UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Tutorial práctico de desarrollo de videojuegos para ordenadores MSX

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Frederic García Nieto

Tutor: Jorge González Mollá

Curso 2019-2020

Resumen

Este trabajo pretende ilustrar cómo desarrollar videojuegos en ordenadores actuales para el estándar MSX de ordenadores domésticos de 8 bits haciendo uso de herramientas de software abierto a través de la compilación cruzada de proyectos mediante la librería FUSION-C. Al tratarse de ordenadores clásicos de 8 bits, con pocos recursos *hardware*, cual sistema empotrado actual, se precisa conocer la arquitectura *hardware* del sistema MSX así como hacer uso de programación eficiente para aprovechar al máximo los recursos del computador, siendo necesario desarrollar los conocimientos aprendidos durante los estudios de grado para aplicarlos a estas máquinas.

Palabras clave: MSX, videojuegos, compilación cruzada, desarrollo *homebrew*, retroinformática.

Resum

Aquest treball pretén il·lustrar com desenvolupar videojocs en ordinadors actuals per a l'estàndard MSX d'ordinadors domèstics de 8 bits fent ús d'eines de programari obert a través de la compilació creuada de projectes mitjançant la llibreria FUSION-C. Al tractar-se d'ordinadors clàssics de 8 bits, amb pocs recursos *hardware*, com si d'un sistema encastat actual es tractara, fa falta conèixer l'arquitectura *hardware* del sistema MSX així com fer ús de programació eficient per aprofitar al màxim els recursos de l'ordinador, sent necessari el desenvolupar els coneixements apresos durant els estudis de grau per aplicar-los a aquestes màquines.

Paraules clau: MSX, videojocs, compilació creuada, desenvolupament *homebrew*, retroinformàtica.

Abstract

This work aims to illustrate how to develop videogames in current computers for the MSX standard of 8-bit home computers using open software tools through the cross-compilation of projects by the utilization of the FUSION-C library. Being classic 8-bit computers, with few hardware resources, which current embedded system, it is necessary to know the hardware architecture of the MSX system as well as make use of efficient programming to make the most of the computer resources, being necessary to apply what was learned during the degree to apply it to these machines.

Keywords : MSX, videogames, cross-compilation, homebrew development, retro-computing.

Tabla de contenidos

Índice de figuras

Índice de tablas

1 . Introducción

1.1 . Motivación

La primera razón existente que motiva la realización de este trabajo es la personal pues el primer contacto que tuve con la informática fue a través de un ordenador del estándar MSX en pleno auge de la informática doméstica en España. Periodo coetáneo al conocido como "Edad de oro del software español"[1], donde la industria española de desarrollo de videojuegos, aunque precaria, llegó a ser la segunda en cuanto a producción en Europa. Tratando de emular los videojuegos a los que jugaba en este ordenador que tuvo lugar mi aproximación a la programación y a sus lenguajes, concretamente al lenguaje interpretado BASIC que incorporaba de fábrica el ordenador. El tratar de entender el funcionamiento de la máquina y querer desarrollar programas para ella, motivaron que iniciara los estudios universitarios de informática. Con los conocimientos adquiridos, tanto a nivel de estructura y arquitectura de computadores como de paradigmas y lenguajes de programación, con este trabajo pretendo, de alguna forma, "cerrar el círculo" iniciado en aquel entonces.

La segunda razón es que, hoy en día, todavía está vivo el sistema MSX. Existe una amplia y activa comunidad de usuarios de MSX que realiza eventos anuales exclusivos a lo largo de la geografía española: "Reunión de usuarios de MSX de Barcelona"[2], "Reunión de usuarios de MSX de Sevilla", por ejemplo; o participa dentro de otros eventos donde concurren más sistemas de 8-bits, como RetroMadrid[3], RetroZaragoza[4] o Retropolis[5], esta última organizada por el Museo de Informática de la Universidad Politécnica de Valencia. Además, se produce nuevo *hardware*[6] para los ordenadores MSX e incluso competiciones de desarrollo *homebrew* de videojuegos como la *MSXdev Contest*[7]. A pesar de que el microprocesador Z80 de Zilog (procesador central del MSX), apareció en el mercado en 1976, versiones de este siguen fabricándose actualmente siendo muy utilizados en sistemas embebidos en forma de

1 https://es.wikipedia.org/wiki/Edad_de_oro_del_software_espa%C3%B1ol

2 RU MSX Barcelona: https://www.aamsx.com/reuniones_ES.php

3 RetroMadrid: https://www.retromadrid.org/

4 RetroZaragoza: http://www.retrozaragoza.com/

5 Retropolis: http://museo.inf.upv.es/es/retropolisvalencia/

6 https://www.msx.org/wiki/#MSX_Hardware

7 MSXdev: https://www.msxdev.org/

microcontrolador, al no precisar, los sistemas embebidos, de tanta potencia de cómputo ni de una amplia capacidad de memoria[8].

La tercera razón que motiva este trabajo es esta similitud de cualidades entre los sistemas embebidos y los ordenadores, hoy clásicos, de los años 80. Pues, *grosso modo* y a nivel de aprendizaje, podría contemplarse, a nivel de recursos, un ordenador MSX como un sistema embebido y, por tanto, el alumno de Grado de Informática podría practicar el método de programación y compilación cruzada[9], ampliamente usada en la programación de microcontroladores y sistemas embebidos, a la vez que podría participar en competiciones de desarrollo *homebrew* de videojuegos para MSX, aportando su granito de arena a la comunidad y alargando el tiempo de vida de estos ordenadores.

La cuarta y última razón es la existencia del Museo de Informática de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia, que dispone entre sus fondos de varios ejemplares de ordenadores MSX no sólo para su exposición. Esto, sumado a que el propio museo organiza el evento de retroinformática Retrópolis y que la E.T.S. de Ingeniería Informática organiza talleres y competiciones de programación en su semana cultural, dentro de este marco, este trabajo puede ayudar a motivar iniciativas que, a su vez, redunden en la tercera razón.

1.2 . Objetivos

El objetivo general de este trabajo es el de elaborar una guía o manual básico que ayude a abordar el desarrollo *homebrew* de videojuegos para ordenadores MSX de primera generación (MSX-1), haciendo uso de aplicaciones y herramientas modernas en ordenadores actuales mediante la compilación cruzada de proyectos.

De este objetivo general derivan los siguientes objetivos específicos:

- Estudiar la arquitectura del sistema MSX y del *hardware* que compone el estándar para la primera generación y sus características.
- Analizar y plantear las herramientas de desarrollo libre o gratuito con el fin de preparar el entorno de desarrollo necesario para la creación de videojuegos para el MSX.
- Conocer como implementa la librería FUSION-C el acceso a los distintos componentes de la arquitectura del MSX.
- Conocer las facetas y roles de desarrollo de un videojuego empleados durante la "Edad de oro del software español" y vigentes en el desarrollo *homebrew,* basados en equipos de desarrollo pequeños (programador, grafista y músico) o incluso de desarrollador en solitario.
- Aprendizaje basado en proyecto mediante el desarrollo y programación de una versión del mítico videojuego *Bomb Jack* de Tehkan.

8 https://www.retroaccion.org/exposicion-el-impacto-del-microprocesador-zilog-z80-0

9 https://es.wikipedia.org/wiki/Compilador_cruzado

El principal destinatario de este trabajo es el alumno de estudios de Grado en Ingeniería Informática por las razones (tres y cuatro) expuestas en el punto anterior (Motivación). Por ello se considera que el lector tiene los conocimientos básicos en Ingeniería Informática de, al menos, segundo curso de grado.

1.3 . Estructura de la memoria

El presente trabajo se estructura en tres bloques repartidos en 7 capítulos. El primer bloque aborda los aspectos organizativos y conclusiones de este trabajo y se compone de los capítulos 1, 2 y 7. El segundo bloque describe la arquitectura del ordenador MSX (*hardware*) y las aplicaciones y herramientas (*software*) que compondrán el entorno de desarrollo de videojuegos. Este bloque está compuesto por los capítulos 3, 4 y 5. El tercer bloque trata del desarrollo de un videojuego concreto y de los aspectos que ello conlleva (capítulo 6).

A continuación se realiza una breve descripción del contenido de cada uno de ellos:

- **Capítulo 1, Introducción**: Este capítulo expone la motivación que origina este trabajo, los objetivos planteados, cómo se estructura la memoria del trabajo y comentarios sobre la bibliografía utilizada.
- **Capítulo 2, Estudio estratégico**: En este capítulo se realiza un análisis de los Trabajos Finales de Grado ya realizados que abordan el desarrollo *homebrew* o el ordenador MSX, además de plantear una propuesta de gestión organizativa para la realización de este trabajo.
- **Capítulo 3, El sistema MSX**: En este capítulo se presenta la arquitectura del sistema MSX y se describen las unidades funcionales o partes que lo componen.
- **Capítulo 4, El entorno de desarrollo**: en este capítulo se presentan las herramientas y aplicaciones utilizadas y que conforman un completo entorno de desarrollo cruzado para la realización *homebrew* de videojuegos destinados a ordenadores MSX.
- **Capítulo 5, La librería FUSION-C**: En este capítulo se presentan algunas funciones, agrupadas en módulos, que integran la librería FUSION-C y que implementan el acceso y el tratamiento de los componentes que conforman el *hardware* del sistema MSX de primera generación.
- **Capítulo 6, Bomb Jack**: En este capítulo se aborda el proyecto-aprendizaje basado en el desarrollo y programación, según el modelo clásico del ciclo de vida del *software*, de una versión para ordenadores MSX de primera generación del mítico videojuego Bomb Jack de Tehkan.
- **Capítulo 7, Conclusión**: En este último capítulo se analiza el cumplimiento de los objetivos y los posibles trabajos futuros y ampliaciones.

En documentos a parte se incluyen varios anexos que o bien desarrollan y profundizan algunos capítulos (capítulo 2: ANEXO I - Diccionario de la estructura de desglose de trabajo; capítulo 4: ANEXO II - Instalación del SDK y ANEXO III -

Configurar la aplicación Gimp para pintar imágenes SCREEN 2 y capítulo 6: ANEXO IV - Máquina de estados del avatar) o bien conforman el código fuente del videojuego y de programas adicionales desarrollados para la implementación (ANEXO V - Proyecto aprendizaje - Código fuente) y la imagen de disco y ROM del videojuego (ANEXO VI - Proyecto aprendizaje - BombJack Imagen de disco (DSK y ROM).

Pese a que el destinatario de este trabajo es el alumno de Grado de Ingeniería Informática, se ha tenido a bien incluir un pequeño glosario al final de la memoria para aquel lector ajeno a la temática del trabajo.

1.4 . Notas bibliográficas

Las fuentes bibliográficas utilizadas para la realización del trabajo se hallan principalmente en libros, revistas, páginas web de Internet y Trabajos Final de Grado que tratan el desarrollo *homebrew* o introducen el sistema MSX.

Tanto las revistas como la mayoría de libros consultados pertenecen a la biblioteca personal de quien suscribe el presente trabajo. La consulta de los libros [1][2][3] ha sido posible gracias al afán de preservación de conocimiento de la época por parte de la comunidad de usuarios de MSX, quien los ha puesto al libre alcance de los usuarios en formato digital a través de páginas web de Internet.

Los Trabajos Final de Grado consultados forman parte del repositorio RiuNet de la Universidad Politécnica de Valencia.

Al inicio de cada capítulo se mencionarán las fuentes bibliográficas consultadas para la elaboración del mismo.

2 . Estudio estratégico

En este capítulo se presenta la estrategia planificada para abordar el desarrollo del presente trabajo. El capítulo se ha dividido en tres secciones. La primera de ellas hace un repaso sobre Trabajos Fin de Grado de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia previos que tratan los temas de de desarrollo de videojuegos y ordenadores domésticos de los años 80 del siglo XX. La segunda sección enuncia una propuesta de trabajo que cumpla con los objetivos marcados y que, posteriormente, se desarrolla en la tercera sección con una perspectiva de gestión de proyectos.

