa un canvas de dimensiones reducidas, de 160x128 [2]. La Figura 2.26 da una idea al lector del área que abarca este modo.



Figura 2.26: Ejemplo práctico del modo 5.

Al igual que el modo 3, el modo 5 usa gran parte de la VRAM, lo que reduce el número de *tiles* disponibles para *sprites* [3].

2.4.3 Fondos

Los fondos de la GBA permiten especificar mediante el uso de “mapas” qué *tiles* utilizar para llenar el canvas. A pesar de que técnicamente los *bitmaps* se catalogan como fondos en el manual [2], se suelen separar los modos 0, 1, 2 y 3 de los modos 4, 5 y 6 por las grandes diferencias de funcionamiento.

A diferencia de los *sprites*, la mayor parte de la configuración se hace en los registros de configuración específicos para los fondos. Por lo tanto, lo que hay que tener en cuenta al mostrar fondos en la consola es el formato que sigue cada una de las entradas del mapa, que especifica el *tile* a utilizar y el número y tipo de fondos a emplear.

En cuanto al formato que sigue una entrada del mapa correspondiente a un fondo, tiene la siguiente distribución:

- Bits {F-C}: Índice de la paleta de colores a utilizar en caso de estar en el modo de color de 4 bpp.
- Bits {B-A}: Es similar a lo visto en el atributo 1 de los *sprites*. Permite invertir tanto horizontal como verticalmente el *tile* especificado en los bits {9-0}.
- Bits {9-0}: Índice del *tile* a utilizar.

Sin embargo, con las herramientas disponibles, el programador no tendrá que preocuparse del formato que sigue cada entrada en el mapa. Las utilidades como Grit ya automatizan la generación de los mapas de cada fondo.

Lo que si tiene que tener en cuenta el programador son los modos que configurará en los registros mostrados en la sección 2.3.1.3, indicando cuántos de los 4 fondos disponibles planea utilizar y si utilizarán una matriz de transformación. Dependiendo de los modos seleccionados, unos fondos u otros quedarán inhabilitados o limitados a un uso específico. En la Tabla 2.23 se denota con “r” a los fondos normales y con “t” a los fondos que utilizan la matriz de transformación.

Modo	Fondo 0	Fondo 1	Fondo 2	Fondo 3
0	r	r	r	r
1	r	r	t	-
2	-	-	t	t

Tabla 2.23: Modos para los fondos de la Game Boy Advance.

Otro aspecto que se ve afectado por el modo seleccionado es el tamaño del fondo. Tal y como se puede observar en la Tabla 2.24, el tamaño del fondo depende en parte de si utiliza una matriz de transformación. El valor de la columna *flag* es el que le corresponde a los bits {F-E} en el registro de control. Los valores mostrados utilizan como base el *tile* de 8x8 y, para dar una mejor idea del tamaño, se incluye entre paréntesis el tamaño en píxeles del fondo.

Flag	Tamaño para r	Tamaño para t
00	32x32 (256x256)	16x16 (128x128)
01	64x32 (512x256)	32x32 (256x256)
10	32x64 (256x512)	64x64 (512x512)
11	64x64 (512x512)	128x128 (1024x1024)

Tabla 2.24: Tamaños disponibles para cada uno de los fondos.

Una demostración de un fondo renderizado se puede observar en la Figura 2.27. Se trata de un fondo de 32x32 (o 256x256 píxeles) en movimiento actualizando los valores de los registros de offset.

2.4. PICTURE PROCESSING UNIT (PPU)



Figura 2.27: Fondo 32x32.

Además de las diferencias de tamaño observadas en la Tabla 2.24, los fondos que utilizan la matriz de transformación también ven las entradas del mapa modificadas. En este caso se prescinde del índice de paleta y de la orientación de cada *tile*. De esta forma, cada entrada del mapa acaba siendo solo el índice del mapa y su tamaño se ve reducido a 1 byte [3].

Basándose en los mismos conceptos vistos en la sección 2.4.1.2, hay que definir los valores de una matriz 2x2 para modificar el fondo en cuestión. En la Figura 2.29 se ofrecen dos variantes del fondo mostrado en la Figura 2.28 cambiando el tamaño y la rotación del mismo.

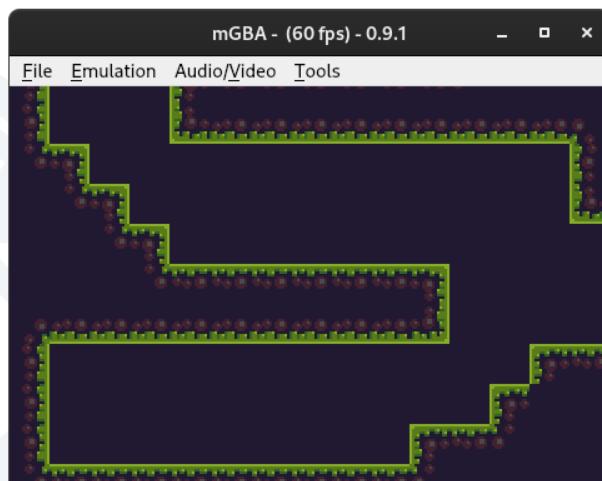


Figura 2.28: Fondo 32x32.



(a) Fondo escalado un 100 %.

(b) Fondo rotado por 45º.

Figura 2.29: Matriz de transformación sobre el fondo de la Figura 2.28.

2.4.4 Efectos especiales

La GBA ofrece al programador diversas formas de pulir y refinar sus juegos con efectos especiales acelerados por hardware. Los cuatro efectos especiales son el **blending**, **efectos fade in/out**, el **modo “mosaico”** y el **modo ventana**.

Todos ellos se pueden aplicar tanto a *sprites* como a fondos. El efecto no varía si se aplica a un objeto o a un fondo, por lo que los ejemplos mostrados valen para ambos.

2.4.4.1. Blending

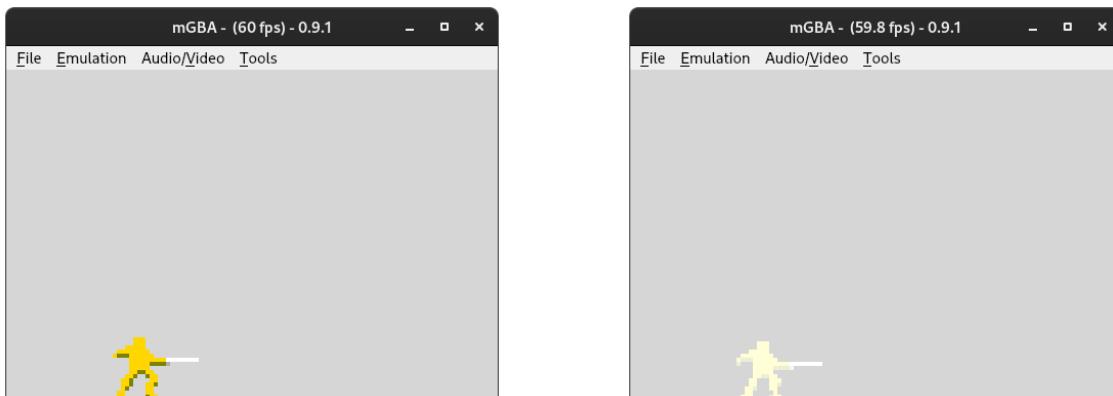
Como su nombre sugiere, permite mezclar los colores que se encuentran detrás del *sprite*, ya sea un fondo u otro *sprite*. El nivel del efecto se configura en los registros adicionales vistos en el sección 2.3.1.

El registro que guarda los coeficientes de mezcla, BLDALPHA (para más información consultar el sección 2.3.1), calcula el valor de cada píxel siguiendo la fórmula mostrada en la ecuación (2.15). El cálculo se realiza para cada color de forma separada, de ahí el mínimo de 31, número que representa el máximo valor que puede alcanzar un color de 5 bits.

$$valor = \min(31, color_{frontal} * coef_{frontal} + color_{posterior} * coef_{posterior}) \quad (2.15)$$

Una demostración del efecto que se consigue con el *sprite* utilizado anteriormente combinado con un fondo gris se puede observar en la Figura 2.30.

2.4. PICTURE PROCESSING UNIT (PPU)



(a) Demostración sin *blending*.

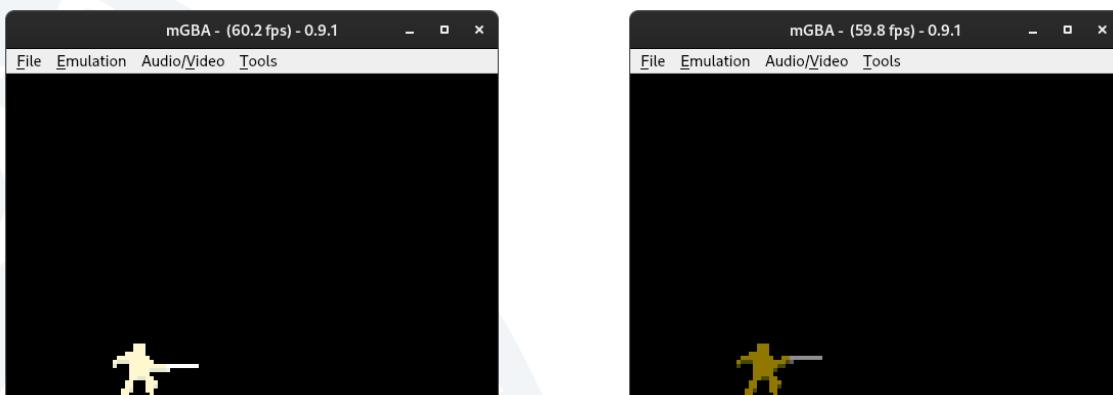
(b) Demostración con *blending*.

Figura 2.30: Efecto conseguido configurando el efecto *blending* de la consola.

2.4.4.2. Efectos *fade*

Para facilitar las transiciones entre escenas, la *PPU* ofrece la posibilidad de cambiar gradualmente de una imagen a otra completamente negra, y de una imagen a otra completamente blanca.

Configurando los registros BLDCNT y BLDY vistos en la sección 2.3.1, se modifica la intensidad de la imagen mostrada por pantalla. Si se actualiza el valor de BLDY se conseguirá el deseado efecto *fade in/out*. En las Figura 2.31 se muestra un aumento y disminución de la intensidad sobre una imagen.



(a) Aumento de la intensidad.

(b) Disminución de la intensidad.

Figura 2.31: Demostración del cambio de intensidad a través de BLDCNT y BLDY.

2.4.4.3. Modo mosaico

Este modo reduce el nivel de detalle de un objeto o fondo. Para ello, la GBA internamente aplica el valor de un píxel a los vecinos posicionados debajo y a su derecha. El número de píxeles que acaban compartiendo el mismo valor quedará determinado por el número de

distorsión horizontal y vertical que se indique en el registro correspondiente. Por ejemplo, si el píxel 0 tiene un valor determinado y la distorsión horizontal es de 10, entonces los píxeles 1-10 tendrán el mismo color que el píxel 0. Siguiendo el ejemplo, si la distorsión horizontal es de 10, el siguiente píxel a tener en cuenta sería el 11. Un ejemplo del efecto se puede observar en la Figura 2.32.



Figura 2.32: Sprite renderizado con efecto “mosaico” (izquierda), junto a su representación sin alterar (derecha).

2.4.4.4. Modo ventana

El último efecto destacable que ofrece la GBA es la posibilidad de crear ventanas en pantalla y controlar qué contenido es renderizado dentro y fuera de ellas. La consola proporciona 2 ventanas para fondos, y cada *sprite* puede configurarse para controlar qué se renderiza dentro y fuera del área que ocupa. En caso de superponerse las dos ventanas disponibles, el contenido renderizado es la unión del contenido permitido en ambas. Por otra parte, el contenido no renderizado es todo lo que no entra en la unión mencionada anteriormente.

La Figura 2.33 muestra un ejemplo sobre el fondo de la Figura 2.27. En este ejemplo se observa una ventana que ocupa la mitad de la pantalla en la que se permite el renderizado de dos fondos (el fondo con el paisaje y el fondo con la vegetación). Fuera de la ventana, solo se permite el renderizado de uno de los fondos (el de la vegetación).



Figura 2.33: Demostración del funcionamiento de una ventana.

3 DISEÑO Y DESARROLLO DEL JUEGO

El objetivo principal del proyecto, desde el punto de vista técnico, es el desarrollo de un videojuego para la Game Boy Advance usando el menor número posible de librerías externas. Forzando así un mayor entendimiento de cómo funciona el dispositivo junto con la lógica requerida para poder hacer ciertas funciones aunque ya se proporcionen en la mayoría de librerías existentes para la consola.

Completadas las dos primeras tareas de la planificación, el estudio de la arquitectura y de técnicas de programación, se decide implantar una visión para el proyecto. Para ello, el autor revisita juegos de su agrado en los que basar el proyecto.

Una vez finalizado el proceso de búsqueda de inspiración, la idea inicial del proyecto fue la de recrear las mecánicas y jugabilidad del popular juego de PC *Nidhogg* (véase la Figura 3.1).



Figura 3.1: Nidhogg, un juego para ordenador.

El autor empezó a diseñar los *assets* del juego recreándolos en *Pixelorama*, tal y como se muestra en la Figura 3.2.



Figura 3.2: Assets para el primer prototipo del juego.

Sin embargo, al finalizar la primera iteración se observaron los siguientes riesgos que afectaban el desarrollo del juego:

- Multijugador: Una de las fortalezas del juego original fue la posibilidad de conectar con otros jugadores y competir a través de la red, o en la propia máquina con múltiples mandos. Una implementación equivalente tendría que hacer uso de la comunicación serial que ofrece la consola, funcionalidad que dificulta probar el juego en hardware y que la mayoría de emuladores no soporta completamente, ya sea de forma local o en

red. En la Figura 3.3 se muestra el estado actual de dicha funcionalidad en el emulador mGBA.

Planned features

- Networked multiplayer link cable support.
- Dolphin/JOY bus link cable support.
- MP2k audio mixing, for higher quality sound than hardware.
- Re-recording support for tool-assist runs.
- Lua support for scripting.
- A comprehensive debug suite.
- Wireless adapter support.

Figura 3.3: Funcionalidades por implementar.

- Arte: Otro error que se cometió en las primeras fases de desarrollo fue subestimar el tiempo requerido para crear los recursos o *assets* del juego. Por este motivo, el autor decidió hacer uso de recursos *open source* creados por la comunidad, utilizándolos siempre según las condiciones de las licencias utilizadas.

Habiendo reevaluado proyecto, se optó por realizar un juego de plataformas. El diseño giraría en torno a un personaje con diversas habilidades, siendo necesario familiarizarse con cada una de ellas para poder superar el juego.

En la segunda iteración se prototiparía un juego medieval (véase la Figura 3.4) utilizando los *assets* creados por *Dagon*¹ y acorde con las características descritas previamente. Sin embargo, antes de comenzar con el tercer sprint, se descartaría el diseño al no convencer al autor del proyecto.

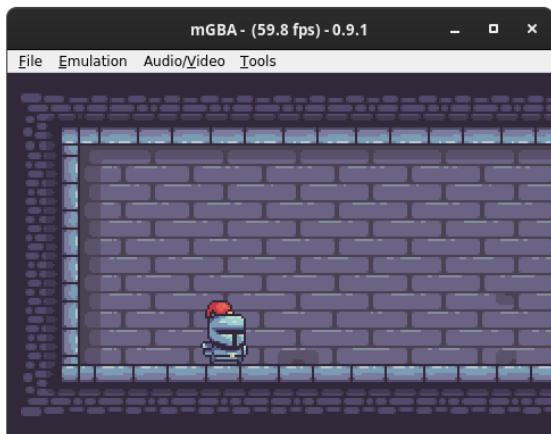


Figura 3.4: Segundo prototipo del juego.

Finalmente, a partir del tercer *sprint* se optó por utilizar los *assets* creados por *o_lobster*². En la Figura 3.5 se ve una muestra del conjunto de recursos seleccionado en última instancia.

¹<https://im-dagon.itch.io/dungeon-pack>

²<https://o-lobster.itch.io/platformmetroidvania-pixel-art-asset-pack>

3.1. ESPECIFICACIONES



Figura 3.5: Arte escogido para el desarrollo del juego.

A continuación, se detallarán los requisitos y las rutinas utilizadas para poder desarrollar el juego.

3.1.1 ESPECIFICACIONES

En esta sección se describe la solución propuesta por el autor para satisfacer las necesidades del juego. La estructura de esta sección se deriva de asignaturas cursadas previamente [20]. Los acrónimos que se utilizarán en las próximas secciones son los siguientes:

- **OBJ:** Objetivo (o requisitos generales del juego).
- **PU:** Perfil de Usuario.
- **ACT:** Actor del producto desarrollado.
- **RF:** Requisito Funcional.
- **RI:** Requisito de Información.
- **RNF:** Requisito No Funcional.

3.1.1.1 Requisitos generales

A continuación, se describen los requisitos generales del juego en forma de objetivos.

OBJ-1	Procesamiento de la entrada del jugador
Versión	0.5
Descripción	El juego deberá procesar la entrada del usuario y actualizar los valores por pantalla, ya sea en los menús o en los niveles del juego.
Comentarios	Ninguno.

Tabla 3.1: Objetivo 1.

OBJ-2	Procesamiento de enemigos
Versión	0.5
Descripción	El juego deberá actualizar de forma independiente la posición y estado de cualquiera de los enemigos que aparezcan en cada nivel del juego.
Comentarios	Ninguno.

Tabla 3.2: Objetivo 2.

OBJ-3	Procesamiento de objetos
Versión	0.5
Descripción	El juego deberá actualizar de forma independiente la posición y estado de cualquiera de los objetos que aparezcan en cada nivel del juego.
Comentarios	Ninguno.

Tabla 3.3: Objetivo 3.

OBJ-4	Procesamiento de las colisiones
Versión	0.5
Descripción	El juego deberá procesar las colisiones del jugador con objetos, enemigos y el entorno.
Comentarios	Ninguno.

Tabla 3.4: Objetivo 4.

OBJ-5	Gestión del progreso del jugador
Versión	0.5
Descripción	El juego deberá gestionar el progreso del jugador, de manera que el usuario no tenga que intervenir para tener una experiencia de juego continua.
Comentarios	Ninguno.

Tabla 3.5: Objetivo 5.

3.1. ESPECIFICACIONES

3.1.2 Casos de uso

En esta sección se incluye el diagrama de casos de uso del juego y la especificación de los perfiles y actores que interactúan con el software. El diagrama se muestra en la Figura 3.6.

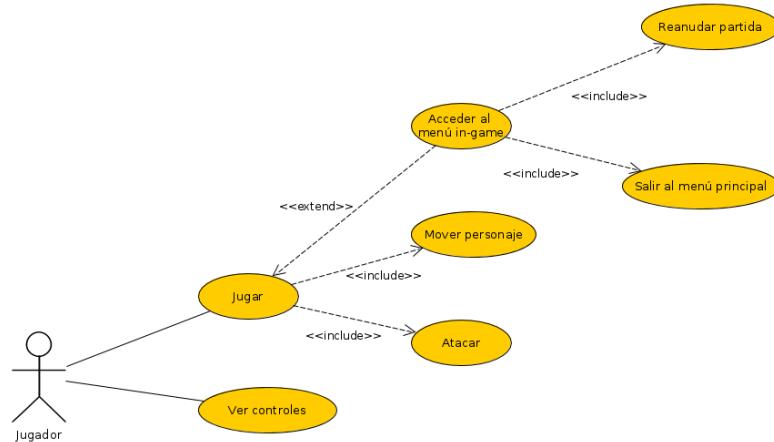


Figura 3.6: Diagrama de casos de uso.

Dado que el procesamiento del juego transcurre de forma local en la consola, mediante un dispositivo único, solo existe un actor en el sistema y por ende, un perfil de usuario.

PU-1	Jugador
Descripción	Perfil utilizado para identificar cualquier usuario que juegue al producto desarrollado, tanto en emuladores como en hardware.
Actores asociados	ACT-1 (Jugador)
Comentarios	Ninguno.

Tabla 3.6: Perfil de Usuario 1.

ACT-1	Jugador
Perfil de usuario	PU-1
Requisitos asociados	RF-1, RF-2, RF-3, RF-4, RF-5, RF-6, RF-7
Descripción	Formado por la persona que juega y utiliza el software desarrollado.
Comentarios	Ninguno.

Tabla 3.7: Actor 1.

3.1.3 Requisitos funcionales

En esta sección se incluirán los requisitos funcionales del proyecto, extraídos del diagrama de casos de uso.

RF-1	Jugar
Objetivos asociados	OBJ-1, OBJ-2, OBJ-3, OBJ-4
Requisitos asociados	RI-1
Descripción	El jugador inicia la partida.
Precondición	El jugador debe haber encontrarse en el menú principal.
Secuencia normal	<ol style="list-style-type: none"> 1. Mover el cursor al botón adecuado. 2. El jugador presiona “A”. 3. Se carga el último nivel que no haya sido superado.
Postcondición	El jugador inicia una partida.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.8: Requisito funcional 1.

RF-2	Ver controles
Objetivos asociados	OBJ-1
Requisitos asociados	Ninguno.
Descripción	El jugador selecciona la visualización de los controles.
Precondición	El jugador debe haber encontrarse en el menú principal.
Secuencia normal	<ol style="list-style-type: none"> 1. Mover el cursor al botón adecuado. 2. El jugador presiona “A”. 3. El juego carga los nombres por pantalla.
Postcondición	El jugador visualiza los controles por pantalla.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.9: Requisito funcional 2.

3.1. ESPECIFICACIONES

RF-3	Acceder al menú in-game
Objetivos asociados	OBJ-1
Requisitos asociados	Ninguno.
Descripción	El jugador accede al menú del juego. Desde ahí puede seleccionar varias opciones
Precondición	El jugador debe encontrarse en mitad de una partida.
Secuencia normal	<ol style="list-style-type: none"> 1. El jugador presiona “Start”. 2. El juego carga un menú sobre la partida actual.
Postcondición	El jugador visualiza los un menú.
Excepciones	Ninguna.
Comentarios	Las físicas del juego se ven pausadas ya que la “delta” del juego no se actualiza.

Tabla 3.10: Requisito funcional 3.

RF-4	Mover personaje
Objetivos asociados	OBJ-1, OBJ-4
Requisitos asociados	Ninguno.
Descripción	El jugador mueve el personaje principal.
Precondición	El jugador debe encontrarse en mitad de una partida.
Secuencia normal	<ol style="list-style-type: none"> 1. El jugador presiona cualquiera de los que habilitan el movimiento del personaje. 2. El juego actualiza la posición del personaje según la entrada del jugador. 3. El juego comprueba cualquier tipo de colisión para actualizar el estado del jugador o sobrescribir la posición del mismo.
Postcondición	El personaje ve actualizada su posición y estado.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.11: Requisito funcional 4.

RF-5	Atacar
Objetivos asociados	OBJ-1, OBJ-4
Requisitos asociados	Ninguno.
Descripción	El jugador ataca.
Precondición	El jugador debe encontrarse en mitad de una partida y el personaje no puede estar en el aire.
Secuencia normal	<ol style="list-style-type: none"> 1. El jugador presiona “B”. 2. El juego comprueba si existe contacto con algún enemigo. 3. En caso afirmativo, el juego procesa el ataque.
Postcondición	En caso de contacto, los enemigos ven actualizado su estado.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.12: Requisito funcional 5.

RF-6	Reanudar partida
Objetivos asociados	OBJ-1
Requisitos asociados	Ninguno.
Descripción	El jugador reanuda la partida después de haber pausado el juego.
Precondición	El jugador debe encontrarse en el menú in-game.
Secuencia normal	<ol style="list-style-type: none"> 1. Mover el cursor al botón adecuado. 2. El jugador presiona “A”. 3. El juego desocupa los recursos que utilizaba el menú in-game y reanuda la partida.
Postcondición	La partida es reanudada y la “delta” vuelve a actualizarse.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.13: Requisito funcional 6.

3.1. ESPECIFICACIONES

RF-7	Salir al menú principal
Objetivos asociados	OBJ-1
Requisitos asociados	Ninguno.
Descripción	El jugador vuelve al menú principal del juego.
Precondición	El jugador debe encontrarse en el menú in-game.
Secuencia normal	<ol style="list-style-type: none"> 1. Mover el cursor al botón adecuado. 2. El jugador presiona “A”. 3. El juego desocupa los recursos que utilizaba el nivel del juego y carga el menú principal.
Postcondición	El juego carga el menú principal.
Excepciones	Ninguna.
Comentarios	Ninguno.

Tabla 3.14: Requisito funcional 7.

3.1.4 Requisitos de información

En esta sección se incluirán los requisitos de información del proyecto. Esto solo involucra el progreso que ha hecho el jugador en previas sesiones de juego.

RI-1	“Save file”
Objetivos asociados	OBJ-5
Requisitos asociados	Ninguno.
Descripción	El juego debe almacenar el progreso realizado por el jugador.
Datos específicos	<ul style="list-style-type: none"> ■ Último nivel completado: Número entero.
Comentarios	Ninguno.

Tabla 3.15: Requisito de información 1.

3.1.5 Requisitos no funcionales

En esta sección se incluirán los requisitos no funcionales del proyecto.

RNF-1	Rendimiento
Objetivos asociados	OBJ-1, OBJ-2, OBJ-3, OBJ-4
Requisitos asociados	Ninguno.
Descripción	Con la finalidad de ofrecer una experiencia jugable agradable al jugador, el juego tiene como objetivo ofrecer una tasa de refresco de 60 fotogramas por segundo. Para ello se utiliza en la medida de lo posible cualquier opción que acelere el procesamiento del juego por medio de hardware.
Comentarios	En el caso del proyecto, se ha optado por utilizar tiles y sprites antes que el modo bitmap. También se debe tener en cuenta las limitaciones y ventajas que ofrece el hardware del dispositivo, un ejemplo de ello es utilizar en la medida de lo posible enteros de 32 bits como tipo de datos o evitar hacer escrituras de 1 byte en la VRAM.

Tabla 3.16: Requisito no funcional 1.

RNF-2	Batería
Objetivos asociados	OBJ-1, OBJ-2, OBJ-3, OBJ-4
Requisitos asociados	Ninguno.
Descripción	El juego además de tener que ofrecer un buen rendimiento, debe tener como prioridad maximizar el uso de la batería cuando sea posible.
Comentarios	De manera simplificada, el objetivo es hacer que el procesador se encuentre el máximo tiempo posible en un estado de baja energía (mediante el uso de interrupciones por ejemplo).

Tabla 3.17: Requisito no funcional 2.

³<https://pegi.info/what-do-the-labels-mean>

3.2. DESARROLLO

RNF-3	Accesibilidad y usabilidad
Objetivos asociados	OBJ-1
Requisitos asociados	Ninguno.
Descripción	El producto desarrollado debe ofrecer una experiencia familiar y similar, en cuanto a controles se refiere, a otros juegos. Además, la estética y temática debe ser aceptable para cualquier tipo de público independientemente de la edad.
Comentarios	Dado que el juego tiene que proporcionar una experiencia aceptable para cualquier tipo de público independientemente de la edad, se ha optado por ofrecer una experiencia similar a lo que describe la calificación PEGI 3 ³ .

Tabla 3.18: Requisito no funcional 3.

RNF-4	Mantenibilidad
Objetivos asociados	OBJ-1, OBJ-2, OBJ-3, OBJ-4, OBJ-5
Requisitos asociados	Ninguno.
Descripción	Facilitar la comprensión, y posible contribución en un futuro, de personas ajena al desarrollo inicial del juego.
Comentarios	El desarrollo del código se llevará a cabo acorde con las pautas marcadas por el <i>Linux kernel coding style</i> elaborado por el equipo de desarrollo del kernel Linux.

Tabla 3.19: Requisito no funcional 4.