2.1 . Trabajos anteriores

Existen varios Trabajos Final de Grado de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia que tratan el desarrollo de videojuegos para diferentes plataformas, como muestra se encuentran:

- López Rodríguez, I. (2011). *Guía de desarrollo para Nintendo DS*. http://hdl.handle.net/10251/11221
- Sarriá Trejo, MDC. (2012). *Evaluación e implementación de juegos clásicos para Nintendo DS*. http://hdl.handle.net/10251/17621
- Martínez Pérez, JL. (2014). *Desarrollo de un videojuego para Android e iOS*. http://hdl.handle.net/10251/43430
- Musoles Tornador, S. (2014). *Desarrollo de videojuegos sobre la plataforma Android*. http://hdl.handle.net/10251/43331
- Aragó García, V. (2014). *Diseño e implementación de un videojuego para dispositivos móviles Android: Castle & Goblins*. http://hdl.handle.net/10251/43335
- Jaén Gomariz, JD. (2015). *Tutorial práctico para desarrollo de videojuegos sobre plataforma Nintendo NDS*. http://hdl.handle.net/10251/56433
- García Gallart, S. (2016). *Desarrollo de un videojuego mediante Unity: MushChan Run*. http://hdl.handle.net/10251/70826
- Lara García, D. (2018). *Diseño e implementación de videojuegos para dispositivos móviles con el lenguaje Scratch*. http://hdl.handle.net/10251/111171

También existen trabajos previos que abordan los microordenadores domésticos de los años 80 del siglo XX, como, por ejemplo:

- Castellano Munuera, IJ. (2013). *Los microordenadores Amstrad: pasado y presente*. http://hdl.handle.net/10251/32300
- Cervelló Fenollar, N. (2014). *Los microordenadores Atari: pasado y presente*. http://hdl.handle.net/10251/48154
- Estruch Miñana, A. (2014). *Los microordenadores Sinclair: pasado y presente*. http://hdl.handle.net/10251/43336
- Martín Sesma, S. (2016). *Arqueología informática: los ordenadores MSX en los inicios de la microinformática doméstica*. http://hdl.handle.net/10251/70909

Este último trabajo [Martín Sesma 2016] introduce, con anterioridad a este trabajo, el sistema MSX. Lo hace desde el análisis histórico y estudio descriptivo de la evolución del sistema en sus diversas generaciones, la revisión de los videojuegos más conocidos para el sistema y de recursos existentes en Internet que permiten, entre otras, emularlo.

2.2 . Propuesta

El principal lenguaje de programación usado para desarrollar videojuegos para ordenadores MSX, tanto en la "Edad de oro del software español" como hoy día en el desarrollo *homebrew*, es el lenguaje ensamblador —lenguaje de segunda generación—. Esto es debido a que permite programar la máquina a bajo nivel obteniendo alta velocidad de ejecución. Como contraprestación, el programador debe conocer no sólo la estructura del ordenador sino también la arquitectura del microprocesador para el que programa. Se propone hacer uso de lenguajes de alto nivel de tercera generación, concretamente el lenguaje C, en lugar del tradicional lenguaje ensamblador para el desarrollo de videojuegos en MSX-1. Por una parte porque se obtienen las bondades de los lenguajes de alto nivel: abstracción respecto del *hardware* de la máquina, gestión de memoria dinámica, uso de librerías estándar, compilación de código y uso de paradigmas de programación estructurada y modular, todo ello manteniendo la alta velocidad de ejecución y sin perder la posibilidad del acceso a bajo nivel pues permiten la incrustación de código en lenguaje ensamblador[4].

Y por otra parte porque, al igual que ocurre con la programación de sistemas embebidos, existen herramientas de desarrollo para ordenadores personales actuales que generan código o procesan datos teniendo como plataforma destino de los mismos a los ordenadores MSX[5]. Esto permite al programador una mayor seguridad en cuanto a la persistencia de los proyectos que realiza, mayor flexibilidad y mayor potencia de cómputo que facilite la elaboración del arte del videojuego.

Pese a la abstracción del *hardware* que proporcionan los lenguajes de alto nivel, principalmente respecto del microprocesador, es necesario conocer la estructura y arquitectura de la máquina y cómo esta se organiza para entender su funcionamiento. Así mismo, también es necesario conocer el conjunto de herramientas *software* que constituyan el entorno de desarrollo para la realización de videojuegos, así que se propone, como primera parte de este trabajo, elaborar una documentación completa del sistema MSX de primera generación que sirva de guía de consulta rápida para el

programador y elaborar, también, documentación que plantee herramientas *software* de desarrollo libre o gratuito, tanto a nivel de creación del arte del videojuego como de una API, la librería FUSION-C, que permita la compilación cruzada de código, permitiendo al programador disponer de un entorno de desarrollo completo.

Los equipos de desarrollo de videojuegos para ordenadores de 8-bits, tanto en la "Edad de oro del software español" como en el desarrollo *homebrew*, son pequeños, basados en los paradigmas de, o bien estar compuestos de tres personas: un grafista que realice el arte gráfico del videojuego, un músico que realice el arte sonoro (músicas y efectos de sonido) y un programador que realice la lógica del juego; o bien lo compone un único desarrollador en solitario que se encarga de las tres facetas anteriores del videojuego: gráficos, sonidos y programación[6].

Se propone, como segunda parte de este trabajo, realizar el proyecto de aprendizaje de una nueva versión del videojuego *Bomb Jack* para ordenadores MSX-1 bajo el paradigma de desarrollador en solitario, por considerar que pedagógicamente es el más interesante al tratar los tres perfiles básicos en el desarrollo de videojuegos para estos ordenadores tanto en la "Edad de oro del software español" como en el desarrollo *homebrew* actual. Se propone, también, la realización del videojuego según la metodología de Ingeniería del Software y desarrollo de calidad basado en el modelo clásico o en cascada del ciclo de vida del software, evitando así caer en los errores que produjeron la Crisis del Software[10] en la década de los 70 del siglo XX.

El paradigma de desarrollo escogido obliga a planificar los tiempos de desarrollo de forma secuencial de todas las facetas inherentes al desarrollo de un videojuego y que se podrían paralelizar si se optase por otro paradigma de desarrollo. Por ello, con tal de minimizar los tiempos en los perfiles grafista y músico, se ha escogido el videojuego *Bomb Jack*, por ser un videojuego ya existente y tener definidas las fases de diseño creativo: fantasía, mecánicas, arte…, y estar, además, dicho arte liberado en Internet.

2.3 . Plan de trabajo

En este apartado se presenta el plan de trabajo completo planificado para desarrollar este Trabajo Fin de Grado, en el que se contemplan las dos partes propuestas en el punto anterior.

2.3.1. Planificación

El *alcance* de este trabajo se ha definido previamente en el apartado 1.2 al fijar los objetivos del mismo. En el apartado 2.2, de propuesta, se han marcado los *productos entregables* como resultado del trabajo desarrollado: documentación que conforma la primera parte de la propuesta y el código de la programación de la segunda parte de la

10 https://es.wikipedia.org/wiki/Crisis_del_software

propuesta. Además como producto entregable final se proporcionará una imagen virtual del disco y otra imagen en formato ROM con la versión compilada del videojuego *Bomb Jack* versionado para ordenadores MSX-1.

También se han indicado los *recursos* necesarios, tanto humanos, al definir el paradigma de desarrollo, y materiales: conjunto de programas informáticos libres o gratuitos que conformarán el entorno de desarrollo. Falta fijar los recursos materiales *hardware* necesarios: un ordenador personal actual con sistema operativo Windows o Linux. El uso de recursos de *software* libre o gratuitos y el disponer de un solo recurso humano, hace que la estimación del *coste* económico sea nulo.

Como *criterios de éxito* se establecen, principalmente, la realización de los productos entregables además de conseguir que la versión del videojuego *Bomb Jack* cumpla con los factores de calidad del *software* de características operativas: corrección, fiabilidad, eficiencia, integridad y facilidad de uso.

2.3.2. Estructura de desglose de trabajo (EDT)

La Figura 2.1 muestra el diagrama de la estructura de desglose de trabajo con la descomposición en tareas a desarrollar que permita la realización de este TFG. Se ha considerado conveniente explosionar la tarea encargada de desarrollar el videojuego *Bomb Jack* en un nuevo diagrama (véase la Figura 2.2) para fijar las fases a ejecutar en su desarrollo según el modelo clásico del ciclo de vida de desarrollo *software* y para que, con posterioridad, pueda servir como fuente de consulta directa ante cualquier nuevo desarrollo de un videojuego.

Se anexa el diccionario de la estructura de desglose de trabajo (ANEXO I - Diccionario EDT) en documento aparte, donde se describe cada paquete de trabajo y los productos entregables asociados a ellos.

2.3.3. Planificación de plazos

Se fija el tiempo total de esfuerzo estimado para el desarrollo de este trabajo en 360 horas dado el paradigma de desarrollo escogido en la segunda parte de la propuesta, que establece el recurso humano en un único desarrollador. El total de horas de esfuerzo previstas se balancean, inicialmente, entorno al 50% del número de horas estimadas para cada una de las dos propuestas: 167 horas de esfuerzo para la primera parte de la propuesta y 193 horas para la segunda parte.

La Tabla 2.1 muestra la estimación de esfuerzo en horas necesario para la realización de cada actividad de la estructura de desglose de trabajo del TFG. Al igual que con el diagrama de la EDT, se ha considerado conveniente reflejar aparte, en otra tabla (Tabla 2.2), la estimación de horas de esfuerzo para la actividad de la EDT del desarrollo del videojuego *Bomb Jack*.