RNF-5	Administración del progreso del jugador
Objetivos asociados	OBJ-5
Requisitos asociados	RI-1
Descripción	Crear una experiencia de juego continua, sin cortes, donde el progreso del jugador se guardará sin que este tenga que manualmente especificarlo.
Comentarios	El progreso del jugador solo se verá actualizado cuando este supere un nivel que no haya superado anteriormente.

Tabla 3.20: Requisito no funcional 5.

3.2 DESARROLLO

Una vez comentadas las especificaciones del proyecto, se dedica esta sección a comentar el funcionamiento específico de varias partes del código. Para una visión general del código se ha optado por utilizar diagramas de flujo, mientras que para una descripción específica del código se ha optado por utilizar pseudocódigo. Además, se comentarán aspectos destacables del desarrollo realizado. La estructura general se inspira en [21].

3.2.1 Menú

El diagrama de flujo del menú se puede observar en la Figura 3.7, siendo lo único destacable del proceso el procesamiento de la entrada del usuario.

Una vez transicionado del inicio a la escena donde se muestra el menú del juego, se cargan los *assets* que se utilizarán para el menú principal y se lee cualquier dato guardado de partidas previas. Posteriormente, en un bucle infinito, se procede a actualizar las animaciones y posiciones de los *sprites* y fondos del menú (el menú es dinámico). En cada iteración del bucle, se consulta el “input” del usuario y en caso de no reconocer el botón presionado como válido se vuelve al inicio del bucle, actualizando los elementos que no controla el jugador y esperando a que se acabe de renderizar el fotograma actual.

En caso de ser válido, se comprueba si el “input” mueve el cursor o presiona el botón seleccionado. En caso de mover el cursor, se actualiza la posición del *sprite*. En caso de presionar el botón seleccionado, se cargará la siguiente escena.

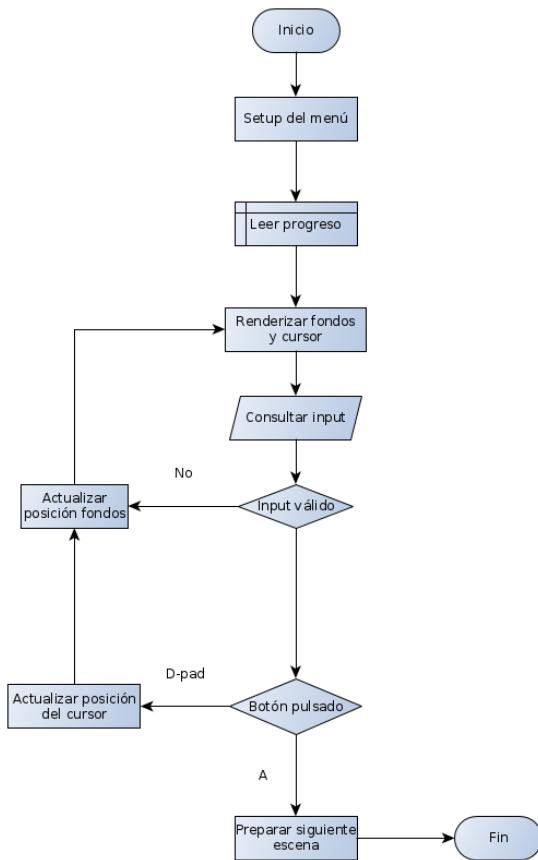


Figura 3.7: Diagrama de flujo del menú principal.

3.2.2 Nivel del juego

Para cada nivel del juego, se lee la entrada del usuario y se procesa de forma acorde, además de actualizar de forma independiente los enemigos y objetos del nivel. El proceso se puede observar en la Figura 3.8.

3.2. DESARROLLO

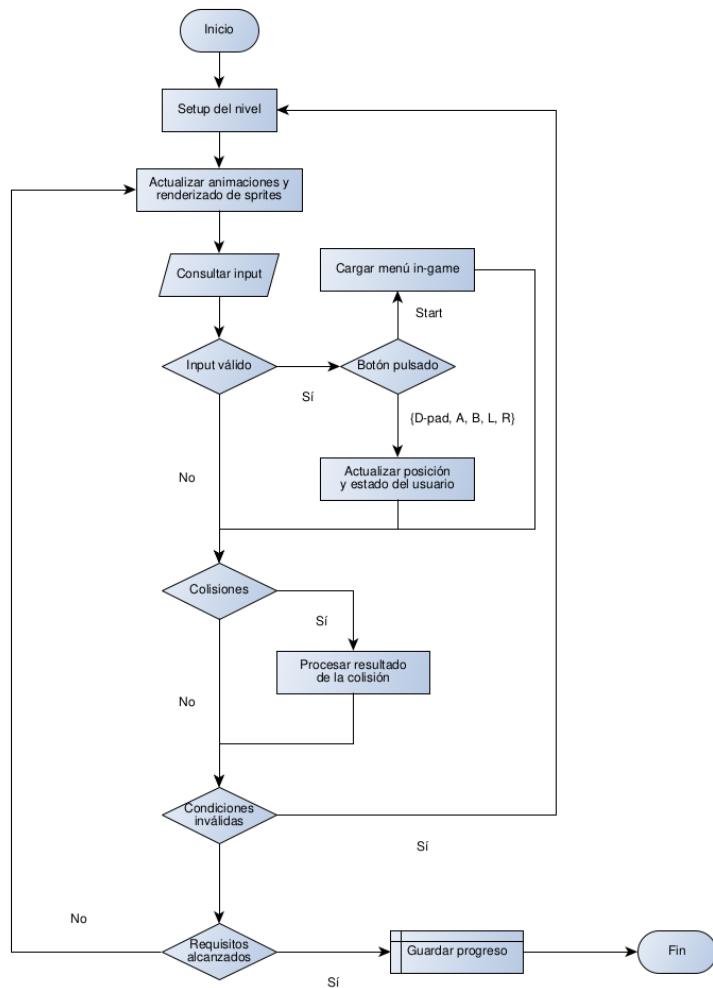


Figura 3.8: Diagrama de flujo para cada nivel del juego.

A diferencia del menú, el cual no contaba con enemigos, en el bloque de “Actualizar animaciones y renderizado de sprites” también se actualizan las posiciones de enemigos y objetos que puedan aparecer en el nivel. De forma adicional, se dedica un bloque al procesamiento de colisiones en los que se tiene en cuenta el entorno y otras entidades. Dado que para completar el nivel se tienen que cumplir ciertas condiciones, al final de cada iteración del bucle se comprueba si las condiciones se han cumplido y si el jugador debe reiniciar el nivel actual o pasar al siguiente. Es importante recalcar que el jugador no puede guardar manualmente el progreso, este solo se puede guardar en caso de que el jugador consiga pasar al siguiente nivel.

También, como se hizo referencia en el diagrama de casos de uso, se permite acceder a un menú in-game. El proceso es similar al visto en menú principal pero con ligeros cambios tal y como se puede observar en la Figura 3.9.

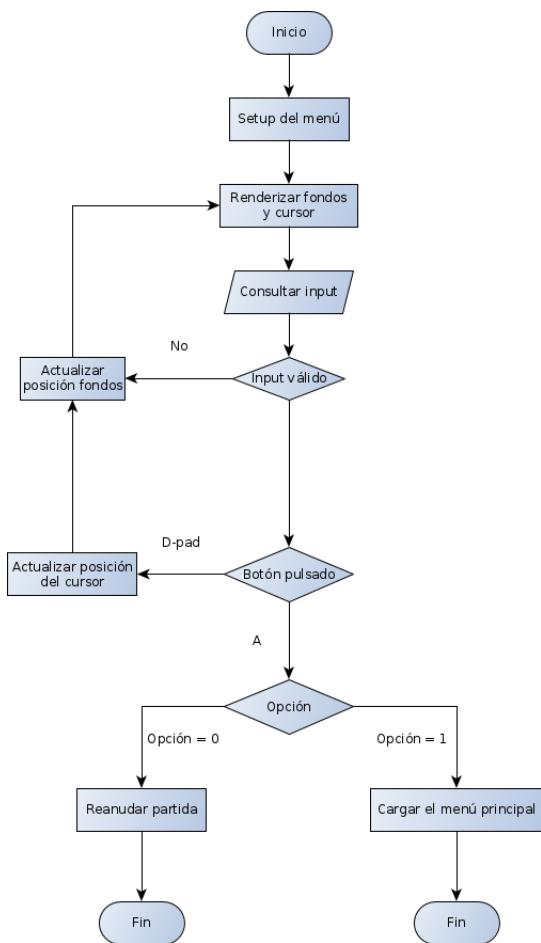


Figura 3.9: Diagrama de flujo para el menú in-game.

El bloque de “Colisiones”, mostrado en la Figura 3.8, sigue los pasos mostrados a continuación para procesar las colisiones del juego:

- Comprueba las colisiones del entorno en el eje X del personaje principal.
- Comprueba las colisiones del entorno en el eje Y del personaje principal.
- Comprueba las colisiones del entorno en el eje X de los enemigos.
- Comprueba las colisiones entre los enemigos y el personaje principal.

A continuación, se mostrará el pseudocódigo de cada uno de los apartados que conforma el bloque de “Colisiones”. Las funciones mostradas (Algoritmos 1 y 2) se han simplificado, omitiendo el procesamiento necesario al trabajar con valores desplazados por 8 bits (representación de coma fija).

Sin embargo, antes de mostrar el pseudocódigo destinado a las colisiones, se muestra el Listado 3.1, utilizada para conseguir el índice de un *tile* específico. La función no sigue

3.2. DESARROLLO

el cálculo habitual de $x + y * w^4$ por culpa de la distribución no uniforme de los *tiles* en la VRAM [3].

```

1 uint32_t se_index(uint32_t y, uint32_t x)
2 {
3     // n = x + y * 32
4     uint32_t n = x + (y << 5);
5
6     if(x >= 32) {
7         n += 0x03E0;
8     }
9
10    if(y >= 32 && IS_BG_512x512) {
11        n += 0x0400;
12    }
13
14    return n;
15 }
```

Listado 3.1: Fragmento de código de la librería TONC.

Algoritmo 1: Procesa el incremento en el eje X teniendo en cuenta el entorno.

Input: y, x, incremento

Output: incremento

```

// Si se mueve a la derecha, se añade un offset
if incremento > 0 then
    x ← x + offset
// Cada tile ocupa 8 píxeles
incTiles ← abs(incremento)/8
// Se recorre cada tile recorrido con el incremento
for i ← 0 to incTiles do
    // Se recorre el espacio que ocupa la altura del personaje
    for j ← 0 to alturaPersonaje do
        // Se suma i en caso de ir a la derecha
        // y se resta en caso de ir a la izquierda
        tile ← getTileMapa(y + j, x±i)
        if tile es prohibido then
            incremento ← maxIncPermitido(x, i)
            return incremento
    
```

return incremento

⁴Los fondos que utilizan la matriz de transformación si siguen la ecuación convencional, pero para el proyecto se utilizarán fondos “normales”, en los cuales hay que seguir la metodología mostrada.

Algoritmo 2: Procesa el incremento en el eje Y teniendo en cuenta el entorno.

Input: y , x , incremento
Output: incremento

```

// Si se mueve hacia abajo, se añade un offset
if incremento > 0 then
     $y \leftarrow y + offset$ 

// Cada tile ocupa 8 píxeles
incTiles  $\leftarrow abs(incremento)/8$ 
// Se recorre cada tile recorrido con el incremento
for  $i \leftarrow 0$  to incTiles do
    // Se recorre el espacio que ocupa la anchura del personaje
    for  $j \leftarrow 0$  to anchuraPersonaje do
        // Se suma i en caso de ir abajo
        // y se resta en caso de ir arriba
        tile  $\leftarrow getTileMapa(y \pm i, x + j)$ 
        if tile es prohibido then
            incremento  $\leftarrow maxIncPermitido(y, i)$ 
            return incremento

return incremento

```

En el caso de los enemigos, la dirección en la que se mueven se ve afectada por las colisiones que detecten. En caso de colisionar, el enemigo se invertirá y moverá en la dirección contraria. Por razones de optimización se omiten:

- La comprobación del eje Y dado que los enemigos no pueden saltar. Para comprobar si los enemigos se acercan a una posición en caída libre, se comprobará el *tile* posicionado debajo. De ahí el *alturaEnemigo* + 1.
- El cálculo del mayor incremento permitido dado que no afectará de forma considerable la jugabilidad. Esta es la razón por la que se retorna 0 dentro del bucle.

A diferencia del movimiento del personaje, el movimiento de los enemigos se realiza de forma independiente. En cada iteración se llamará a un método que recorrerá todos los enemigos vivos para actualizar su posición utilizando el Algoritmo 3. Es necesario aclarar el uso del parámetro *a*, valor que será diferente dependiendo del enemigo. Si el movimiento del enemigo no permite el movimiento por “aire”, el parámetro *a* será mayor o igual a 1. En caso de que el movimiento del personaje permita desplazarse sin tener ningún bloque sólido debajo, el valor del parámetro *a* será 0.

Finalmente, la detección de colisiones entre el personaje principal y los enemigos se realiza tal y como se muestra en el Algoritmo 4.

Algoritmo 3: Procesa el incremento en el eje X del enemigo teniendo en cuenta el entorno.

Input: y, x, incremento

Output: incremento

```
// Si se mueve a la derecha, se añade un offset
if incremento > 0 then
    | x ← x + offset
    |
    // Cada tile ocupa 8 píxeles
    incTiles ← abs(incremento)/8
    |
    // Se recorre cada tile recorrido con el incremento
    for i ← 0 to incTiles do
        |
        // Se recorre el espacio que ocupa la altura del enemigo + a
        // a denota si el enemigo procesará los bloques que estén debajo
        for j ← 0 to alturaEnemigo + a do
            |
            // Se suma i en caso de ir a la derecha
            // y se resta en caso de ir a la izquierda
            tile ← getTileMapa(y + j, x±i)
            if tile es prohibido then
                | return 0
            |
        |
    |
return incremento
```

Algoritmo 4: Procesa las colisiones entre los enemigos y el personaje.

Input: numeroEnemigos

```
if personaje.estado esta siendo atacado then
    | return
    |
// Se recorren todos los enemigos vivos
for i ← 0 to numeroEnemigos do
    |
    // Si la distancia entre las dos entidades es menor
    // que la distancia minima declarada para el eje Y
    // continuar comprobando el eje X
    if abs(personaje.y - enemigos[i].y) < distColisionY then
        |
        if abs(personaje.x - enemigos[i].x) < distColisionX then
            |
            personaje.estado ← ATACADO
            |
            personaje.vida ← personaje.vida - 1
            |
            posicionEspada ← personaje.x ± offset
            if atacando & abs(posicionEspada - enemigos[i].x) < distColisionX
            |
            then
                | enemigos.vida ← enemigos.vida - 1
```

3.3 ASPECTOS DESTACADOS DEL DESARROLLO

En esta sección, se comentan los aspectos a tener en cuenta del desarrollo y algunas de las prácticas y conocimientos obtenidos en la tarea de “Estudio de las técnicas de programación” programada en la planificación del proyecto.

3.3.1 Pruebas de rendimiento

La GBA no es un sistema convencional dado que no tiene una forma de mostrar información fácilmente por pantalla. Es por esto, que para las pruebas de rendimiento realizadas se ha utilizado un sistema peculiar. Entre las formas que se han utilizado para medir el rendimiento se distinguen:

- Medida que incluye No\$GBA en su interfaz para comprobar rápidamente la carga de la CPU (véase la Figura 3.10). A pesar de ser un medio poco fiable, es de utilidad para hacerse una idea general del rendimiento, y para distinguir de inmediato rutinas con una amplia diferencia de rendimiento.

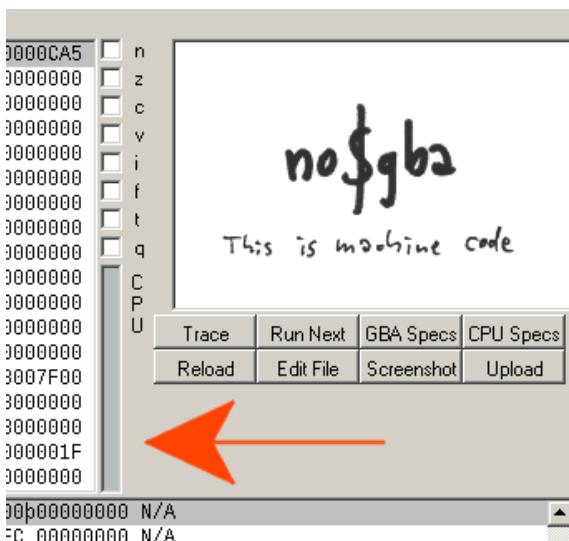


Figura 3.10: Interfaz que utiliza No\$GBA para mostrar el uso de la CPU.

- Uso de los temporizadores que incluye la consola para medir con más precisión. La visualización de los valores puede consultarse en el depurador o por pantalla. Dado que las pruebas se realizan también en hardware, se ha optado por mostrar los valores por pantalla dado que no se cuenta con el hardware necesario para mantener una conexión JTAG (véase la Figura 3.11).

3.3. ASPECTOS DESTACADOS DEL DESARROLLO



Figura 3.11: El valor del temporizador en la esquina superior izquierda.

Un ejemplo donde se comparó la eficiencia de dos fragmentos de código, fue en la carga de *assets* antes de iniciar una escena en el juego. Para la prueba se comparó el tiempo que transcurría al copiar los datos utilizando *DMA* y la función *memcpy*. El código utilizado es similar al mostrado en el Listado 3.2.

```

1 // Se activa el temporizador a f = 262.21 KHz
2 *((volatile uint16_t*)0x04000102) = 0x0081;
3
4 // Se copian los datos ...
5
6 // Se consigue el valor del temporizador
7 timer = *((volatile uint16_t*)0x04000100);
8
9 // Se desactiva el temporizador
10 *((volatile uint16_t*)0x04000102) = 0x0000;
11
12 // Se muestra el valor por pantalla ...
13
14
15

```

Listado 3.2: Tomando medidas mediante temporizadores.

Con los valores del *benchmark* obtenidos, se divide la frecuencia utilizada entre el temporizador para hallar los segundos que han transcurrido. En el caso del ejemplo anterior los resultados se muestran en la Figura 3.12. Con estos resultados se llegó a la conclusión de cargar los datos mediante *DMA* era la opción más adecuada.

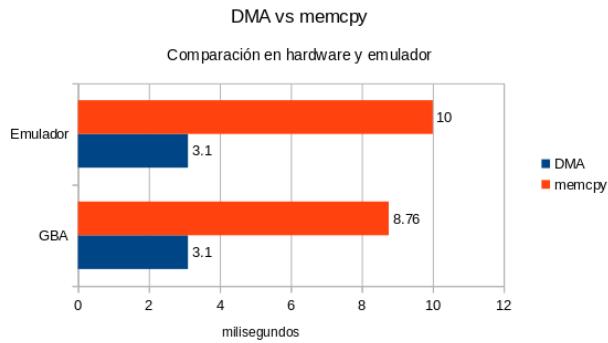


Figura 3.12: Resultados de la prueba de rendimiento.

3.3.2 Sincronización vertical

Uno de los primeros problemas que surgieron en el desarrollo del juego, fue la desincronización vertical que se producía al actualizar los fondos y *sprites* por pantalla. Este fenómeno se produce cuando se actualizan objetos mientras la pantalla se está actualizando. Una instancia donde se puede observar este problema se muestra en la Figura 3.13, en concreto en la parte superior de la imagen. Aquí, se puede observar la distorsión que se produce en el personaje y la moneda.



Figura 3.13: Desincronización vertical. Imagen obtenida de [22].

Inicialmente para resolver el problema, el programa se esperaba a que la consola terminase de renderizar el fotograma actual para cambiar las posiciones de los fondos y objetos. Para ello se comprueba el valor de VCount en la dirección de memoria 0x04000006 mediante dos bucles *while* mostrados en el Listado 3.3 [3]. La finalidad del primer bucle es la de evitar actualizar el fotograma más de una vez cuando el procesamiento realizado se completa antes de renderizarse el siguiente fotograma.

```

1 volatile uint16_t *vcount = (volatile uint16_t *)0x04000006;
2
3 while (*vcount >= 160);
4 while (*vcount < 160);
5
6 // Actualizar posiciones y estados ...

```

Listado 3.3: Implementación de VSync mediante dos bucles.

3.3. ASPECTOS DESTACADOS DEL DESARROLLO

Sin embargo, esta implementación no es eficiente y no cumple con los requisitos no funcionales RNF-1 y RNF-2, dado que el procesador se encuentra al 100 % mientras espera para procesar el siguiente fotograma. La solución por lo tanto, es utilizar las interrupciones del sistema, evitando así un consumo excesivo del procesador y de la batería. Esto se consigue, con la instrucción de ensamblador *swi*, la cual en la Game Boy Advance llamará a una de las funciones predefinidas de la BIOS. En este caso, la función que interesa utilizar es la 0x05, la función que pone al dispositivo en un estado de baja energía hasta que se produzca una interrupción VBlank. Para más información sobre las funciones disponibles consultar el Apéndice A.6.

Al abrir uno de los prototipos en el emulador No\$GBA, el cual ofrece una gráfica mostrando el uso del procesador, se puede observar una importante diferencia en consumo. En la Figura 3.14 se observa a la izquierda, el procesador al 100 % sin que ninguno de los personajes se mueva, mientras que a la derecha se observa un uso de la CPU casi inexistente al utilizar interrupciones.

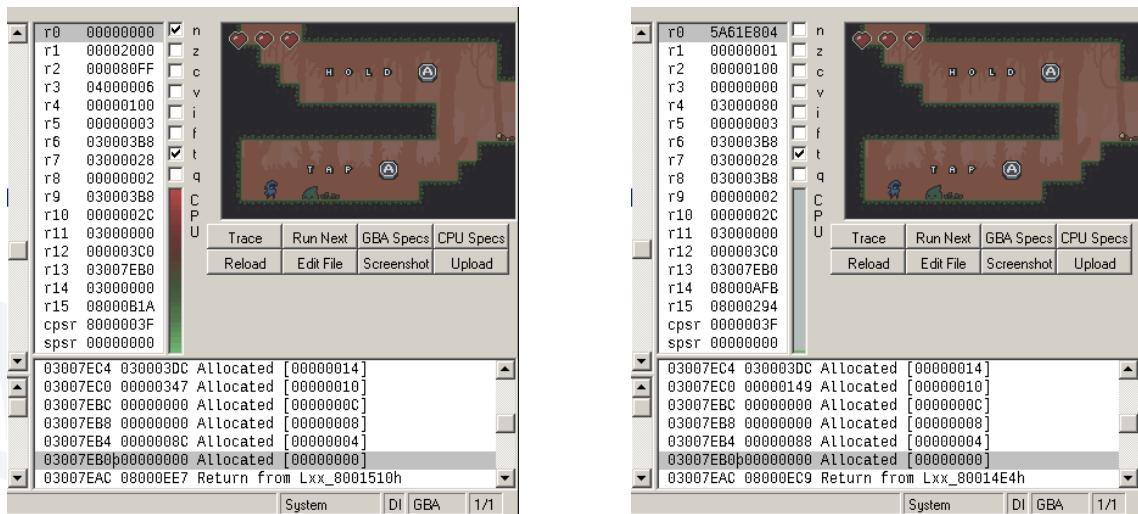


Figura 3.14: Rendimiento de las dos alternativas disponibles.

3.3.3 Operaciones de punto flotante

Una limitación importante que se tuvo en cuenta en el desarrollo fue el rendimiento de la consola al operar números de coma flotante, y dividir y realizar operaciones de módulo. Esto se debe a la falta de una *FPU* (Unidad de Punto Flotante) y una implementación por hardware de la división en la consola [3]. Para remediar estas limitaciones, se procura desplazar bits antes que realizar una división y utilizar números de coma fija. Es necesario mencionar que los compiladores optimizan automáticamente las divisiones en base 2, pero por razones de consistencia, en el código se utilizarán operaciones *shift* incluso para las divisiones en base 2.

Una demostración de una operación con números representados con coma fija para la sentencia $\frac{1}{2} + \frac{1}{2}$, se observa en el Listado 3.4.

```

1 int a, b, c;
2 a = b = 1 << 8;
3
4 a = a >> 1;
5 b = b >> 1;
6
7 c = a + b;
8 c = c >> 8; // c = 1

```

Listado 3.4: Ejemplo de un cálculo con números de coma fija.

A pesar de tener que realizar un preprocesamiento y postprocesamiento de los datos, sigue siendo más eficiente que realizar una división de coma flotante en un dispositivo sin *FPU*.

3.3.4 Limitaciones de lectura y escritura

Otra limitación del hardware afecta las operaciones de escritura y lectura sobre ciertas direcciones de memoria. La más llamativa es la limitación de escritura de la VRAM, RAM de paleta y OAM. En estas tres secciones el programador no puede realizar escrituras de 1 byte. En el caso de tener que escribir 1 byte, se tiene que leer 2 bytes, sobrescribir el byte que se desea modificar y actualizar el valor.

Otro ejemplo, esta vez de lectura, se encuentra en la mayoría de registros I/O. Los espacios de memoria dedicados a modificar los *offsets* de los fondos, se puede escribir pero no se puede leer los valores almacenados. Esta es la razón por la que se tienen que utilizar variables auxiliares al trabajar con ciertos registros de entrada y salida.

3.3.5 Texto

Las dos opciones que se barajaron para mostrar texto por pantalla en el juego fueron una implementación por *sprites* y una implementación por mapa. La primera consiste en reservar un espacio de la OAM y representar cada letra mediante un *sprite* diferente. La segunda implementación, consiste en incluir el texto en los fondos, incluyendo cada una de las letras en los *tiles*.

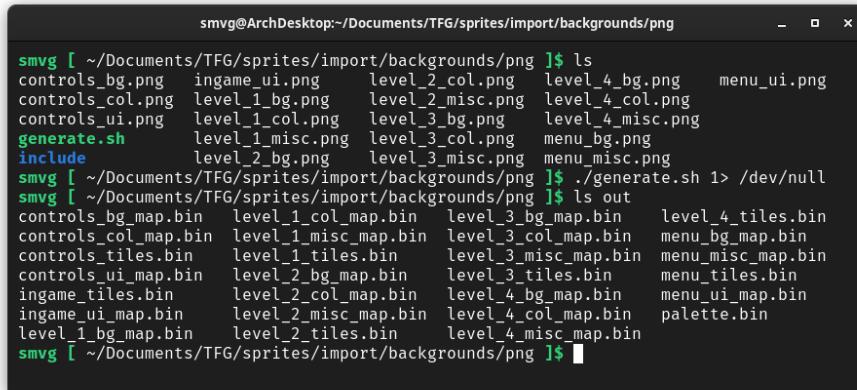
La implementación por la optó el autor fue una híbrida. Para los créditos iniciales, menú principal, menú de controles y menú in-game se utilizó una implementación mediante *sprites*. Esto permitía al programador modificar fácilmente el texto mostrado por pantalla. Por otro lado, para los niveles, en concreto las indicaciones mostradas en los niveles, se utilizó una implementación por mapa. De esta forma el programador no se tenía que preocupar de cambiar la posición del texto cuando el jugador se moviese por el nivel.