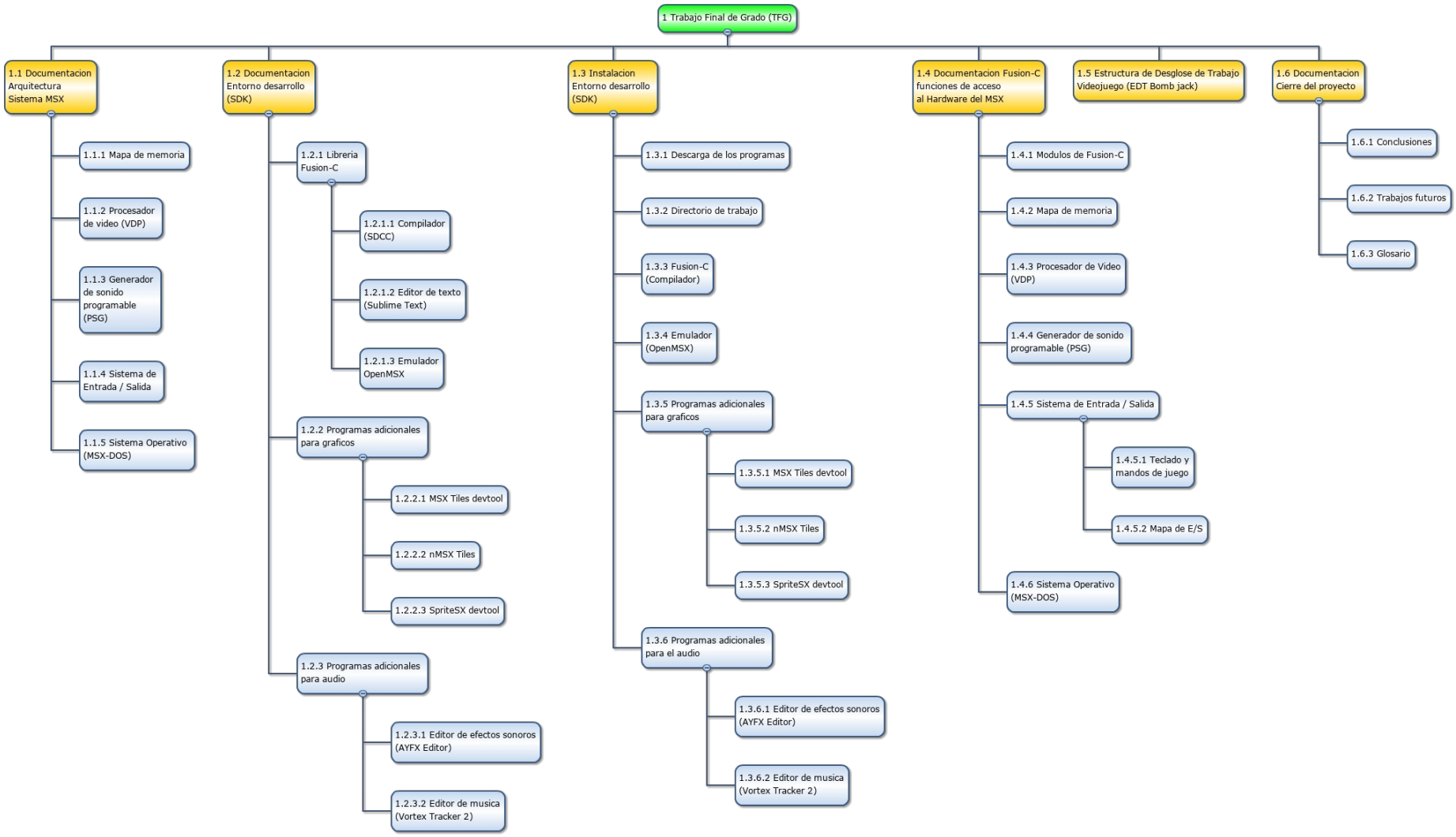

Figura 2.1: Estructura de Desglose de Trabajo del TFG

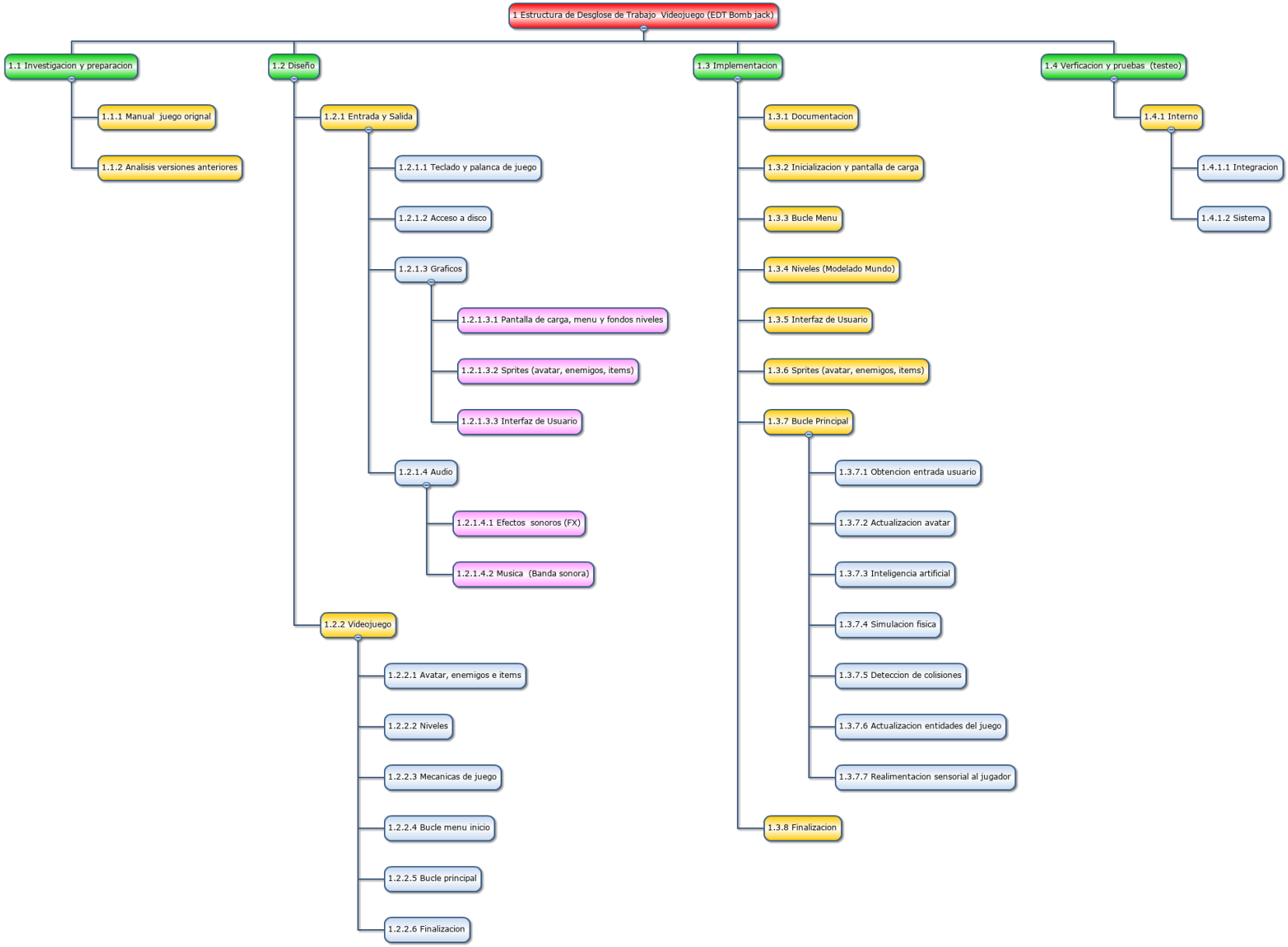

Figura 2.2: Estructura de Desglose de Trabajo del proyecto de aprendizaje (videojuego Bomb Jack versión MSX-1)

Actividades EDT (TFG)	Horas esfuerzo	Total horas / fase EDT
1.1 Documentación Arquitectura MSX		**120**
1.1.1 Mapa de memoria	32	
1.1.2 Procesador de vídeo (VDP)	48	
1.1.3 Generador de sonido programable (PSG)	20	
1.1.4 Sistema de E/S	12	
1.1.5 MSX-DOS	8	
1.2 Documentación Entorno Desarrollo (SDK)		**8**
1.2.1 Librería Fusion-C		
1.2.1.1 Compilador SDCC	1	
1.2.1.2 Editor de texto (Sublime text)	1	
1.2.1.3 Emulador OpenMSX	1	
1.2.2 Programas adicionales para gráficos		
1.2.2.1 MSX Tiles devtool	1	
1.2.2.2 nMSX Tiles	1	
1.2.2.3 SpriteSX devtool	1	
1.2.3 Programas adicionales para audio		
1.2.3.1 Editor de efectos sonoros (AYFX)	1	
1.2.3.2 Editor de música (Vortex Tracker 2)	1	
1.3 Instalación Entorno desarrollo (SDK)		**20**
1.3.1 Descarga de los programas	1	
1.3.2 Directorio de trabajo	1	
1.3.3 Fusion-C (compilador)	1	
1.3.4 Emulador (OpenMSX)	2	
1.3.5 Programas adicionales para gráficos		
1.3.5.1 MSX Tiles devtool	3	
1.3.5.2 nMSX Tiles	3	
1.3.5.3 SpriteSX devtool	3	
1.3.6 Programas adicionales para audio		
1.3.6.1 Editor de efectos sonoros (AYFX)	3	
1.3.6.2 Editor de música (Vortex Tracker 2)	3	
1.4 Documentación Fusion-C (funciones)		**7**
1.4.1 Módulos de Fusion-C	1	
1.4.2 Mapa de memoria	1	
1.4.3 Procesador de vídeo (VDP)	1	
1.4.4 Generador de sonido programable (PSG)	1	
1.4.5 Sistema de E/S		
1.4.5.1 Teclado y mandos de juego	1	
1.4.5.2 Mapa de E/S	1	
1.4.6 Sistema Operativo de disco MSX-DOS	1	
1.5 EDT Bomb Jack	193	**193**
1.6 Documentación Cierre Proyecto		**12**
1.6.1 Conclusiones	4	
1.6.2 Trabajos futuros	4	
1.6.3 Glosario	4	
Total horas de esfuerzo	**360**	360

Tabla 2.1: Horas de esfuerzo de las actividades del TFG

Actividades EDT (Bomb Jack)	Horas esfuerzo	Total horas / fase EDT
1.1 Investigación y preparación		**8**
1.1.1 Manual juego original	4	
1.1.2 Analisis versiones anteriores	4	
1.2 Diseño		**49**
1.2.1 Entrada y Salida		
1.2.1.1 Teclado y joystick	2	
1.2.1.2 Acceso a disco	2	
1.2.1.3 Gráficos		
1.2.1.3.1 Pantalla de carga, menú y fondos niveles	2	
1.2.1.3.2 Sprites (avatar, enemigos, ítems)	2	
1.2.1.3.3 Interfaz de usuario	2	
1.2.1.4 Audio		
1.2.1.4.1 Efectos de sonido (FX)	3	
1.2.1.4.2 Músicas (Banda sonora)	2	
1.2.2 Videojuego		
1.2.2.1 Avatar, enemigos e items	12	
1.2.2.2 Niveles	4	
1.2.2.3 Bucle menú inicio	6	
1.2.2.4 Bucle principal	8	
1.2.2.5 Finalización	4	
1.3 Implementación		**108**
1.3.1 Documentación	8	
1.3.2 Inicialización y pantalla de carga	4	
1.3.3 Bucle Menú	4	
1.3.4 Niveles (modelado mundo)	8	
1.3.5 Interfaz de usuario	8	
1.3.6 Sprites (avatar, enemigos, ítems)	20	
1.3.7 Bucle Principal		
1.3.7.1 Obtención entrada usuario	2	
1.3.7.2 Actualización Avatar	12	
1.3.7.3 Inteligencia Artificial	12	
1.3.7.4 Simulación física	9	
1.3.7.5 Detección de colisiones	9	
1.3.7.6 Actualización entidades del juego	4	
1.3.7.7 Realimentación sensorial al jugador	4	
1.3.8 Finalización	4	
1.4 Verificación y pruebas		**28**
1.4.1.1 Integración	20	
1.4.1.2 Sistema	8	
Total horas de esfuerzo	**193**	193

Tabla 2.2: Horas de esfuerzo de las actividades desarrollo Bomb Jack

Las figuras 2.3 , 2.4 y 2.5 muestran el diagrama de Gantt con el cronograma que planifica las fechas en las que se realizará el conjunto de actividades de este trabajo a partir de la secuencia de actividades y de la duración de horas de esfuerzo estimada. Se ha planificado un tiempo de trabajo de 20 días/mes, con un total de 20 horas/semana y a razón de 4 horas/día para la primera parte de la propuesta. Para la segunda parte, se ha variado el tiempo de trabajo planificado en 12 horas/semana. Se planifica como fecha de inicio el 1 de agosto de 2019 con previsión de finalización del TFG el 4 de marzo de 2020.