3.3.6 Preparación de los archivos binarios

Dada la gran cantidad de imágenes y archivos de audio utilizados, el autor utilizó Grit, SuperFamicom y Audacity para convertir los archivos originales a un array de bytes con el que el programa pueda trabajar. Para las imágenes, se programó un *script* de BASH que automatizase el proceso, creando un mapa para cada escena, un conjunto de *tiles* para cada grupo de escenas y una paleta de colores a compartir entre todos los fondos. Un ejemplo

3.3. ASPECTOS DESTACADOS DEL DESARROLLO

de la salida generada se puede observar en la Figura 3.15. Los comandos ejecutados en el *script* se pueden consultar en el Apéndice A.5.



```

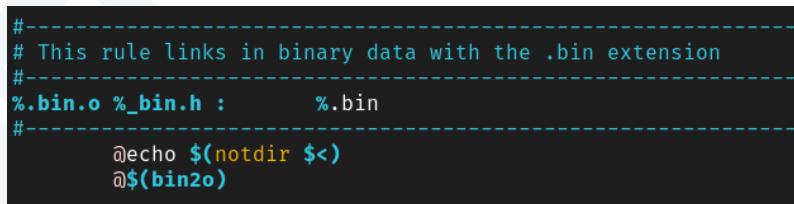
smvg [ ~/Documents/TFG.sprites/import/backgrounds/png ]$ ls
controls_bg.png    ingame_ui.png    level_2_col.png    level_4_bg.png    menu_ui.png
controls_col.png   level_1_bg.png    level_2_misc.png   level_4_col.png
controls_ui.png    level_1_col.png   level_3_bg.png    level_4_misc.png
generate.sh        level_1_misc.png  level_3_col.png   menu_bg.png
include            level_2_bg.png    level_3_misc.png  menu_misc.png
smvg [ ~/Documents/TFG.sprites/import/backgrounds/png ]$ ./generate.sh > /dev/null
smvg [ ~/Documents/TFG.sprites/import/backgrounds/png ]$ ls out
controls_bg_map.bin level_1_col_map.bin level_3_bg_map.bin level_4_tiles.bin
controls_col_map.bin level_1_misc_map.bin level_3_col_map.bin menu_bg_map.bin
controls_tiles.bin  level_1_tiles.bin   level_3_misc_map.bin menu_misc_map.bin
controls_ui_map.bin level_2_bg_map.bin  level_3_tiles.bin  menu_tiles.bin
ingame_tiles.bin   level_2_col_map.bin level_4_bg_map.bin menu_ui_map.bin
ingame_ui_map.bin  level_2_misc_map.bin level_4_col_map.bin palette.bin
level_1_bg_map.bin level_2_tiles.bin   level_4_misc_map.bin
smvg [ ~/Documents/TFG.sprites/import/backgrounds/png ]$ 

```

Figura 3.15: Archivos generados por el *script*.

En el caso de los archivos de audio, se utilizó Audacity para convertir todas las pistas de audio a un solo canal y a una frecuencia de 16 KHz. El resultado se exportaría después a un formato “raw” sin ningún tipo de cabecera [22].

Con los archivos “.bin” generados, el siguiente paso es utilizarlos en el código, para ello el Makefile utilizado (y proporcionado por el *toolchain*) para el proyecto incluye una regla para enlazar todos los archivos binarios al proyecto (véase la Figura 3.16). Esta regla permitirá al programador referenciar los archivos incluyendo en la cabecera un archivo con el siguiente formato: “archivo_bin.h” siendo archivo el nombre del archivo utilizado sin contar la extensión. La variable que guarda el valor del archivo y la variable que guarda el tamaño en bytes sigue un formato similar al de la cabecera: “archivo_bin” y “archivo_bin_size” respectivamente.



```

#-----
# This rule links in binary data with the .bin extension
#-----
%.bin.o %.bin.h :      %.bin
#-
@echo $(notdir $<
@$(bin2o)

```

Figura 3.16: Regla del Makefile.

4 RESULTADO FINAL

En esta sección se muestra el producto desarrollado, explicando en que consiste el juego y las decisiones con respecto al diseño que se han tomado. Todo el código del proyecto se puede encontrar en [23], repositorio alojado en GitHub. La estructura del proyecto se puede consultar en el Apéndice A.1. Además, en el repositorio se proporciona una versión del juego ya preparada para su prueba en emuladores o en hardware. Este archivo, junto al código de esa versión, se puede encontrar en la sección de “Releases” del repositorio. Un ejemplo de una versión publicada del juego se puede observar en la Figura 4.1.

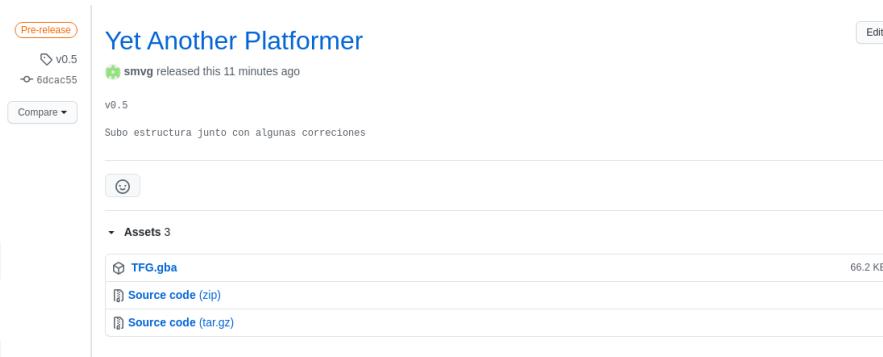


Figura 4.1: Versión publicada del juego.

4.1 YAP - Yet Another Platformer

El juego, titulado *YAP* (*Yet Another Platformer*, logo en la Figura 4.2), pone al jugador en el lugar de un mago que tiene que progresar por cada nivel, derrotando a los enemigos que encuentre por el camino. Los controles utilizados para moverse por el mapa son los habituales

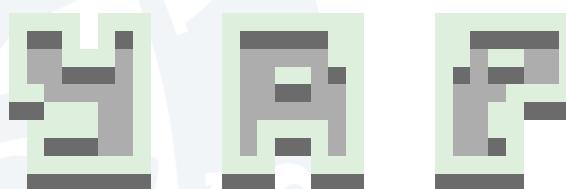


Figura 4.2: Logo del juego *Yet Another Platformer*.

para juegos de este género, el botón izquierdo para moverse a la izquierda y el botón derecho para moverse a la derecha.

El mago que controla el jugador deberá alternar entre dos formas distintas, la azul y dorada, para poder progresar en el juego. Esto se debe a que cada una de las variantes tiene habilidades imprescindibles para completar cada uno de los niveles:

- Variante azul: La variante azul, la cual se activa pulsando el botón “L”, permite al jugador moverse con más rapidez con tal de poder escapar con facilidad de los enemigos. Además permite al jugador saltar a las estructuras elevadas que encontrará en cada uno de los niveles presionando el botón “A”. La limitación de esta variante aparece al enfrentarse con los enemigos ya que no puede atacar, para ello el jugador tendrá que transformarse en la variante dorada del mago.
- Variante dorada: La variante dorada, la cual se activa pulsando el botón “R”, limita el movimiento del jugador, ralentizando su velocidad en el eje X e imposibilitando su movimiento en el eje Y. Sin embargo, ahora el jugador tiene la posibilidad de atacar a los enemigos que se encuentren en proximidad presionando el botón “B”.

4.2 PERSONAJES

Los personajes del juego, excepto por algunas modificaciones menores, son todos obra de *o_lobster*. La primera versión del juego cuenta con el mago mostrado en el *spritesheet* de la Figura 4.3. En la primera fila se observan las imágenes utilizadas para cuando el personaje no se mueve. La segunda fila muestra las imágenes utilizadas para cuando el personaje se mueve por tierra. En la tercera y cuarta fila se observan las imágenes para cuando el personaje cae de una posición elevada y salta. En la quinta y sexta fila se observan las imágenes utilizadas para animar el ataque del personaje, siendo la quinta fila la correspondiente al personaje y la sexta la correspondiente a la espada que saca el mismo. Por último, la última fila, muestra las imágenes utilizadas para la animación de la muerte del personaje. Para separar la animación de muerte de las demás, se ha optado también por parar de actualizar todas las demás animaciones cuando se de el caso (dejando al personaje como la única entidad animada). Cada una de los estados que puede tener el personaje (inactivo, corriendo, saltando y atacando), se actualiza la animación cada 10 fotogramas. Este comportamiento también se observa para los enemigos. En caso de agotar las imágenes destinadas a un estado en particular, se vuelve a empezar por la primera imagen. Un estado que no se incluye en el *spritesheet* es el estado cuando el personaje es atacado. Tomando como inspiración algunos juegos de la época, se ha optado por hacer que el *sprite* parpadeé para señalar que ha recibido daño de un enemigo para así rebajar el número de *tiles* utilizados. Este efecto se consigue activando y desactivando el *sprite* cada cierto periodo de tiempo.

4.2. PERSONAJES

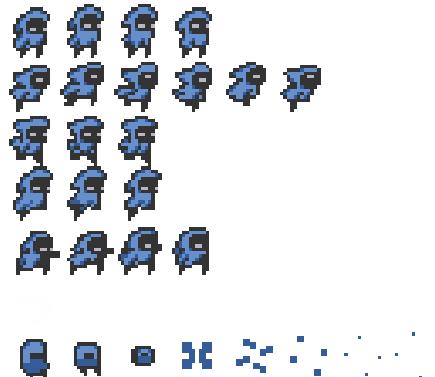


Figura 4.3: Los *sprites* del personaje principal.

La variante dorada del personaje utiliza exactamente el mismo *spritesheet* de la Figura 4.3 pero especificando una paleta de colores alterna. La transición entre cada uno de los estados hace uso del efecto mosaico para así dar la sensación que el mago se está transformando. La Figura 4.4 muestra a los dos personajes.



(a) Variante azul saltando.

(b) Variante dorada atacando.

Figura 4.4: Las dos variantes del personaje principal.

Los dos enemigos incluidos en el juego cuentan con animaciones y estados más simples que el personaje principal. El primer enemigo, el “monstruo verde” cuenta con dos estados, el estado en movimiento o inactivo, y el estado de muerte del enemigo. El *spritesheet* se puede observar en la Figura 4.5. Este enemigo se mueve de forma independiente hasta detectar una colisión en el eje X o vacío en el eje Y (una caída). Cuando detecta la colisión invertirá su posición.



Figura 4.5: Los *sprites* del primer enemigo.

El otro enemigo, la “mosca”, al igual que el “monstruo verde”, cuenta con dos estados, el estado en movimiento o inactivo, y el estado de muerte del mismo. El *spritesheet* se puede observar en la Figura 4.6. A diferencia del “monstruo verde”, este enemigo puede moverse sin tener ningún bloque sólido por debajo, aunque si se encuentra de frente con un bloque invertirá su orientación.



Figura 4.6: Los *sprites* del segundo enemigo.

Los dos enemigos renderizados en un nivel se pueden observar en la Figura 4.7. Para una demostración más extensiva de los diferentes estados de cada personaje consultar el Apéndice A.2.

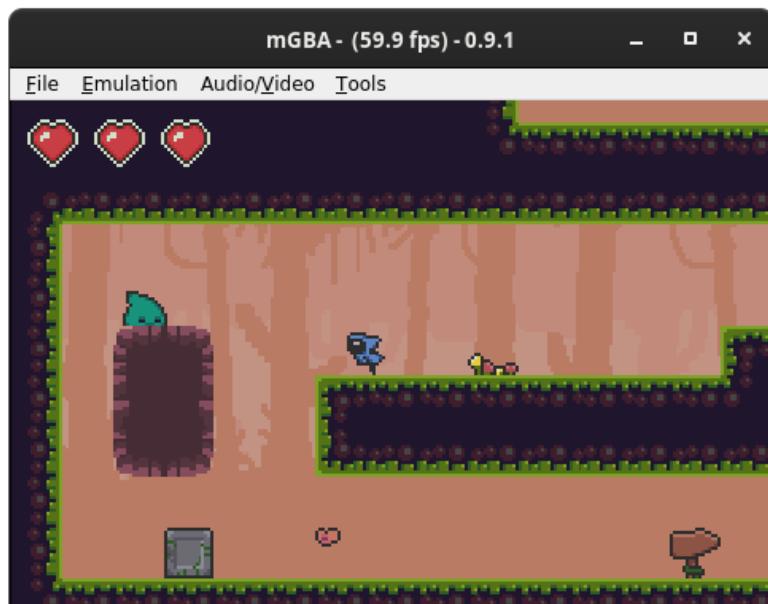


Figura 4.7: Enemigos de uno de los niveles.

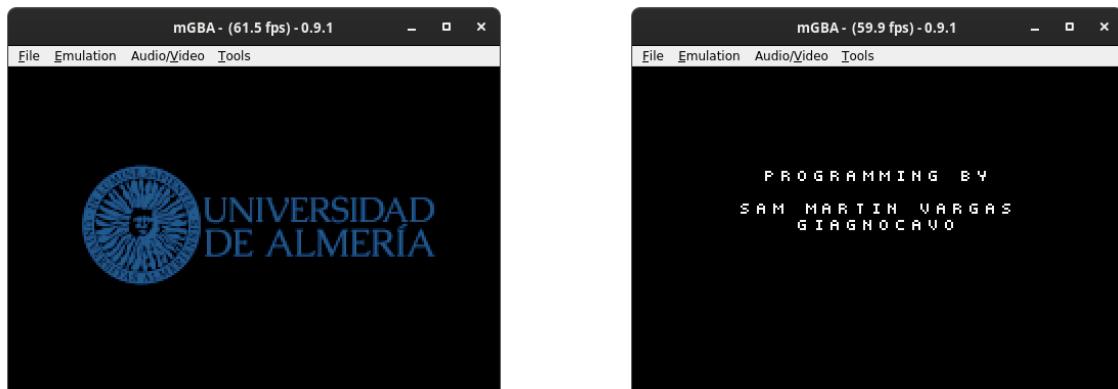
4.3 *Title screens, MENÚS Y NIVELES*

Cada uno de los créditos, menús y niveles reciben el nombre de “escenas” en el código, término que se utiliza en motores gráficos como Unity. Los créditos iniciales del juego muestran el logo de la Universidad de Almería (aprovechando los fondos *bitmap* del dispositivo) y todas las personas cuyos recursos han sido utilizados. Dos ejemplos, incluyendo el logo y el nombre del autor se pueden observar en la Figura 4.8. Para el resto de escenas consultar el Apéndice A.4.