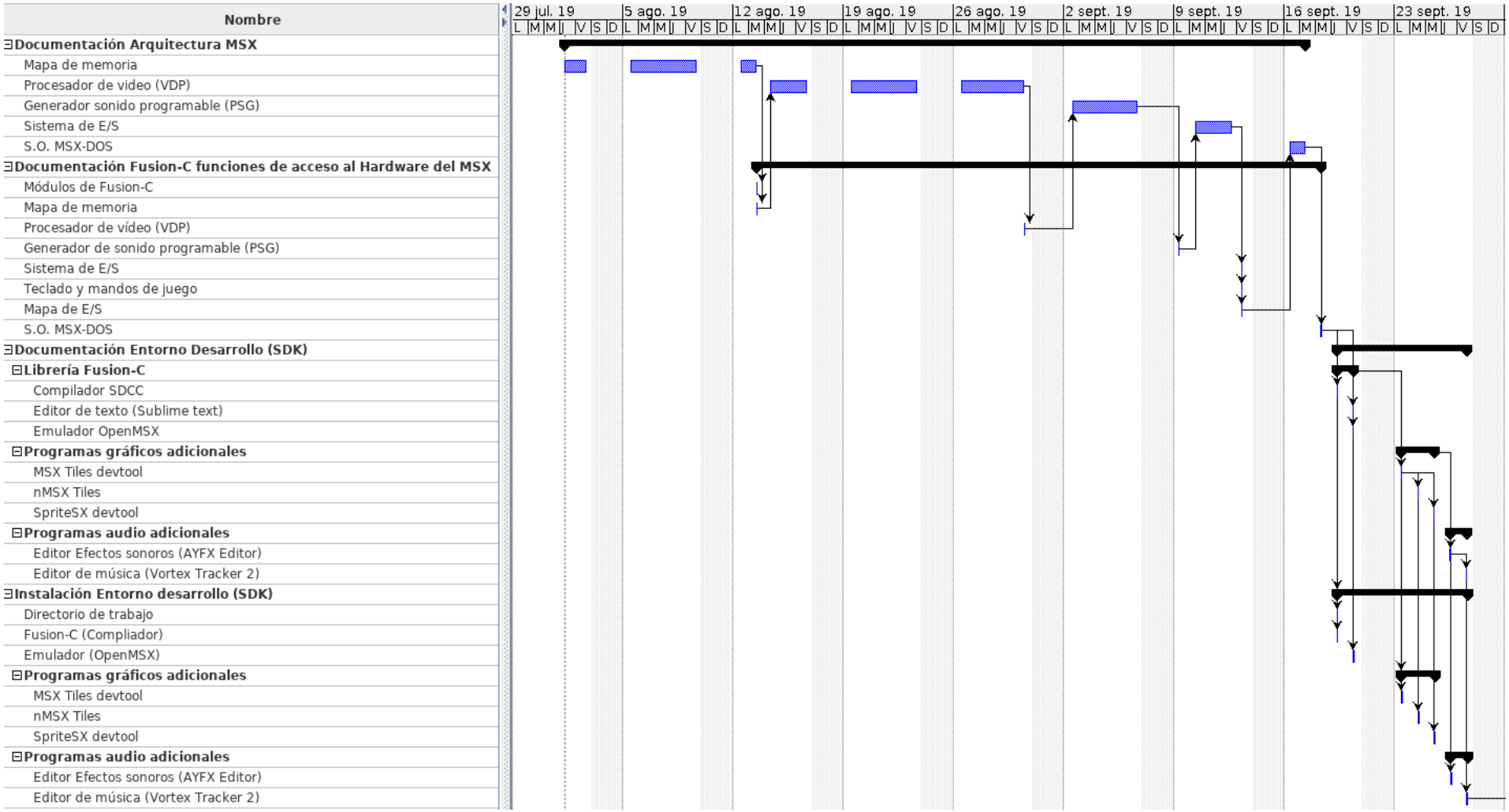

Figura 2.3: Cronograma EDT actividades primera propuesta (guía-manual documentación sobre el MSX y Entorno Desarrollo)

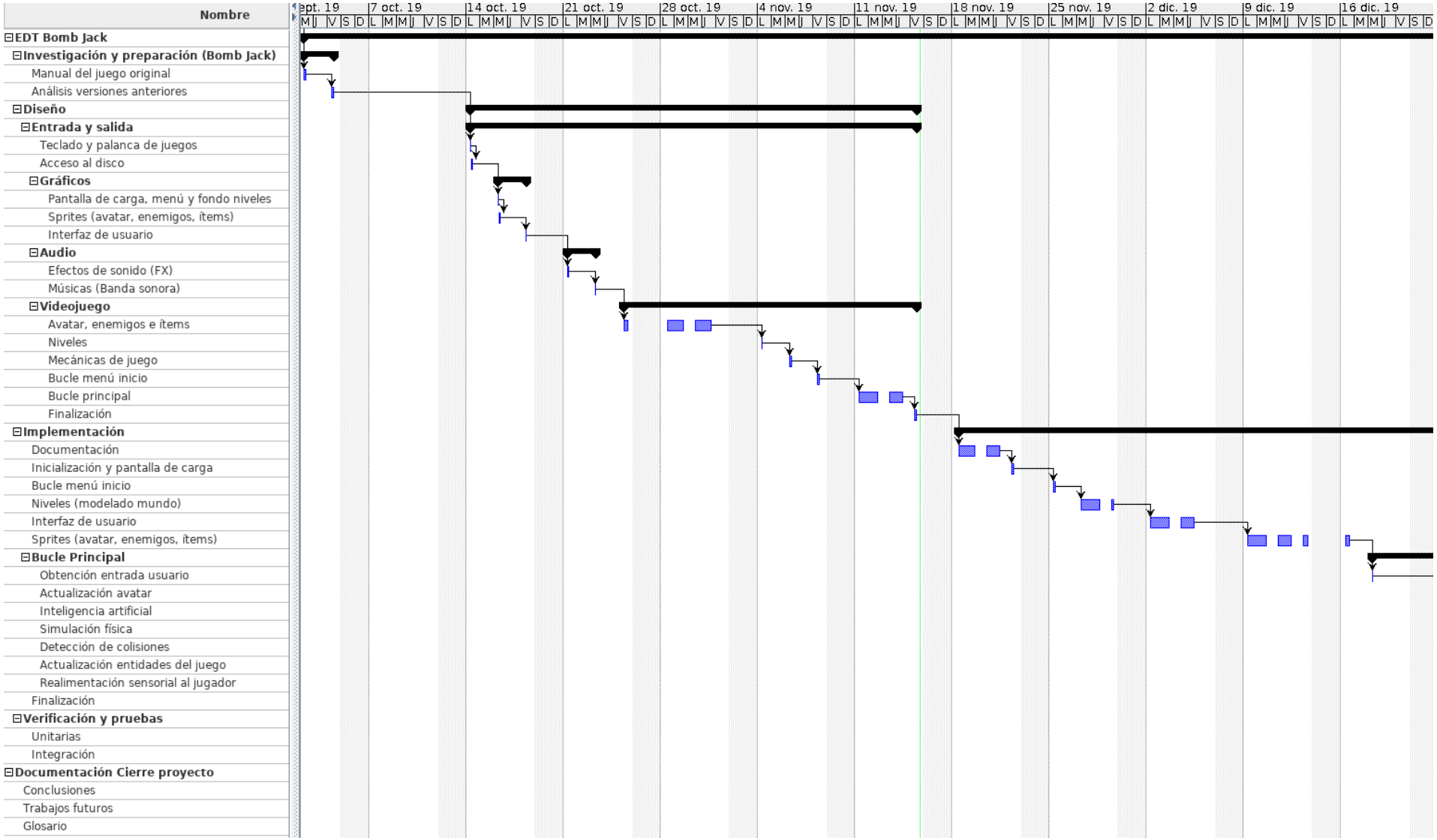

Figura 2.4: Cronograma EDT Desarrollo Bom Jack (primera parte)

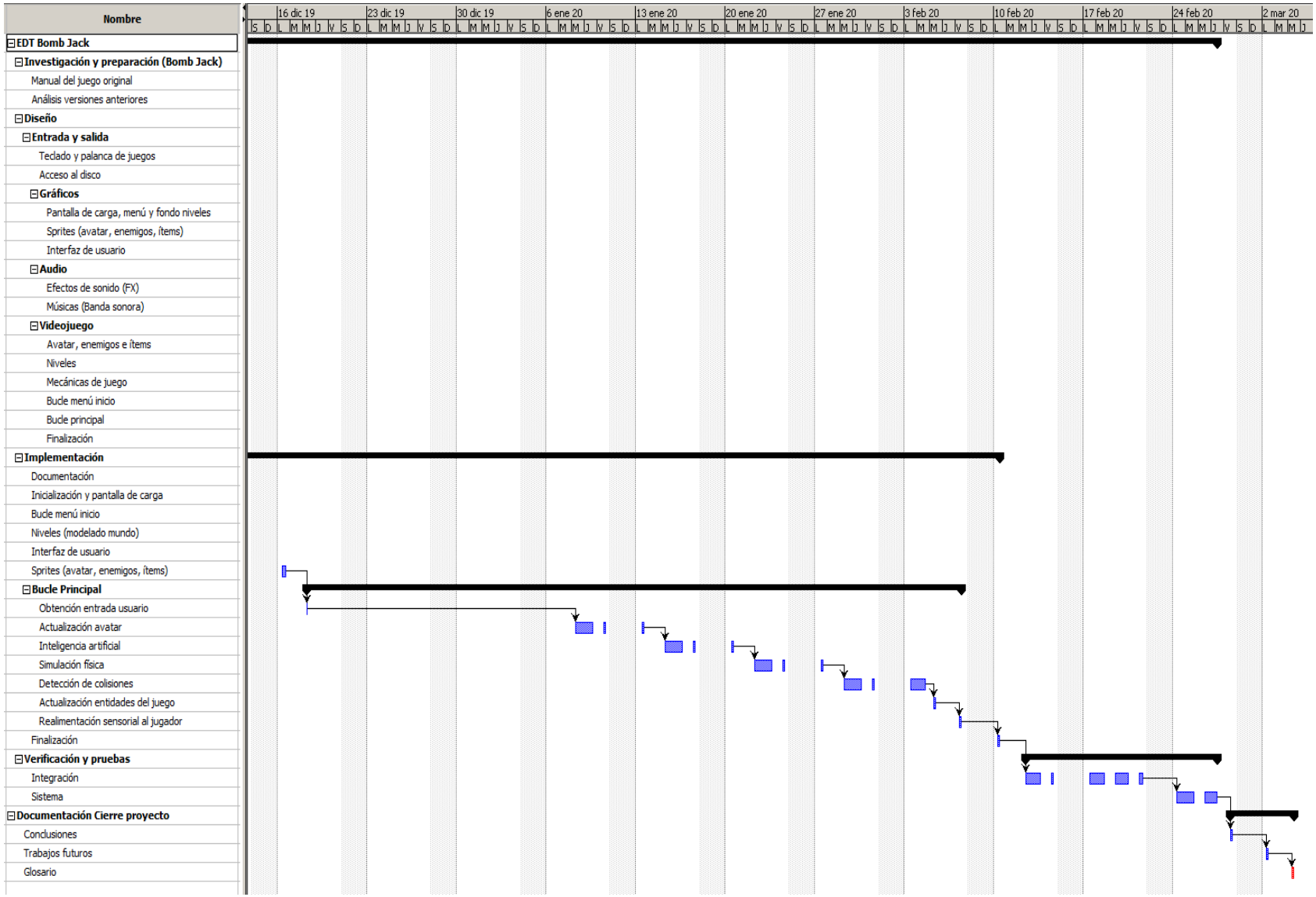

Figura 2.5: Cronograma EDT Desarrollo Bom Jack (segunda parte) y cierre del TFG

3 . El sistema MSX

En este capítulo se va a presentar la arquitectura del sistema MSX y se describirán las unidades funcionales o partes que lo componen. El capítulo se ha dividido en ocho secciones. La primera sección describe globalmente la organización de un computador MSX y las siete secciones restantes describen el funcionamiento de cada uno de los componentes que lo conforman. Las fuentes bibliográficas consultadas para su elaboración han sido [1][2][3][7][8][9][10][11][12][13][14][15][16][17][18][19][20][21][22][23][23][24][25][26][27][28].

3.1 . Estructura del estándar MSX

El sistema MSX supone una norma que estandariza una serie de características y elementos comunes en todos los ordenadores desarrollados bajo la norma, permitiendo maximizar la compatibilidad de los periféricos fabricados o programas desarrollados para ellos. La norma marca unas características mínimas que todos los ordenadores MSX deben cumplir pero permite la libertad al fabricante para añadir o alterar cualquiera de los elementos mientras cumplan las características mínimas que define la norma.

Existen 4 generaciones o revisiones de la norma: MSX-1 (1983), MSX-2 (1985), MSX-2+ (1988) y MSX Turbo-R (1990)[11]. Cada revisión de la norma ha ampliado las características comunes que todos los ordenadores deben implementar y ha buscado mantener la compatibilidad con las revisiones anteriores. Comentar todas las mejoras a la norma con cada revisión se sale del alcance que pretende este trabajo pues, siendo que cada generación es compatible con generaciones anteriores, se contempla exclusivamente el desarrollo de videojuegos sobre ordenadores MSX de primera generación o MSX-1. Por ello, se enumeran y describen solamente las características mínimas que comparten los ordenadores MSX de primera generación, que son:

- Microprocesador Z-80 A de Zilog o equivalente.
- Memoria ROM de 32 KB con la BIOS y el intérprete de BASIC MSX.
- Memoria RAM con un mínimo 8 KB de tamaño.
- Chip de vídeo (VDP) TMS9918A de Texas Instruments o equivalente.
- Chip de sonido (PSG) AY-3-8910 de General Instruments o equivalente.
- Bus de expansión de una ranura (*slot*) para cartuchos o *interfaces*.
- Un puerto para *joystick*.