La siguiente escena con la que el jugador se encuentra, una vez acabados los créditos iniciales del juego, es el menú principal. El menú principal cuenta con fondos y *sprites* animados

4.3. TITLE SCREENS, MENÚS Y NIVELES

y una imagen proveniente del emulador se puede observar en la Figura 4.9. Las diferencias observadas entre el menú actual y los mostrados previamente se deben a cambios durante el desarrollo del juego. El menú definitivo es el mostrado en la Figura 4.9.



(a) *Title screen* con el logo de la UAL.

(b) Game Boy Advance Micro

Figura 4.8: *Title screen*.

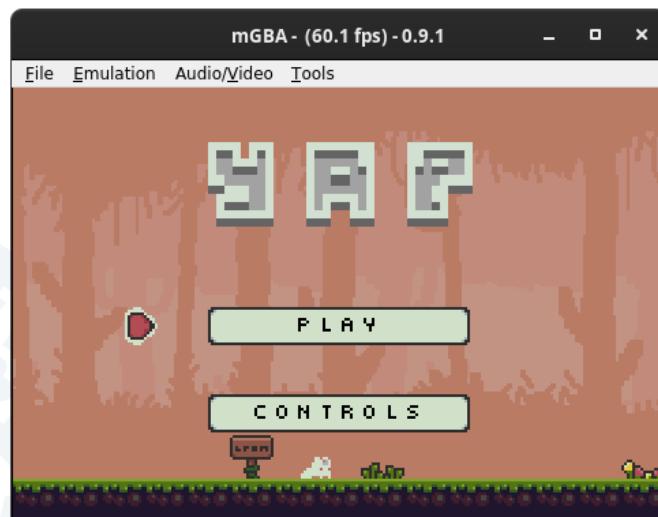


Figura 4.9: Menú principal del juego.

Aquí el jugador puede empezar una partida, o seguir el progreso en caso de haber jugado anteriormente, y consultar los controles del juego. En la escena de controles, se incluye un espacio para que el jugador pueda probarlos tal y como se puede observar en la Figura 4.10.



Figura 4.10: Escena de controles del juego.

El primer nivel del juego cuenta con pequeñas indicaciones para guiar al jugador durante su primera partida. Las dos escenas que conforman el primer nivel se pueden apreciar en la Figuras 4.11 y 4.12. A pesar de no forzar al jugador a realizar las indicaciones mostradas, el diseño del primer nivel requiere que el jugador sepa realizar todas las acciones y funcionalidades para poder completarlo. El resto de niveles se puede observar en el Apéndice A.4.



Figura 4.11: Primera parte del nivel.



Figura 4.12: Segunda parte del nivel.

Todos los niveles del juego utilizan los 4 fondos “normales” que ofrece la GBA. El primero de ellos se reserva para cualquier estructura que provoque una colisión con el personaje y enemigos, el suelo por ejemplo. El segundo fondo se utiliza para aquellos objetos que forman parte del nivel pero no provocan colisiones. El tercer fondo, se reserva para el fondo naranja utilizado en la Figura 4.9. Finalmente, el cuarto fondo se utiliza para guardar el menú in-game, el cual solo se activa si el jugador presiona “Start”.

4.4 MÚSICA

Dado que no se puede demostrar las dos piezas de música utilizadas para el juego, realizadas por *Goose Ninja*, se comentará brevemente la implementación por *Direct Sound*.

En el juego, se hacen uso de dos muestras de audio, diseñadas para poder reproducirse en ciclo sin cortes. Para hacer posible su reproducción en el dispositivo, se han convertido a una frecuencia de 16 KHz a 8 bits. Cada ciclo se realiza a partir de los contadores del dispositivo y de las transferencias DMA de tipo 1. Las transferencias además se configuran para que ocurran cada vez que el buffer de audio se quede vacío.

4.5 INDICADORES IN-GAME

El único indicador utilizado es el número de corazones (vida) con el que cuenta el personaje. El indicador se localiza en la esquina superior izquierda, y al aparecer el personaje realiza una pequeña animación para llamar la atención del jugador. A diferencia del resto animaciones, esta depende de la matriz de transformación ya que cambia la escala de un tamaño inapreciable al tamaño original en un corto periodo de tiempo. En la Figura 4.13 se pueden observar el estado inicial y final de la animación. Dado que los corazones denotan la vida del jugador, estos irán desapareciendo conforme el jugador pierde vida.



(a) Inicio de la animación.

(b) Fin de la animación.

Figura 4.13: Tres corazones indicando la vida del jugador.

4.6 EVALUACIÓN DEL JUEGO EN EL HARDWARE ORIGINAL

En esta sección se mostrará el procedimiento a seguir para poder probar el juego en el hardware original. Para ello se hará uso de la tarjeta *Supercard* adquirida junto con el software necesario para cargar el juego en la tarjeta. En el repositorio se incluirá una copia del software dado que no se puede asegurar su disponibilidad en los servidores utilizados por el fabricante para alojarlo.

Con el código compilado, ejecutando el comando “make” sobre el directorio raíz del proyecto, se ejecuta el programa “Super Card” (véase la Figura 4.14). Una vez abierto, en la ventana de opciones se seleccionará la tarjeta microSD que se introducirá en el cartucho. Seguidamente, en la ventana principal se seleccionará el juego para procesarlo y guardarla en la tarjeta.

Seguidamente, se desconecta la tarjeta microSD y se introduce en el cartucho “Supercard”. Al iniciar la consola, se mostrará un menú con la lista de todos los juegos instalados en la tarjeta. Para abrir el juego, basta con seleccionarlo con el pad direccional y el botón “A”. En la Figura 4.15 se puede observar el juego ejecutándose en hardware de forma nativa.

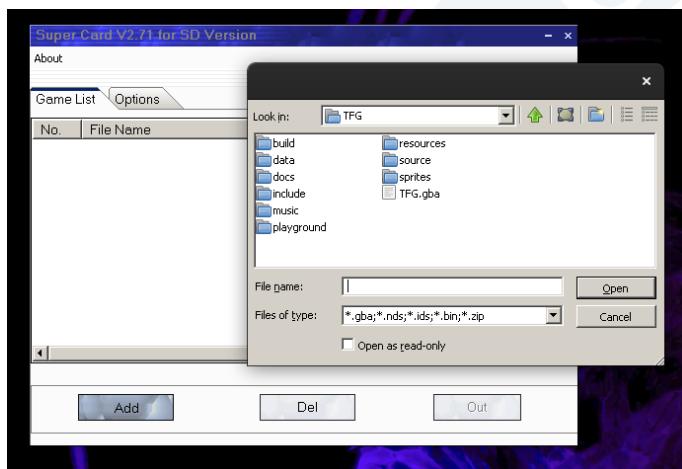


Figura 4.14: Software para meter juegos en el cartucho.



Figura 4.15: El juego se ejecuta en la Nintendo DS.

5 CONCLUSIONES DEL PROYECTO

En esta sección se realizará un balance del trabajo realizado indicando a su vez varias mejoras que pueden realizarse en las próximas versiones del producto.

El autor considera el proyecto uno de los más interesantes que ha realizado en la carrera, permitiéndole aplicar varios de los conocimientos adquiridos durante la misma. Los conocimientos aplicados varían enormemente, cubriendo conceptos como *DMA* proporcionados en la asignatura de “Periféricos e Interfaces”, el uso de ondas visto en la asignatura de “Transmisión de Datos y Redes de Computadores” e incluso algunos conceptos matemáticos en el área de matrices. El desarrollo de un juego para la Game Boy Advance también ha permitido un mejor entendimiento de varios de los dispositivos con los que el autor ha trabajado anteriormente, en concreto los microcontroladores Arduino y ESPRESSIF mencionados en la introducción del trabajo. Esto último demuestra la efectividad que tiene el dispositivo en el espacio educativo, tal y como se demostró en [4, 5, 6].

Además, el autor considera que actualmente tiene un mejor entendimiento de las tareas a realizar para proyectos futuros que involucren el desarrollo de un videojuego, identificando mejor las tareas que puede realizar y las que no. De las tareas que no se han podido realizar, como el diseño del arte y de la música, el autor considera que con más tiempo este aspecto podría haber sido mejorado dado que acabó siendo un factor limitante durante el desarrollo del juego. Otro aspecto a mejorar, relacionado con el arte del juego, es la poca cantidad de contenido disponible en el producto final ya que los enemigos, personajes y fondos dependían del contenido “open source” disponible online.

A pesar de que el proyecto no ha quedado con los estándares que el autor deseaba, tiene la motivación no solo para seguir desarrollando el proyecto hasta conseguir una versión con la que esté satisfecho, sino para continuar desarrollando juegos para la Game Boy Advance. En sus próximos proyectos, espera combinar los conocimientos ya adquiridos con otros con finalidades más específicas como la comunicación serial que ofrece la consola para comunicarse con otros jugadores.

5.1 TRABAJO FUTURO

A continuación se incluirán las mejoras que el autor considera implementar durante los próximos meses para alcanzar el nivel de calidad deseado. Las mejoras no incluyen las “remodelaciones” desde cero que desearía realizar el autor, sino mejoras inmediatas al producto actual.

- Mejora del movimiento del jugador, haciendo que los controles sean más fluidos para el jugador.
- Mejora en las colisiones del jugador con el mapa. Actualmente las colisiones del jugador con el entorno se realizan de forma aproximada cuando este se mueve a una velocidad no exacta (por ejemplo, al moverse a 1.25 píxeles por unidad de tiempo).
- Buscar una implementación de la sincronización vertical que permita jugar al juego a 120 Hz en emuladores. El autor no ha investigado al respecto pero cree que es posible y una opción interesante dado que una gran parte de los jugadores utiliza emuladores en dispositivos considerablemente más potentes que la consola original.
- Uso de efectos de sonido para las acciones del personaje principal (por ejemplo, al saltar y atacar). Estos se realizarían con los canales de audio 1, 2, 3 y 4.
- Implementación de un sistema de texto que permita el uso de más de un lenguaje. Actualmente el juego solo se encuentra en inglés.
- Implementación de un sistema que permita el uso de más de un mago. Esto permitiría añadir más variantes al juego (actualmente solo hay 2). El prototipo que el autor tenía en mente, sería uno similar al ofrecido por otros juegos con varias opciones a seleccionar. El jugador presionaría el botón “L” o “R”, acción que abriría un menú, y con el pad direccional seleccionaría la variante a utilizar. Una imagen mostrando el prototipo se puede observar en la Figura 5.1.

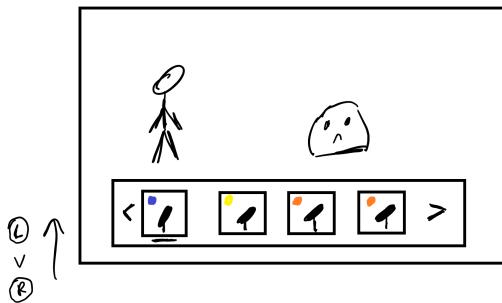


Figura 5.1: Prototipo del nuevo sistema de selección.

- Implementación de las secciones más costosas en ensamblador. El autor tiene pensado implementar las funciones encargadas de gestionar las colisiones de los enemigos y personaje principal.

BIBLIOGRAFÍA

- [1] R. Copetti, "Game Boy Advance architecture: A practical analysis." <https://www.copetti.org/writings/consoles/game-boy-advance/>, Mayo 2021. Último acceso: 2021-07-10.
 - [2] Nintendo®, *AGB Programming Manual*, 1.1 ed., 2001.
 - [3] J. Vijn, "Tonic v1.4.2." <https://www.coranac.com/tonic/text/>. Último acceso: 2021-07-10.
 - [4] B. B. Camper, *Homebrew and the social construction of gaming: community, creativity, and legal context of amateur Game Boy Advance development*. PhD thesis, Massachusetts Institute of Technology, 2005.
 - [5] N. C. Cruz, J. L. Redondo, J. D. Álvarez, y P. M. Ortigosa, "Aplicaciones de chip-8, una máquina virtual de finales de los 70, en los estudios actuales de ingeniería informática,"
 - [6] I. Finlayson, "Using the Game Boy Advance to teach Computer Systems and Architecture," *Journal of Computing Sciences in Colleges*, vol. 32, no. 3, pp. 78–84, 2017.
 - [7] G. Kacmarcik y S. G. Kacmarcik, "Introducing computer programming via Game Boy Advance homebrew," in *Proceedings of the 40th ACM technical symposium on Computer science education*, pp. 281–285, 2009.
 - [8] D. A. Patterson y J. L. Hennessy, *Computer Organization and Design: The Hardware Software Interface ARM Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2016.
 - [9] P. Foltýnek, M. Babiuch, y P. Šuránek, "Measurement and data processing from internet of things modules by dual-core application using esp32 board," *Measurement and Control*, vol. 52, no. 7-8, pp. 970–984, 2019.
 - [10] J. Koutonen y M. Leppänen, "How are agile methods and practices deployed in video game development? a survey into finnish game studios," in *International Conference on Agile Software Development*, pp. 135–149, Springer, 2013.
 - [11] P. Tan, R. Eberhardt, S. Verrilli, y A. Grant, "Cms.611j creating video games," 2014.
 - [12] M. Korth, "Gbatek: Game Boy Advance/Nintendo DS technical info." <https://problemkaputt.de/gbatek.htm>, 2007. Último acceso: 2021-07-10.
 - [13] J. L. Hennessy, D. A. Patterson, y K. Asanović, *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann/Elsevier, 5th ed ed.
-

- [14] Advanced RISC Machines Ltd., *ARM7TDMI Technical reference manual*, r4p1 ed., 2004.
- [15] R. Meier, "Ce 1921 computer architecture," 2020.
- [16] "The audio advance - gameboy advance sound resources." <http://beologic.com/gba/index.php>. Último acceso: 2021-07-10.
- [17] A. Rubini y J. Corbet, *Linux device drivers*. "O'Reilly Media, Inc.", 2005.
- [18] D. Fussell, "CS384G Computer Graphics," 2011.
- [19] J. A. L. Ramos y J. C. Díaz, "Álgebra lineal y Matemáticas discretas," 2021.
- [20] N. P. Soriano y J. C. Rodríguez, "Desarrollo de interfaces de usuario," 2020.
- [21] A. Cuesta, "Ingeniería de videojuegos," 2021.
- [22] I. Finlayson, "CPSC 305 Computer Systems and Architecture," 2021.
- [23] S. M. V. Giagnocavo, "Repositorio del código desarrollado." <https://github.com/smvg/YAP>, 2021. Último acceso: 2021-07-10.