11 https://en.wikipedia.org/wiki/MSX

- Conector a casete con dos velocidades posibles.

La mayoría de fabricantes de ordenadores MSX ampliaron rápidamente los ordenadores que sacaron al mercado con tamaño de memoria RAM de 64 KB, dos ranuras de expansión y dos puertos para *joystick*.

La Figura 3.1[12] ilustra la interconexión de los elementos que componen la arquitectura de la norma MSX.

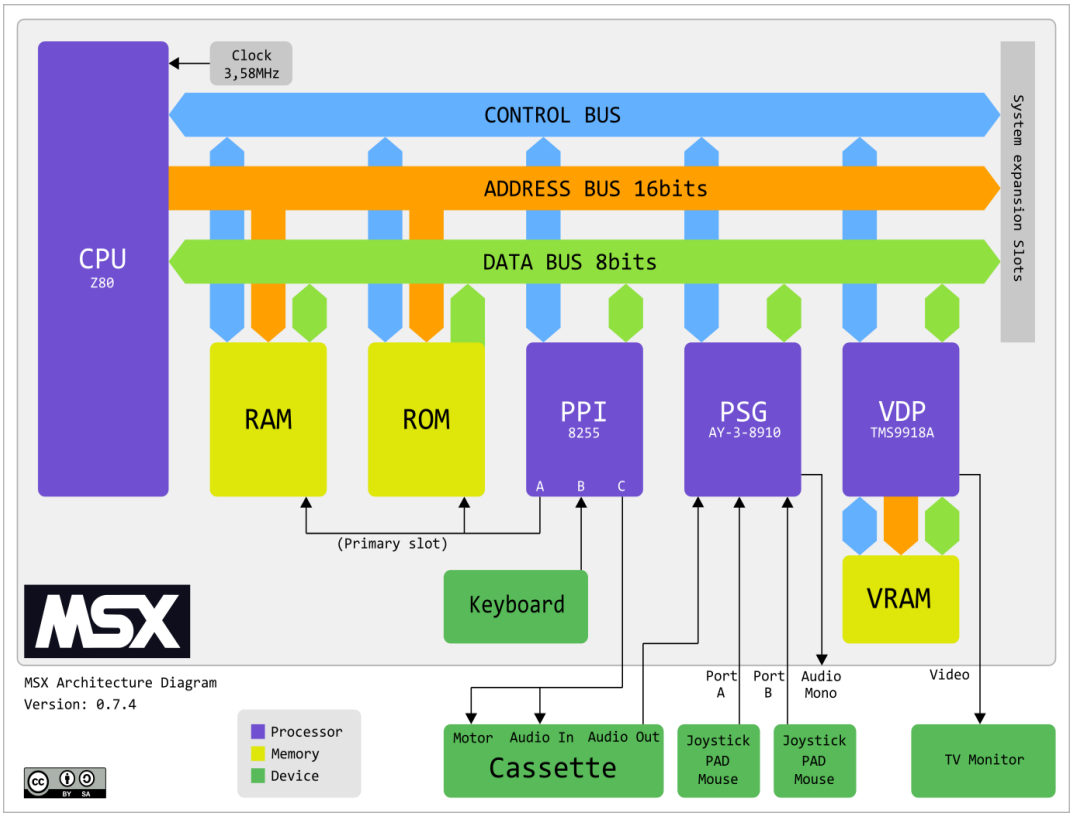

Figura 3.1: Esquema de bloques del sistema MSX

3.2 . CPU Z80 de Zilog

El procesador Z80 de Zilog pertenece a los microprocesadores de 8 bits por su tamaño de palabra de datos. Tiene un bus de direcciones de 16 bits, lo que le permite direccionar hasta 64 KB (2^{16} bytes) de memoria principal, organizando los bytes en formato *little endian*. Su arquitectura es híbrida entre la organización de acumulador y de registros generales, lo que le sitúa dentro del tipo registro-memoria. Tiene 22 registros, 18 de ellos de 8 bits y 4 de 16 bits; 12 de ellos se pueden usar en pares

12 Fuente:https://aorante.blogspot.com/2018/10/diagrama-de-la-arquitectura-del-msx-msx.html

posibilitando tener 6 registros de 16 bits. La Figura 3.2[13] muestra la arquitectura interna del procesador Z80.

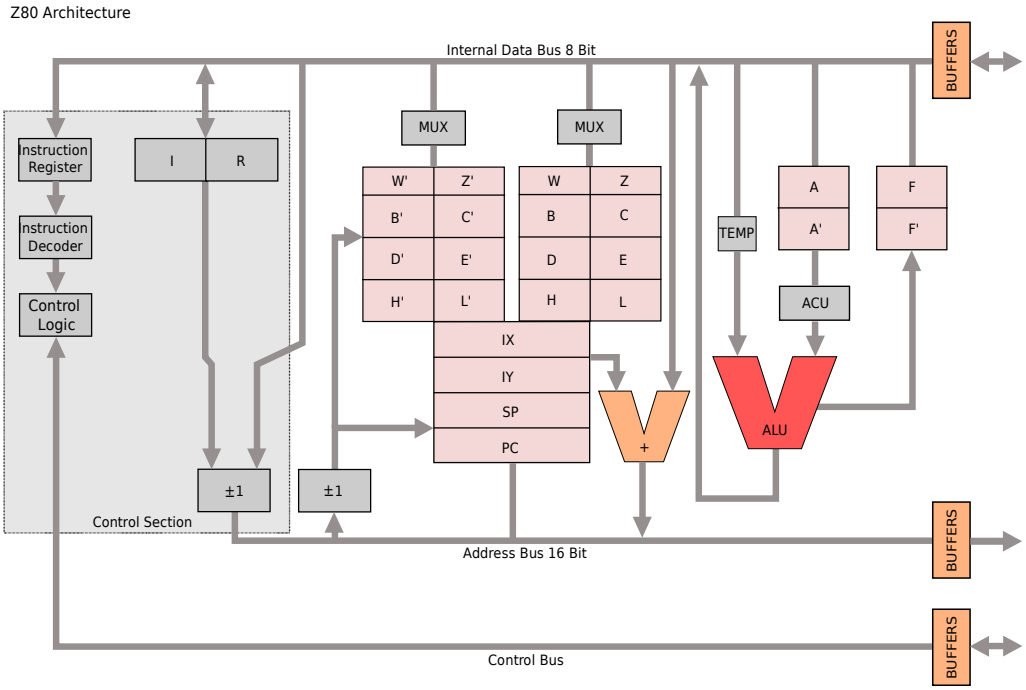

Figura 3.2: Arquitectura interna del microprocesador Z80

Su arquitectura computacional es CISC con un conjunto de instrucciones aproximado de 158 instrucciones en lenguaje ensamblador. Las instrucciones tienen una longitud de 1 ó 2 bytes, si además le acompañan datos o una dirección, son necesarios 1 ó 2 bytes adicionales. La ejecución de una instrucción puede constar entre una y siete fases :

1. Búsqueda de código de la operación.
2. Lectura/escritura de memoria.
3. Lectura/escritura de E/S.
4. Requerimiento/concesión del bus.
5. Solicitud/reconocimiento de interrupción.
6. Solicitud/reconocimiento de interrupción no enmascarable.
7. Salida de un HALT.

La frecuencia de reloj del Z80 es de 4 Mhz siendo capaz de ejecutar 0,580 MIPS[14], aunque en el sistema MSX la frecuencia se reduce a 3,58 Mhz. Esto significa que se produce un ciclo de reloj cada 286 microsegundos. La instrucción más rápida necesita 4 ciclos de reloj para ejecutarse y la más lenta tarda 21 ciclos de reloj.

13 Fuente: Appaloosa - Trabajo propio, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=2933572

14 https://en.wikipedia.org/wiki/Instructions_per_second

3.3 . Mapa de memoria

Como se ha comentado en el apartado anterior, el microprocesador Z80 tiene un bus de direcciones de 16 bits, lo que le permite gestionar simultáneamente hasta un total de 64 KB de memoria (de 0 – 65535 direcciones en notación decimal, de 0000h – FFFFh en notación hexadecimal). Este mapa de memoria, que es lineal para el microprocesador, se divide en cuatro páginas de 16 KB de tamaño cada una de ellas. Estas páginas se numeran de cero a tres (página 0 – página 3), siendo la página cero la que ocupa las direcciones más bajas de la memoria y la página 3 la que ocupa las direcciones más altas.

Tal y como se ha visto en el apartado *3.1 Estructura del estándar MSX*, la norma para los MSX de primera generación establece una memoria ROM de 32 KB con la BIOS y el intérprete de BASIC MSX. Esta memoria ROM se ubica en las páginas 0 (BIOS) y 1 (Intérprete de BASIC). También establece una memoria RAM con un mínimo 8 KB de tamaño que se ubica en la parte más alta de la memoria, la página 3, y se va ampliando hasta direcciones más bajas. De la memoria RAM, la BIOS, para uso de variables de sistema, se reserva y ocupa los 4 KB más altos de memoria. Esto deja, según los requisitos mínimos de la norma, sólo otros 4 KB de memoria RAM para programas en lenguaje BASIC. Este espacio de memoria tan limitado hizo que los fabricantes de ordenadores MSX sacaran al mercado ordenadores con mayor capacidad de memoria, ampliando esos 8 KB mínimos a 16 KB, 32 KB ó 64 KB. La Figura 3.3 muestra el mapa de memoria completo con los 64 KB del espacio de direcciones lógicas del microprocesador Z80 del MSX, con la distribución de las páginas y ubicadas las memorias ROM y RAM.

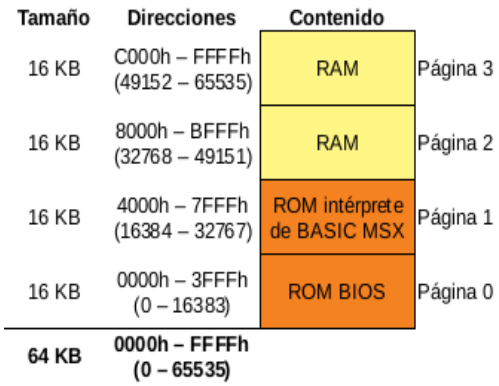

Figura 3.3: Mapa de memoria con su distribución en páginas y ubicadas la memoria ROM y RAM

Esta configuración de mapa completo sólo permite un máximo de 32 KB de memoria RAM, limitando, *a priori*, cualquier desarrollo de programas a ese tamaño de memoria. Para eliminar ese límite y poder desarrollar o ejecutar programas de mayor tamaño, el sistema MSX organiza la memoria en bancos de memoria —también llamados *slots*, ranuras o segmentos—, donde cada banco es un bloque de 65.535 (64 K) posiciones de

memoria (el espacio de direccionamiento lógico del Z80), divididos en las 4 páginas de 16 KB de tamaño cada una que pueden ocuparse por memoria ROM o RAM —los bancos de memoria no tienen porqué tener todas las páginas implementadas—. Según los mínimos establecidos por la norma, todos los ordenadores MSX deben tener al menos dos bancos primarios: un primer banco, interno, el BANCO DEL SISTEMA o *slot-0*, donde se ubican las memorias ROM y RAM (la Fig. 3.3 ilustraría este banco) y un segundo banco o *slot-1,* el BANCO DEL CARTUCHO, externo, que correspondería con la ranura para cartuchos o *interfaces* donde poder insertar memoria RAM adicional, cartuchos de programas ROM como videojuegos o controladores de periféricos como unidades de disco.