A CONTENIDO ADICIONAL

A.1 ESTRUCTURA DEL PROYECTO

La estructura escogida para el proyecto ha sido la marcada por el “Makefile” proporcionado por el *toolchain* utilizado. La estructura es la observada en la Figura A.1.

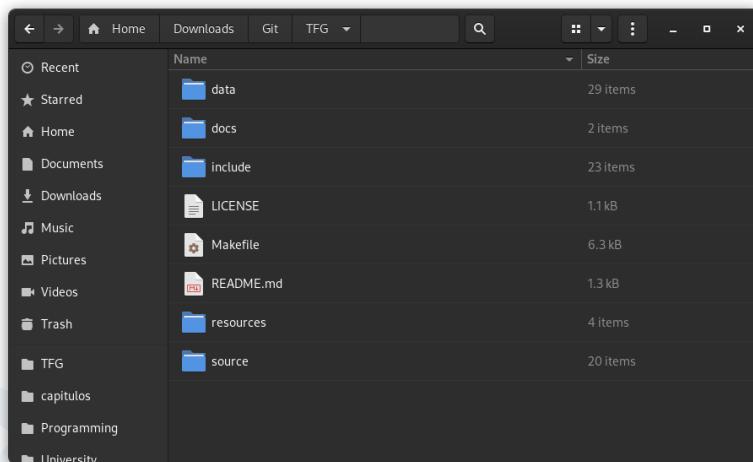


Figura A.1: Estructura del proyecto.

Cada uno de los archivos y directorios mostrados tiene la siguiente función:

- data: En esta carpeta se localizan todos los archivos binarios, como imágenes y pistas de audio, que posteriormente son convertidas a un formato compatible con la consola.
- docs: Aquí reside el documento \LaTeX junto con otros apuntes en suelo realizados durante el desarrollo del juego.
- include: Todas las cabeceras del código residen aquí.
- LICENSE: En este archivo se describen las condiciones de la licencia escogida, la MIT¹.
- Makefile: El archivo que permite a “make” compilar el código y convertir los assets del juego. Precisa una instalación previa de *devkitpro* dado que utiliza varias herramientas incluidas en el *toolchain*.

¹<https://opensource.org/licenses/MIT>

- README.md: En este archivo se incluyen varias instrucciones a los usuarios que visitan el repositorio GitHub.
- resources: En esta carpeta se incluyen varios recursos y documentación. Puede servir de utilidad para en caso de perder acceso a la red.
- source: Aquí reside todo el código del juego.

A.2 PERSONAJES



Figura A.2: Variante azul corriendo.



Figura A.3: Variante azul iniciando la transformación.



Figura A.4: Últimos fotogramas de la transformación de la variante azul a la dorada.



Figura A.5: Variante dorada.



Figura A.6: Ataque del personaje principal. además se observa la muerte del “monstruo verde”.



Figura A.7: Variante dorada del personaje principal junto a uno de los enemigos.



Figura A.8: Ataque del personaje principal. Además se observa la muerte de la “mosca”.



Figura A.9: El personaje principal saltando.



A.2. PERSONAJES



Figura A.10: El personaje principal cayendo.



Figura A.11: El personaje principal desapareciendo al no tener vida.

A.3 REGISTROS

LCD I/O Registers

40000000h	2	R/W	DISPCNT	LCD Control
4000002h	2	R/W	-	Undocumented - Green Swap
4000004h	2	R/W	DISPSTAT	General LCD Status (STAT,LYC)
4000006h	2	R	VCOUNT	Vertical Counter (LY)
4000008h	2	R/W	BG0CNT	BG0 Control
400000Ah	2	R/W	BG1CNT	BG1 Control
400000Ch	2	R/W	BG2CNT	BG2 Control
400000Eh	2	R/W	BG3CNT	BG3 Control
4000010h	2	W	BG0HOFS	BG0 X-Offset
4000012h	2	W	BG0VOFS	BG0 Y-Offset
4000014h	2	W	BG1HOFS	BG1 X-Offset
4000016h	2	W	BG1VOFS	BG1 Y-Offset
4000018h	2	W	BG2HOFS	BG2 X-Offset
400001Ah	2	W	BG2VOFS	BG2 Y-Offset
400001Ch	2	W	BG3HOFS	BG3 X-Offset
400001Eh	2	W	BG3VOFS	BG3 Y-Offset
4000020h	2	W	BG2PA	BG2 Rotation/Scaling Parameter A (dx)
4000022h	2	W	BG2PB	BG2 Rotation/Scaling Parameter B (dmx)
4000024h	2	W	BG2PC	BG2 Rotation/Scaling Parameter C (dy)
4000026h	2	W	BG2PD	BG2 Rotation/Scaling Parameter D (dmy)
4000028h	4	W	BG2X	BG2 Reference Point X-Coordinate
400002Ch	4	W	BG2Y	BG2 Reference Point Y-Coordinate
4000030h	2	W	BG3PA	BG3 Rotation/Scaling Parameter A (dx)
4000032h	2	W	BG3PB	BG3 Rotation/Scaling Parameter B (dmx)
4000034h	2	W	BG3PC	BG3 Rotation/Scaling Parameter C (dy)
4000036h	2	W	BG3PD	BG3 Rotation/Scaling Parameter D (dmy)
4000038h	4	W	BG3X	BG3 Reference Point X-Coordinate
400003Ch	4	W	BG3Y	BG3 Reference Point Y-Coordinate
4000040h	2	W	WIN0H	Window 0 Horizontal Dimensions
4000042h	2	W	WIN1H	Window 1 Horizontal Dimensions
4000044h	2	W	WIN0V	Window 0 Vertical Dimensions
4000046h	2	W	WIN1V	Window 1 Vertical Dimensions
4000048h	2	R/W	WININ	Inside of Window 0 and 1
400004Ah	2	R/W	WINOUT	Inside of OBJ Window & Outside of Windows
400004Ch	2	W	MOSAIC	Mosaic Size
400004Eh	-	-	-	Not used
4000050h	2	R/W	BLDCNT	Color Special Effects Selection
4000052h	2	R/W	BLDALPHA	Alpha Blending Coefficients
4000054h	2	W	BLDY	Brightness (Fade-In/Out) Coefficient
4000056h	-	-	-	Not used

Figura A.12: Registros LCD.

Sound Registers

4000060h	2	R/W	SOUND1CNT_L	Channel 1 Sweep register	(NR10)
4000062h	2	R/W	SOUND1CNT_H	Channel 1 Duty/Length/Envelope	(NR11, NR12)
4000064h	2	R/W	SOUND1CNT_X	Channel 1 Frequency/Control	(NR13, NR14)
4000066h	-	-		Not used	
4000068h	2	R/W	SOUND2CNT_L	Channel 2 Duty/Length/Envelope	(NR21, NR22)
400006Ah	-	-		Not used	
400006Ch	2	R/W	SOUND2CNT_H	Channel 2 Frequency/Control	(NR23, NR24)
400006Eh	-	-		Not used	
4000070h	2	R/W	SOUND3CNT_L	Channel 3 Stop/Wave RAM select	(NR30)
4000072h	2	R/W	SOUND3CNT_H	Channel 3 Length/Volume	(NR31, NR32)
4000074h	2	R/W	SOUND3CNT_X	Channel 3 Frequency/Control	(NR33, NR34)
4000076h	-	-		Not used	
4000078h	2	R/W	SOUND4CNT_L	Channel 4 Length/Envelope	(NR41, NR42)
400007Ah	-	-		Not used	
400007Ch	2	R/W	SOUND4CNT_H	Channel 4 Frequency/Control	(NR43, NR44)
400007Eh	-	-		Not used	
4000080h	2	R/W	SOUNDCNT_L	Control Stereo/Volume/Enable	(NR50, NR51)
4000082h	2	R/W	SOUNDCNT_H	Control Mixing/DMA Control	
4000084h	2	R/W	SOUNDCNT_X	Control Sound on/off	(NR52)
4000086h	-	-		Not used	
4000088h	2	BIOS	SOUNDBIAS	Sound PWM Control	
400008Ah	..	-		Not used	
4000090h	2x10h	R/W	WAVE_RAM	Channel 3 Wave Pattern RAM	(2 banks!!)
40000A0h	4	W	FIFO_A	Channel A FIFO, Data 0-3	
40000A4h	4	W	FIFO_B	Channel B FIFO, Data 0-3	
40000A8h	-	-		Not used	

Figura A.13: Registros de Sonido.

DMA Transfer Channels

40000B0h	4	W	DMA0SAD	DMA 0 Source Address
40000B4h	4	W	DMA0DAD	DMA 0 Destination Address
40000B8h	2	W	DMA0CNT_L	DMA 0 Word Count
40000BAh	2	R/W	DMA0CNT_H	DMA 0 Control
40000BCh	4	W	DMA1SAD	DMA 1 Source Address
40000C0h	4	W	DMA1DAD	DMA 1 Destination Address
40000C4h	2	W	DMA1CNT_L	DMA 1 Word Count
40000C6h	2	R/W	DMA1CNT_H	DMA 1 Control
40000C8h	4	W	DMA2SAD	DMA 2 Source Address
40000CCh	4	W	DMA2DAD	DMA 2 Destination Address
40000D0h	2	W	DMA2CNT_L	DMA 2 Word Count
40000D2h	2	R/W	DMA2CNT_H	DMA 2 Control
40000D4h	4	W	DMA3SAD	DMA 3 Source Address
40000D8h	4	W	DMA3DAD	DMA 3 Destination Address
40000DCh	2	W	DMA3CNT_L	DMA 3 Word Count
40000DEh	2	R/W	DMA3CNT_H	DMA 3 Control
40000E0h	-	-		Not used

Figura A.14: Registros Direct Memory Access.

Timer Registers

4000100h	2	R/W	TM0CNT_L	Timer 0 Counter/Reload
4000102h	2	R/W	TM0CNT_H	Timer 0 Control
4000104h	2	R/W	TM1CNT_L	Timer 1 Counter/Reload
4000106h	2	R/W	TM1CNT_H	Timer 1 Control
4000108h	2	R/W	TM2CNT_L	Timer 2 Counter/Reload
400010Ah	2	R/W	TM2CNT_H	Timer 2 Control
400010Ch	2	R/W	TM3CNT_L	Timer 3 Counter/Reload
400010Eh	2	R/W	TM3CNT_H	Timer 3 Control
4000110h	-	-		Not used

Figura A.15: Registros de Temporización.

Serial Communication (1)

4000120h	4	R/W	SIODATA32	SIO Data (Normal-32bit Mode; shared with below)
4000120h	2	R/W	SIOMULTI0	SIO Data 0 (Parent) (Multi-Player Mode)
4000122h	2	R/W	SIOMULTI1	SIO Data 1 (1st Child) (Multi-Player Mode)
4000124h	2	R/W	SIOMULTI2	SIO Data 2 (2nd Child) (Multi-Player Mode)
4000126h	2	R/W	SIOMULTI3	SIO Data 3 (3rd Child) (Multi-Player Mode)
4000128h	2	R/W	SIOCNT	SIO Control Register
400012Ah	2	R/W	SIOMLT_SEND	SIO Data (Local of MultiPlayer; shared below)
400012Ah	2	R/W	SIODATA8	SIO Data (Normal-8bit and UART Mode)
400012Ch	-	-		Not used

Figura A.16: Registros de Comunicación Serial (1).

Keypad Input

4000130h	2	R	KEYINPUT	Key Status
4000132h	2	R/W	KEYCNT	Key Interrupt Control

Figura A.17: Registros de los botones.

Serial Communication (2)

4000134h	2	R/W	RCNT	SIO Mode Select/General Purpose Data
4000136h	-	-	IR	Ancient - Infrared Register (Prototypes only)
4000138h	-	-		Not used
4000140h	2	R/W	JOYCNT	SIO JOY Bus Control
4000142h	-	-		Not used
4000150h	4	R/W	JOY_RECV	SIO JOY Bus Receive Data
4000154h	4	R/W	JOY_TRANS	SIO JOY Bus Transmit Data
4000158h	2	R/?	JOYSTAT	SIO JOY Bus Receive Status
400015Ah	-	-		Not used

Figura A.18: Registros de Comunicación Serial (2).

Interrupt, Waitstate, and Power-Down Control

4000200h	2	R/W	IE	Interrupt Enable Register
4000202h	2	R/W	IF	Interrupt Request Flags / IRQ Acknowledge
4000204h	2	R/W	WAITCNT	Game Pak Waitstate Control
4000206h	-	-		Not used
4000208h	2	R/W	IME	Interrupt Master Enable Register
400020Ah	-	-		Not used
4000300h	1	R/W	POSTFLG	Undocumented - Post Boot Flag
4000301h	1	W	HALTCNT	Undocumented - Power Down Control
4000302h	-	-		Not used
4000410h	?	?	?	Undocumented - Purpose Unknown / Bug ??? 0FFh
4000411h	-	-		Not used
4000800h	4	R/W	?	Undocumented - Internal Memory Control (R/W)
4000804h	-	-		Not used
4xx0800h	4	R/W	?	Mirrors of 4000800h (repeated each 64K)
4700000h	4	W	(3DS)	Disable ARM7 bootrom overlay (3DS only)

Figura A.19: Registros de Interrupciones.



A.4. ESCENAS

A.4.1. ESCENAS



Figura A.20: Créditos iniciales (1).



Figura A.21: Créditos iniciales (2).



Figura A.22: Menú ingame.



Figura A.23: Segundo nivel (1).