Como se ha indicado, la norma especifica un mínimo de dos bancos primarios de memoria pero el MSX es capaz de gestionar hasta cuatro de estos bancos: *slot-0* o BANCO DEL SISTEMA, *slot-1* o BANCO DEL CARTUCHO, *slot-2* y *slot-3*. La mayoría de los fabricantes implementaron los cuatro bancos primarios de memoria en gran parte de los modelos que sacaron al mercado, permitiendo así ampliar la memoria RAM del ordenador más allá de los 32 KB de tamaño. De hecho, estos cuatro bancos primarios permiten al ordenador MSX-1 gestionar hasta 256 KB de memoria entre ROM y RAM —4 bancos x 64 KB de tamaño cada uno—. Aunque esto es posible, los fabricantes, siguiendo la norma y en pro de la compatibilidad entre ordenadores MSX, dejaron el *slot-0* como BANCO DEL SISTEMA con la memoria ROM del MSX y parte de la memoria RAM (en algunos modelos), el *slot-1* como BANCO DEL CARTUCHO (externo) y los otros dos bancos (*slot-2* y *slot-3*), uno de ellos para ampliar a dos el número de bancos externos para conectar cartuchos o periféricos y el otro restante como banco interno de memoria RAM, que puede contener la parte restante de la memoria RAM que no esté en el BANCO DEL SISTEMA o bien tener sus 64 KB completos con sus cuatro páginas con memoria RAM. Esta última configuración fue la más extendida en los ordenadores MSX de primera generación con 64 KB de memoria RAM ya que, por economía, es más barato incorporar un banco de memoria con sus cuatro páginas completas (64 KB) con memoria RAM que no tener las páginas de RAM repartidas en varios bancos de memoria. La Figura 3.4 ilustra una posible organización de los bancos de memoria con esta última configuración ¡pero ojo! también podría ser el *slot-2* el que contuviese la memoria RAM[15].

Se tiene, por tanto, que el espacio de direcciones lógicas del Z80 es un subconjunto del espacio de direcciones físicas del MSX. Es decir, las cuatro páginas que gestiona el Z80 como memoria lineal pueden estar ubicadas en bancos de memoria diferentes y es posible cambiar la asignación de cualquiera de las cuatro páginas de un banco a otro pero con una restricción: la página 0 lógica, la que gestiona el Z80, sólo puede corresponder a la página 0 física de cualquier banco de memoria, la página 1 lógica sólo puede corresponder con la página 1 física de cualquier banco, la página 2 lógica sólo puede corresponder con la página 2 física de cualquier banco y la página 3 lógica sólo puede corresponder con la página 3 física de cualquier banco.

15 El ordenador Dragon MSX-64 fabricado en España lleva esta configuración. Para consultar las configuraciones de los bancos de memoria de los modelos de ordenador MSX-1 existentes, consultar la página web: https://www.msx.org/wiki/Category:MSX1_Computers

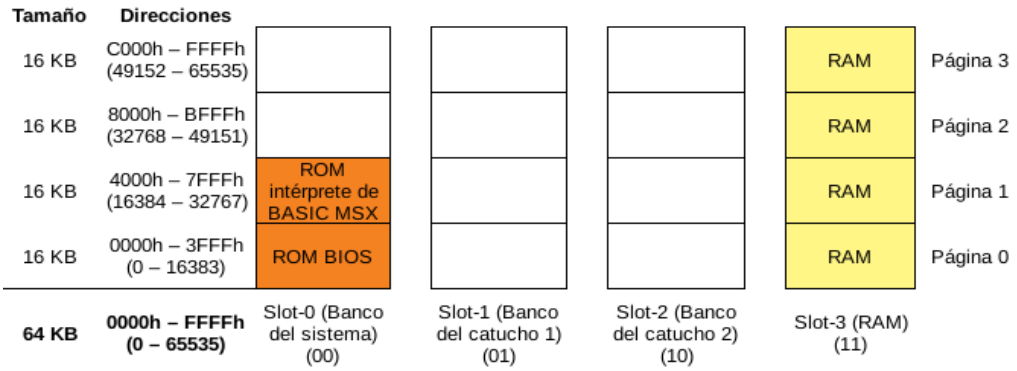

Figura 3.4: Bancos de memoria del MSX

Al iniciarse el ordenador MSX, el microprocesador Z80 configura el mapa de memoria con los 64 KB que puede gestionar analizando las páginas de memoria existentes en los bancos de memoria. Para ello, primero analiza el BANCO DEL SISTEMA, donde por norma debe figurar la memoria ROM de la BIOS y del intérprete de BASIC, asignando la página 0 y la página 1 del mapa lógico de memoria a las páginas físicas 0 y 1 del BANCO DEL SISTEMA[16]. De esta forma, las rutinas de la BIOS entran en funcionamiento. Acto seguido, el ordenador completa entonces su mapa de memoria buscando entre los bancos de memoria el mayor espacio contiguo de memoria RAM a partir de la dirección más alta de memoria (FFFFh) —que corresponde con la página 3, tal y como también especifica la norma—, y avanza hacia abajo, asignando al final del análisis las páginas 2 y 3. En caso de encontrar memoria RAM con el mismo tamaño en más de un banco de memoria, asignaría las páginas 2 y 3 del mapa lógico de memoria a aquellas páginas físicas 2 y 3 pertenecientes al banco de memoria con menor número de segmento. Si, por ejemplo, los bancos *slot-0* y *slot-3* tuviesen ambos en sus páginas 2 y 3 memoria RAM, el microprocesador asignaría las páginas 2 y 3 de mapa lógico a las del *slot-0*. Ídem en caso de encontrar memoria ROM en la página 1 de ambos bancos, la página 1 lógica se asignaría a la página 1 física perteneciente al banco de memoria con menor número de segmento.

El Z80 almacena el resultado del análisis de memoria con la correspondencia entre las páginas lógicas y físicas en el puerto de entrada/salida A8 —notación hexadecimal —. Este puerto de E/S corresponde con el registro A del *Interfaz Programable de Periféricos* (PPI, véase la Fig. 3.1 para una rápida referencia. Se tratará con mayor profundidad en el apartado 3.6). A este registro se le conoce como *registro selector de banco* y sus 8 bits de tamaño, agrupados por pares, indican el banco de memoria al que corresponde cada página lógica (he aquí el por qué el MSX puede gestionar hasta cuatro bancos de memoria). Así pues, los dos bits de menor peso hacen referencia a la página lógica 0 y su contenido, en notación binaria, indica el número del banco de memoria seleccionado al que pertenece esa página. Ídem con el resto de bits hasta llegar a los dos de mayor peso, que hacen referencia a la página lógica 3 y su contenido indica el banco asignado. La Figura 3.5 muestra el contenido del puerto de E/S A8 con la configuración del mapa de memoria resultante en la Fig. 3.3 tras el análisis de memoria partiendo de la configuración de los bancos de memoria de la Fig. 3.4.

16 Algunos ordenadores MSX fabricados por SONY, el HB-201 por ejemplo, incluyen en la página 2 del BANCO DEL SISTEMA memoria ROM con programas de Agenda personal, en este caso, la página 2 del mapa de memoria correspondería a la del BANCO DEL SISTEMA y no a memoria RAM

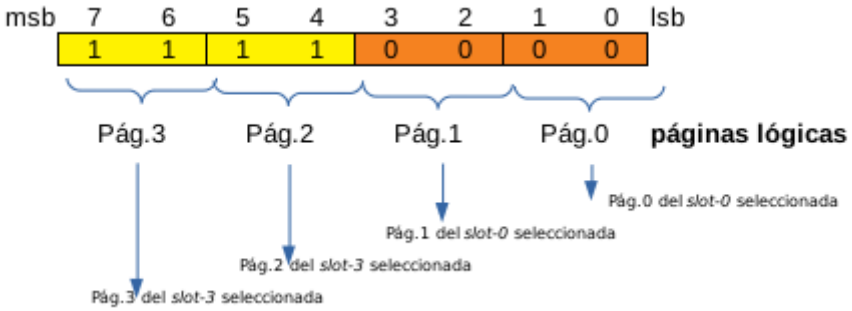

Figura 3.5: Registro selector de banco A8

En circunstancias normales no es necesario modificar el valor del *registro selector de banco* que almacena el Z80 al inicio, salvo el caso de un sistema con 64 KB o más de RAM, como es el planteado en las figuras 3.3 y 3.4, donde parte de la memoria RAM queda oculta al Z80, concretamente las páginas 0 y 1 del *slot-3*. Para poder acceder a ellas, se debe escribir en el puerto A8 los valores que activen el *slot-3* en dichas páginas. Es decir, poner a '1' los bits 0 y 1 —11 en notación binaria— en caso de querer activar la página 0 del *slot-3* y poner a '1' los bits 2 y 3 en caso de querer activar la página 1 del *slot-3*. En caso de querer disponer de los 64 KB de memoria RAM de forma simultánea, el valor a escribir en el puerto A8 será FFh —en notación hexadecimal, 256 en decimal ó 11111111 en binario—. Ha de tenerse en cuenta que al activar la página 0 del *slot-3* se desactiva la BIOS y se pierden todas las rutinas del sistema, esto implica que el programador debe implementar o hacer uso de librerías con rutinas predefinidas o funciones que manejen directamente el *hardware* del MSX.

En caso de existir algún cartucho con programas en memoria ROM —un videojuego, por ejemplo— conectado en alguno de los BANCOS DE CARTUCHO (*slots* 1 ó 2 si se sigue la configuración de bancos de memoria de la Fig.3.4), el MSX, después de seleccionar la RAM del sistema, lo ubica en la página 1 del mapa de memoria desactivando el Intérprete de BASIC y ejecutando el contenido del cartucho —en la página 0 está la BIOS, en la página 3 la RAM del sistema. La página 2 queda indefinida ya que puede contener más memoria RAM o no haber nada—. Esto para cartuchos de 16 KB de memoria ROM, donde no importará el contenido de la página 2. En caso de cartuchos de 32 KB de memoria ROM, lo ubica en la página 1 y el propio videojuego ha de encargarse de averiguar que BANCO DE CARTUCHO tiene asociado la página 1 y escribir en los bits que corresponden a la página 2 del *registro selector de banco* el número de dicho banco. Los cartuchos de más de 32 KB de memoria llevan incorporado en el propio cartucho un gestor de mapa de memoria (*mapeador*) ROM o MegaROM[17].

17 Para ver los *mapeadores* MegaROM existentes véase https://www.msx.org/wiki/MegaROM_Mappers

3.3.1 Memoria expandida (*Subslots*)

Cada uno de los bancos primarios de memoria puede expandirse en otros 4 segmentos secundarios o *subslots* de hasta 64 KB cada uno de ellos para proporcionar más memoria al sistema. Los segmentos secundarios no pueden expandirse. Esto, teóricamente, da un espacio direccionable total de hasta 1 MB (4 bancos primarios x 4 segmentos secundarios x 64 KB de tamaño) en lugar de los 256 KB totales de los cuatro bancos sin expandir. Estos segmentos secundarios también se dividen en 4 páginas de 16 KB. En la práctica, encontrar modelos de MSX-1 con tanta capacidad de memoria es poco factible pues los fabricantes, por lo general, mantuvieron los dos bancos primarios de cartuchos externos sin expandir, expuestos al usuario, y los otros dos bancos primarios para uso interno del ordenador. Los fabricantes, de los dos bancos internos, generalmente, expandieron uno de ellos para alojar memoria ROM que o bien permitiese localizar el Intérprete de BASIC en los diferentes modelos dependiendo del mercado al que fueran destinados, como es el caso del modelo *Bawareth Perfect MSX1*[18] fabricado por Daewoo y distribuido en Arabia Saudita que expande el *slot-0* para incluirlo en idioma árabe; o bien contuviese programas en memoria ROM de agenda personal y/o procesadores de texto, como el modelo *HX-22* fabricado por Toshiba, que expande el *slot-3*; o bien para incorporar programas de producción musical como el modelo *CX5MIIP*[19] fabricado por Yamaha, que expande el *slot-3*; o bien para incluir controladores de periféricos como unidades de disco en aquellos modelos que integraban una unidad, como el modelo *Expert DDPlus*[20] fabricado por Gradiente, que también expande el *slot-3*. La expansión de las ranuras externas se deja al usuario, utilizando un expansor de bancos como el Modulon[21] que le permita insertar cartuchos MegaRAM que amplíen la memoria RAM con o sin gestores de memoria (mapeadores de memoria).

La Figura 3.6[22] muestra los bancos de memoria del ordenador *Expert DDPlus* con el *slot-3* expandido para dar cabida a la controladora de la unidad de disco. Los segmentos secundarios generalmente se numeran de la siguiente manera: Slot<número del banco principal> - <número del segmento secundario>.

18 https://www.msx.org/wiki/Bawareth_Perfect_MSX1

19 https://www.msx.org/wiki/Yamaha_CX5MII

20 https://www.msx.org/wiki/Gradiente_Expert_DDPlus

21 https://supersoniqs.com/2013/07/01/supersoniqs-announces-intelligent-slot-expander-modulon/

22 Fuente: https://www.msx.org/wiki/Gradiente_Expert_DDPlus

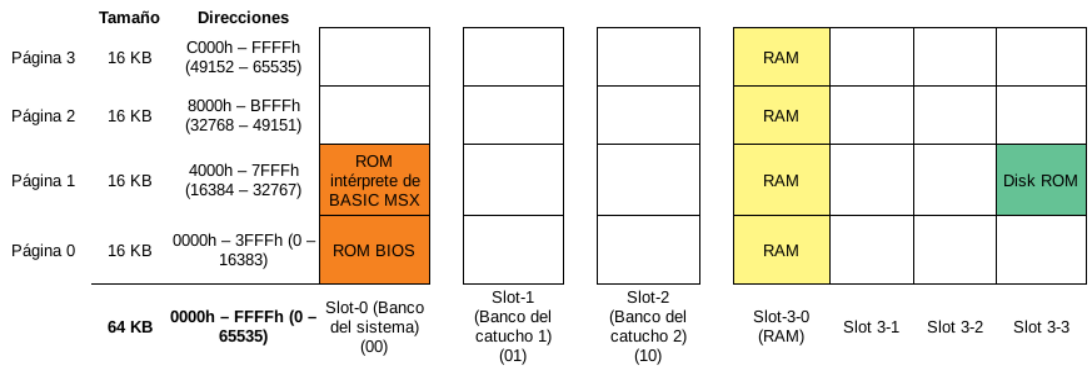

Figura 3.6: Expansión del Banco de memoria

Cuando una página lógica se asigna a una página física de un banco que está expandido ha de especificarse el segmento (*subslot*) al que pertenece. Para saber si un banco primario está expandido en segmentos, primeramente ha de seleccionarse dicho banco primario en la página 3 del *registro selector de banco* (puerto A8 de E/S) y esto es así porque el *registro de selección de segmento* es la posición de memoria FFFFh del banco primario expandido —a diferencia del *registro selector de banco* que es un puerto de E/S y recuérdese que la BIOS, para uso de variables de sistema, se reserva y ocupa los 4 KB más altos de memoria RAM—. Este *registro de selección de segmento* tiene el mismo formato que el *registro selector de banco* (véase la Fig. 3.5) e indica para cada página del banco primario expandido qué segmento (*subslot)* está activado.

Así pues, el microprocesador, al realizar la rutina BIOS de análisis de la memoria, examina por cada banco primario si está expandido en segmentos, asignando la página 3 lógica a la página 3 física del banco primario que se analiza. Acto seguido, escribe en el *registro de selección de segmento* (la dirección de memoria FFFFh) el valor F0h (11110000 en notación binaria) y lee el contenido del registro. Si el valor leído coincide con el valor escrito, significa que el banco primario no está expandido. Por contra, si al leer el valor del registro obtiene el valor escrito en *complemento a uno* (0Fh), significa que el banco primario sí está expandido, entonces analiza todos los segmentos, desde el 0 al 3, antes de proceder con el análisis del siguiente banco. La BIOS almacena el resultado de la comprobación de la expansión de los bancos primarios en la memoria RAM reservada para el sistema. En las direcciones de memoria FCC1h para el *slot-0,* FCC2h para el *slot-1,* FCC3h para el *slot-2* y FCC4h para el *slot-3* almacena si el banco está expandido o no —si el *bit* de mayor peso, es decir, el *bit* número 7 del dato está a '0', indicará que el banco primario no está expandido, si está a '1' indicará que sí lo está —; y en las direcciones FCC5h para el *slot-0,* FCC6h para el *slot-1,* FCC7h para el *slot-2* y FCC8h para el *slot-3* el valor de la configuración de sus *subslots*, en caso de existir. Después guardada en el *registro de selección de segmento* la configuración resultante del análisis de las páginas de los segmentos del banco primario expandido que contiene la memoria RAM.

Por ejemplo, el mapa de memoria del ordenador *Expert DDPlus* (véase la Fig. 3.6) quedaría configurado tras el análisis de memoria con los siguientes valores:

- *Registro selector de banco*: 11110000 —en notación binaria— (F0h). O lo que es lo mismo, las páginas lógicas 0 y 1 asignadas al BANCO DEL SISTEMA (*slot-0*) y las páginas lógicas 2 y 3 en el banco primario número 3 (*slot-3*).

- El *bit* de mayor peso en FCC1h, FCC2h y FCC3h puesto a '0' ya que no están expandidos.
- El *bit* de mayor peso en FCC4h puesto a '1' ya que sí está expandido.
- Valor guardado en la dirección FCC8h: 00001100 —en notación binaria— (0Ch). O lo que es lo mismo, la página 0 del banco en el *subslot 3-0*, la página 1 del banco en el *subslot* 3-3 —la memoria ROM prima sobre la RAM— y las páginas 2 y 3 del banco en el *subslot 3-0*. El valor de las direcciones FCC5h, FCC6h y FCC7h es indiferente pues su bancos no están expandidos.
- *Registro de selección de segmento*: el valor existente en la dirección FCC8h.

Dado que que hubo múltiples fabricantes que lanzaron diversos modelos de ordenador MSX con diversas configuraciones de memoria y con el fin de evitar incompatibilidades entre ordenadores con los desarrollos que se lleven a cabo, es importante realizar la gestión de memoria (*slots*/*subslots*), asegurarse de la correcta ubicación de la memoria RAM —sobre todo para proyectos que requieran más de 32 KB — pues en caso contrario, además de desarrollar programas mal escritos, se puede producir el reinicio del ordenador pues donde se asume que existe una página física de memoria RAM puede haber una de memoria ROM o no existir.

Afortunadamente existe un algoritmo generalizado que permite fijar la memoria RAM en las paginas del *registro de selección de segmento* del banco primario expandido donde reside la RAM. Este algoritmo es conocido como POKE UNIVERSAL y sus operaciones consisten en:

1) Leer el valor del *registro de selección de segmento*: *valor* ← leer (FFFFh).
2) Complementar en uno el valor leído ya que es un banco expandido y está en C1: *valor* ← C1 (*valor)*.
3) Realizar la división entera entre 16 al *valor* para quedarse con el *nibble* alto, que indica la página de la memoria RAM: *valor* ← *valor* / 16.
4) Multiplicar por 17 el *valor* para copiar el *nibble* alto en el *nibble* bajo: *valor* ← *valor* * 17.
5) Escribir el nuevo *valor* en el *Registro de selección de segmento*: FFFFh ← *valor*.

Con el algoritmo de POKE UNIVERSAL se asegura que cuando se configure el *registro selector de banco* para que el Z80 gestione las cuatro páginas lógicas con memoria RAM, todas ellas, efectivamente, estén asignadas a páginas físicas con memoria RAM.

Mención aparte merecen los gestores de memoria conocidos como *mapeadores de memoria* que amplían la memoria del sistema y que la gestionan de forma distinta pues funcionan como una capa superpuesta al mecanismo de segmentos de los bancos primarios extendidos —hacen uso de cuatro puertos de E/S asignada para complementar el bus de direcciones del Z80—. Estos gestores de memoria, aunque aparecieron con la segunda revisión de la norma estándar del MSX (MSX-2), existen en el mercado[23] cartuchos que los incluyen y pueden ser conectados a ordenadores MSX-1.

23 Ver los cartuchos *mapeadores de memoria* existentes en el mercado: https://www.msx.org/wiki/Category:RAM_Expansions#Memory_Mapper_expansion_cartridges

Existe el inconveniente de que la BIOS de los MSX-1 no puede inicializar el *mapeador* limitando la memoria del sistema a 64 KB de tamaño a no ser que el propio cartucho o incluya código que lo inicialice o incluya un sistema operativo de disco (MSX-DOS) que lo haga, como por ejemplo el NESTOR[24]. Explicar el funcionamiento de estos *mapeadores de memoria* se sale de los objetivos del presente trabajo.

3.3.2 ROM BIOS (Basic Input/Output System)

La BIOS del MSX supone una capa de abstracción de *hardware* (HAL) que posibilita la compatibilidad entre los distintos modelos de ordenadores MSX pues oculta el *hardware* real que hay en la capa inferior ofreciendo al programador una API con rutinas (funciones) de acceso a los componentes de la máquina y/o periféricos conectados sin necesidad que el programador conozca los detalles específicos de cada uno de ellos.

El uso de las rutinas que proporciona la BIOS permite que el programa desarrollado sea robusto y compatible. Los resultados de las rutinas o bien se almacenan en la memoria RAM como variables del sistema o bien quedan guardados en los registros del Z80. La BIOS del MSX-1 ofrece un total de 159 rutinas entre las cuales se encuentran:

- Rutinas de E/S.
- Rutinas de acceso al procesador de vídeo (VDP).
- Rutinas de acceso al procesador de sonido (PSG).
- Rutinas de E/S para mandos de juegos (*joysticks*).
- Rutinas de acceso a la unidad de disco (en aquellos ordenadores que dispongan de una unidad de disco).

Para hacer uso de las rutinas de la BIOS o llamadas al sistema, se ha de tener un conocimiento básico del lenguaje ensamblador del Z80 para poder realizar la llamada a la rutina y el paso de parámetros que esta espera. Para profundizar con detalle en las rutinas que ofrece la BIOS y los parámetros de entrada/salida necesarios, se recomienda la consulta de los libros [9][12] o la página web MSX Assembly Page[27]

Dado que la BIOS del MSX es una pieza importante del estándar MSX se recomienda desactivarla lo menos posible, salvo necesidad de uso de memoria RAM en la página lógica 0 y teniéndose presente que se deberán programar directamente las funciones de acceso al *hardware*.

24 NESTOR DOS: https://www.konamiman.com/msx/msx-e.html#nextor

3.3.3 Mapa de Entrada/Salida

El microprocesador Z80 maneja, además del espacio de direccionamiento de la memoria, otras 256 direcciones para comunicarse con el resto de componentes del sistema (procesador de vídeo, procesador de sonido, interfaz periférico programable, impresora, cartuchos...). Estas 256 direcciones se denominan puertos de E/S, son de 8-bits y se corresponden con los registros de dichos componentes.

La Figura 3.7[25] muestra la asignación de los puertos con los correspondientes componentes que define la norma para los MSX de primera generación. En gris se ha pintado aquellos puertos que se dejaron reservados para futuras revisiones. En la página web de MSX Assemby Page[26] puede consultarse el mapa de E/S con la asignación de puertos incluidas en revisiones posteriores o con la aparición de nuevos periféricos.

Puerto (rango)	
00-7F	
80-87	RS-232C
88-8F	
90-93	Impresora
94-97	
98-9B	VDP
A0-A3	PSG
A4-A7	
A8-AB	PPI
AC-AF	
B0-B3	Memoria externa
B4-B7	
B8-BB	Lápiz óptico
C0-CF	
D0-D7	Controlador de disco
D8-FF	

Figura 3.7: Mapa E/S del MSX-1

25 Fuente: [18]

26 MSX Assembly Page, mapa de E/S: http://map.grauw.nl/resources/msx_io_ports.php

3.4 . Procesador de video (VDP)

Es el circuito integrado encargado de controlar la pantalla de video (VDP, acrónimo de *Video Display Processor*). La norma especifica que para la primera generación de MSX, el VDP a usar es el TMS9918A fabricado por Texas Instruments u otro equivalente. La frecuencia de reloj para su funcionamiento en ordenadores destinados al mercado europeo es de 50 Hz mientras que en ordenadores destinados al mercado asiático es de 60 Hz. Es el encargado de generar la señal de interrupción cada 1/50 segundos —50 veces por segundo, en Europa— para que la CPU realice varias tareas, entre ellas, la lectura del teclado.