A.4. ESCENAS

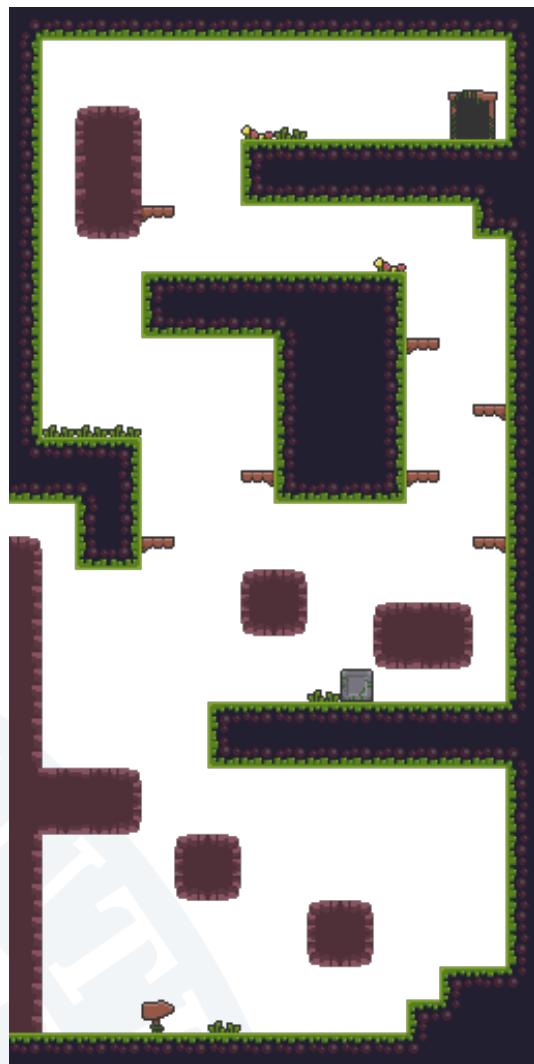


Figura A.24: Segundo nivel (2).

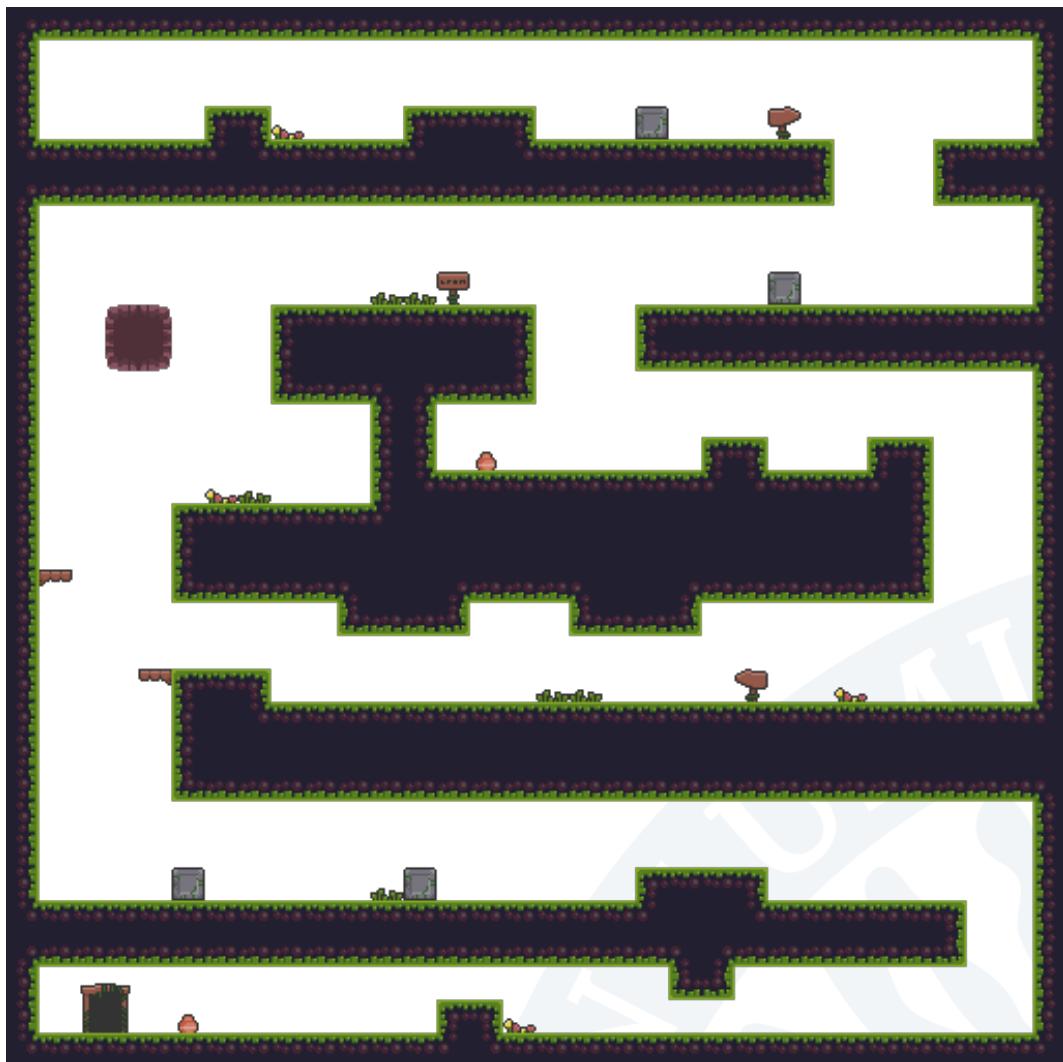


Figura A.25: Tercer nivel (1).



A.4. ESCENAS

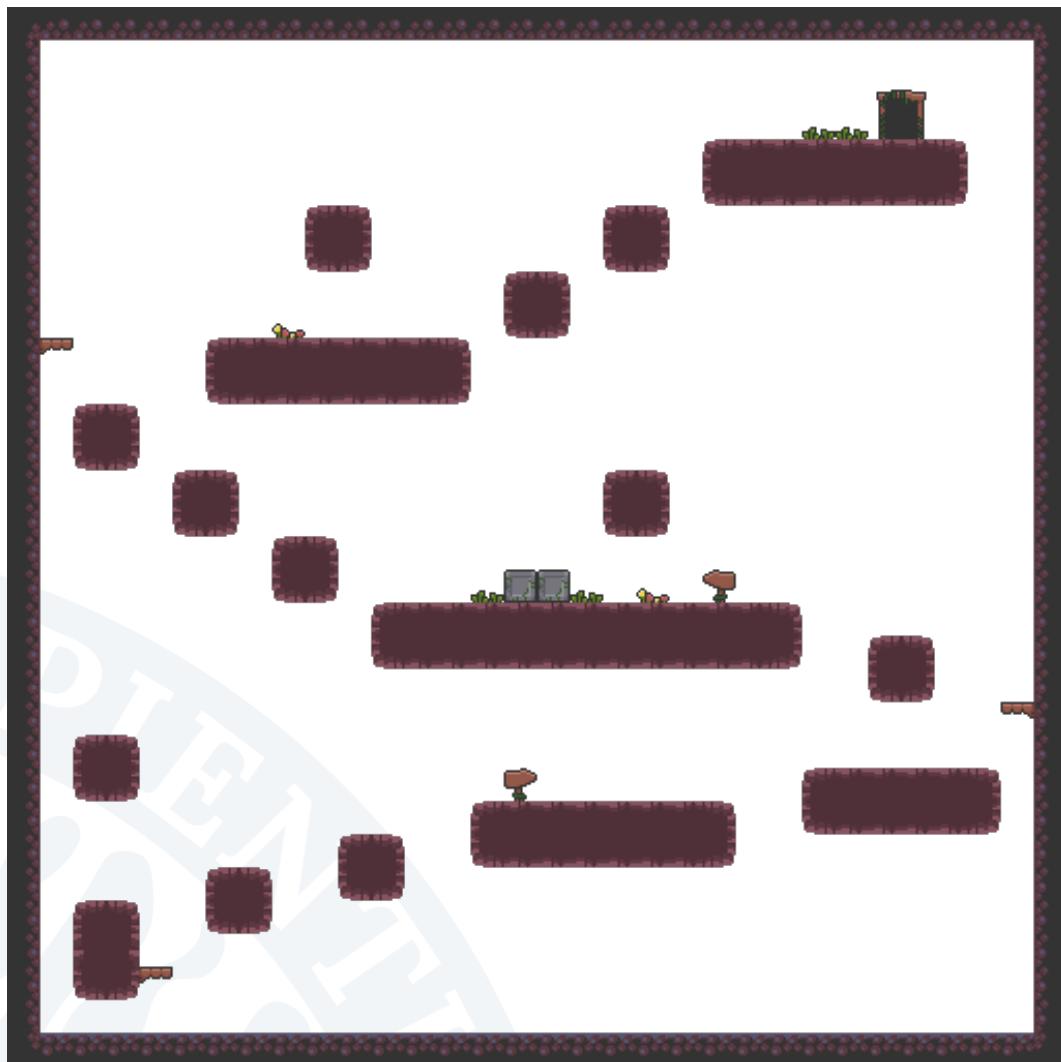


Figura A.26: Tercer nivel (2).

A.5 AUTOMATIZACIÓN MEDIANTE SCRIPTS DE BASH

```

1 #!/bin/bash
2
3 if [ ! $# -ge 1 ]; then
4     echo -e "usage:\t./script <images>"
5     exit -1;
6 fi
7
8 TMP_FILE=.result.png
9
10 convert $@ -background none -append $TMP_FILE
11 superfamiconv palette -i $TMP_FILE -M gba -W 8 -H 8 --color-zero 000000 -d palette.bin -v
12 rm $TMP_FILE
13

```

Listado A.1: Script (palettegen.sh) encargado de generar la paleta de colores para todas las imágenes proporcionadas.

```

1 #!/bin/bash
2
3 if [ ! $# -ge 2 ]; then
4     echo -e "usage:\t./script <images> <palette>"
5     exit -1;
6 fi
7
8 TMP_FILE=.result.png
9
10 convert ${*%${!#}} -background none -append $TMP_FILE
11 superfamiconv tiles -i $TMP_FILE -p ${@:$#} -M gba -B 8 -W 8 -H 8 -d tiles.bin -v
12 rm $TMP_FILE
13

```

Listado A.2: Script (tilesgen.sh) encargado de generar un *tileset* a partir de cada grupo de imágenes proporcionado. También hay que incluir la paleta generada anteriormente.

```

1 #!/bin/bash
2
3 if [ ! $# -eq 3 ]; then
4     echo -e "usage:\t./script <image> <tiles> <palette>"
5     exit -1;
6 fi
7
8 superfamiconv map -i $1 -t $2 -p $3 -M gba -B 8 -W 8 -H 8 -d map.bin -v
9

```

Listado A.3: Script (mapgen.sh) encargado de generar un mapa a partir de un *tileset*, paleta e imagen.

A.6. FUNCIONES DE LA BIOS

```

1  #!/bin/bash
2
3  mkdir -p out
4
5  ./include/palettegen.sh *.png
6
7  ls | grep -o "\w\+_" | sort -u | while read group; do
8      ./include/tilesgen.sh ${group}*.png
9      ls | grep "$group\w\+\.png" | while read file; do
10         ./include/mapgen.sh $file tiles.bin palette.bin
11         name='echo $file | sed 's/.png//g'
12         mv map.bin out/${name}_map.bin
13     done
14     mv tiles.bin out/${group}tiles.bin
15 done
16
17 mv palette.bin out/palette.bin
18

```

Listado A.4: Script encargado de procesar todas las imágenes utilizando palettegen.sh, tilesgen.sh y mapgen.sh.

A.6 FUNCIONES DE LA BIOS

Valor	Función
0x00	SoftReset
0x01	RegisterRamReset
0x02	Halt
0x03	Stop
0x04	IntrWait
0x05	VBlankIntrWait
0x06	Div
0x07	DivArm
0x08	Sqrt
0x09	ArcTan
0x0A	ArcTan2
0x0B	CPUSet
0x0C	CPUFastSet
0x0D	BiosChecksum
0x0E	BgAffineSet
0x0F	ObjAffineSet

Tabla A.1: Funciones de la BIOS a utilizar con la instrucción *swi* (1).

Valor	Función
0x10	BitUnPack
0x11	LZ77UnCompWRAM
0x12	LZ77UnCompVRAM
0x13	HuffUnComp
0x14	RLUnCompWRAM
0x15	RLUnCompVRAM
0x16	Diff8bitUnFilterWRAM
0x17	Diff8bitUnFilterVRAM
0x18	Diff16bitUnFilter
0x19	SoundBiasChange
0x1A	SoundDriverInit
0x1B	SoundDriverMode
0x1C	SoundDriverMain
0x1D	SoundDriverVSync
0x1E	SoundChannelClear
0x1F	MIDIKey2Freq

Tabla A.2: Funciones de la BIOS a utilizar con la instrucción *swi* (2).

Valor	Función
0x20	MusicPlayerOpen
0x21	MusicPlayerStart
0x22	MusicPlayerStop
0x23	MusicPlayerContinue
0x24	MusicPlayerFadeOut
0x25	MultiBoot
0x26	HardReset
0x27	CustomHalt
0x28	SoundDriverVSyncOff
0x29	SoundDriverVSyncOn
0x2A	GetJumpList

Tabla A.3: Funciones de la BIOS a utilizar con la instrucción *swi* (3).



Resumen/Abstract

En este trabajo se ha desarrollado desde cero un nuevo videojuego de plataformas para la Game Boy Advance, una consola famosa en el fenómeno retro al ser una plataforma ideal para aprender y extender los conocimientos adquiridos en el Grado de Ingeniería Informática, en especial aquellos que involucren programación sin múltiples niveles de abstracción.

A lo largo del trabajo se estudian los detalles internos de la máquina y el diseño e implementación de un juego destinado a ejecutarse en un dispositivo de modestas prestaciones.

In this project the aim is to develop from scratch a new platform game for the Game Boy Advance, a device well-known within the retro community for providing an ideal platform for those who seek to learn and extend the knowledge gained during a Degree in Computer Engineering, particularly with respect to bare-metal programming.

The report describes the internal components of the device while also explaining the process, both in terms of design and implementation, of developing a game for a device with limited capabilities.