Controla, además, 16 KB de memoria RAM propia, llamada VRAM (Video RAM) donde se almacena toda la información que el VDP requiere para mostrar la pantalla: texto, colores, *sprites*, fondo...

Dispone de una paleta de 16 colores (véase la Figura 3.8)[27].

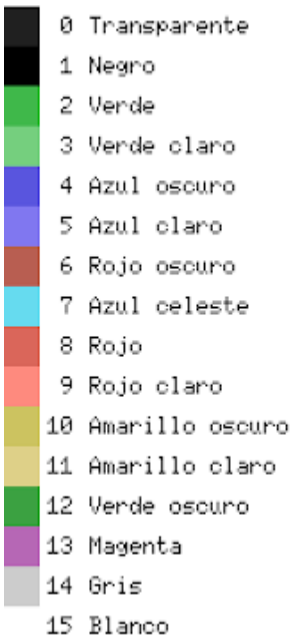

Figura 3.8: Paleta de colores del MSX-1

27 Fuente: https://aorante.blogspot.com/2011/10/paleta-vdp-de-los-msx-1.html

3.4.1. Los modos de pantalla (*Screen*)

Proporciona una resolución horizontal de 256 puntos o *píxeles* por 192 puntos o *píxeles* de resolución vertical, siendo capaz de organizarlos en cuatro modos de pantalla:

- Modo 0 ó *Screen* 0: modo texto de 24 líneas de 40 (40 x 24) caracteres[28] de 6 x 8 *píxeles*, con disponibilidad simultánea de 2 colores. No permite el uso de *sprites*.
- Modo 1 ó *Screen* 1: modo texto mixto de 24 líneas de 32 (32 × 24) caracteres de 8 x 8 *píxeles*, con disponibilidad simultánea de 2 colores. Permite el uso de *sprites* y caracteres coloridos personalizados.
- Modo 2 ó *Screen* 2: modo gráfico de alta resolución de 256 × 192 *píxeles* (24 x 32 caracteres de 8 x 8 *píxeles*), con disponibilidad simultánea de 16 colores. Permite el uso de *sprites*.
- Modo 3 ó *Screen* 3: modo gráfico de baja resolución de 64 × 48 bloques de 4 × 4 *píxeles*. Con disponibilidad simultánea de 16 colores. Permite el uso de *sprites*.

3.4.2. Operaciones de lectura/escritura

El VDP es dirigido por la CPU, quien se comunica con él de manera asíncrona a través de dos puertos del mapa de E/S: los puertos 98h (puerto de datos del VDP) y 99h (puerto de control del VDP), realizando cuatro tipos de transferencia de información posibles:

1. Escritura en los registros: la CPU envía bytes a los registros del VDP. Se necesita de dos transferencias de datos. En la primera, el dato que se envía es del byte a escribir y la segunda transferencia, el dato determina el registro destino —con el *bit* más significativo puesto a '1' para evitar que el número de registro se confunda con una dirección de la VRAM (ej. 10000RRR, siendo RRR el número del registro)—. Para realizar una operación de escritura en los registros, se ha de leer previamente el *registro de estado* del VDP para que se reinicien los dispositivos lógicos de enlace.
2. Lectura de los registros: La CPU lee bytes de los registros del VDP. Realmente sólo puede realizarse la operación de lectura sobre el *registro de estado* del VDP. A diferencia de la operación de escritura en registros, la lectura sólo requiere de una única transferencia.
3. Escritura en la VRAM: La CPU envía bytes a RAM de vídeo. Se utiliza un *registro de direcciones* interno de 14 bits que se incrementa automáticamente. La transferencia de los dos primeros bytes a escribir en la VRAM inicializan este registro. El resto de bytes a escribir se envían de manera secuencial dado que el

28 Un caracter es un bloque de 8x8 *píxeles*.

registro de direcciones ya está inicializado y se incrementa automáticamente tras la operación de escritura.

4. Lectura de la VRAM: La CPU lee bytes de la RAM de vídeo. Esta operación es similar a la de escritura en la VRAM. Se utiliza el *registro de direcciones* de incremento automático.

Como el VDP ha de encargarse del refresco de la pantalla o de la VRAM, las operaciones de lectura/escritura sobre la VRAM sólo pueden realizarse cuando el VDP no está ocupado, por lo que el tiempo necesario para que la CPU transfiera un byte de datos a la VRAM varía entre 2 y 8 microsegundos.

3.4.3. Memoria de vídeo VRAM

La VRAM está dividida en hasta 5 tablas que desempeñan una única función y, que dependiendo del modo de pantalla, se distribuyen de forma distinta. Las direcciones de inicio de cada tabla se almacenan en los registros del VDP. La Tabla 3.1 contiene las direcciones base iniciales para cada tabla en cada modo de pantalla.

La TABLA DE NOMBRES indica al VDP qué imagen debe aparecer en cierta zona de la pantalla. Una entrada de esta tabla es un número entre 0 y 255 que sirve como índice apuntador de la TABLA DE GENERACIÓN DE PATRONES y de la TABLA DE COLORES.

La TABLA GENERADORA DE PATRONES define la imagen a colocar en la pantalla para un valor dado de la tabla de nombres.

La TABLA DE COLORES informa al VDP de los colores a usar en ese modo de pantalla para cada entrada de la TABLA DE NOMBRES. No todos los modos de pantalla disponen de esta tabla.

La TABLA GENERADORA DE SPRITES contiene las definiciones de los *sprites* disponibles. Tiene un tamaño de 2 KB divididos en 256 bloques (1 bloque = 1 *sprite*) de 8 bytes (que definen el *sprite).*

La TABLA DE ATRIBUTOS DE SPRITES contiene la información de los colores de los *sprites* diseñados, su ubicación en la pantalla y el número identificativo del *sprite*. Tiene un tamaño de 128 bytes que se dividen en 32 subtablas —una subtabla por cada uno de los 32 planos disponibles— de 4 bytes de largo. Cada subtabla se organiza según la Figura 3.9[29].

Tabla	Modo 0	Modo 1	Modo 2	Modo 3
Tabla de nombres	0000h	1800h	1800h	0800h
Tabla de color	-	2000h	2000h	-
Tabla generadora de patrones	0800h	0000h	0000h	0000h
Tabla de atributos de *sprites*	-	1B00h	1B00h	1B00h
Tabla generadora de *sprites*	-	3800h	3800h	3800h

Tabla 3.1: Direcciones base de las tablas de la VRAM

29 Fuente: [7], página 147

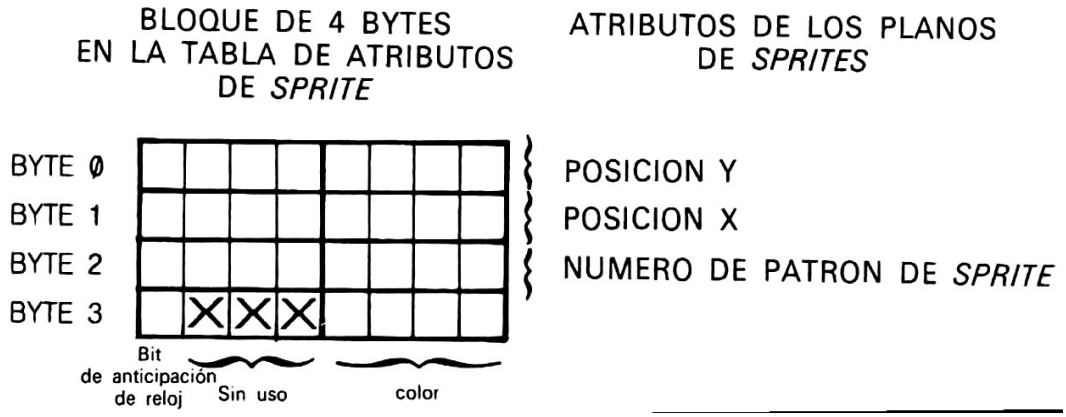

Figura 3.9: Organización de la Tabla de Atributos de Sprites

La primera subtabla —los 4 primeros bytes de la TABLA DE ATRIBUTOS DE SPRITES— corresponde a la información del *sprite* que se dibujará en el plano 0, la segunda subtabla, al *sprite* que se dibujará en el plano 1, y así con el resto de 32 *sprites* y planos. Los dos primeros bytes de la subtabla definen las coordenadas de la pantalla donde se posicionará el *sprite*. El tercer byte es el número de patrón de *sprite*, es decir, el número del *sprite* de la TABLA GENERADORA DE SPRITES a visualizar. El cuarto byte define el color del *sprite* y el *Bit de anticipación de reloj*, que si su valor es '0', la esquina superior izquierda del *sprite* se colocará en las coordenadas de la pantalla X, Y dadas, en caso de valer '1', la esquina superior izquierda del *sprite* se colocará en la coordenada X-32, Y.

3.4.4. Acceso rápido a la VRAM

Como el Z80 no tiene acceso directo a la VRAM y, por tanto a las tablas, debe generarlas previamente en memoria principal y transferirlas al VDP a través de los puertos de datos y control (98h y 99h de E/S) durante el periodo de análisis de la pantalla ya que el VDP no genera ninguna imagen en ese momento. Esto tiene el inconveniente de ser lento.

Para salvar esta lentitud, ha de hacerse uso del *registro de direcciones* interno del VDP y acceder a la VRAM secuencialmente, ya que así sólo se precisa de una sola transferencia de datos en lugar de dos (se evita la transferencia de inicialización de dicho registro, que indica la dirección de la VRAM a la que se quiere acceder).

Por ello, si se va a manipular una gran cantidad de datos de la VRAM, debe mantenerse una copia de los datos de la VRAM a modificar en la RAM, realizar las modificaciones sobre esta copia para después enviarla secuencialmente a la VRAM. Por ejemplo, en el caso de desplazamiento de pantallas (*scroll*), es ventajoso leer la pantalla completa (la TABLA DE NOMBRES), cargarla en una área de la RAM para manipularla

y después volcarla en la VRAM. De esta forma, un desplazamiento de pantalla puede realizarse en menos de 1/50 segundos, mientras que si el acceso a VRAM se hace de manera no secuencial, el tiempo necesario será superior a 3/50 segundos.

3.4.5. Los registros del VDP

El VDP tiene 8 registros de sólo escritura (registros 0..7) y un sólo registro de lectura, el *registro de estado* (registro 8). Estos registros definen los diferentes parámetros y la direcciones base para los distintos bloques en los que se divide la VRAM. La BIOS almacena una copia del valor inicial de los registros de escritura en la memoria RAM reservada para el sistema (variables de sistema RG0SAV - RG7SAV) para que puedan ser leídos. La posiciones de memoria RAM van de la F3DFh (RG0SAV) hasta la F3E6h (RG7SAV). Es conveniente actualizar estas posiciones cada vez que se modifique el valor de los registros de escritura.

La Tabla 3.2 muestra los registros del VDP con sus bits de control.

	7	6	5	4	3	2	1	0
REGISTRO 0	0	0	0	0	0	0	M3	VE
REGISTRO 1	VRAM	BL	IE	M1	M2	0	SIZE	MAG
REGISTRO 2	0	0	0	0	TABLA DE NOMBRES			
REGISTRO 3	TABLA DE COLOR							
REGISTRO 4	0	0	0	0	0	TABLA GENERADOR DE PATRONES		
REGISTRO 5	0	TABLA DE ATRIBUTOS DE SPRITES						
REGISTRO 6	0	0	0	0	0	TABLA GENERADOR DE SPRITES		
REGISTRO 7	COLOR TINTA				COLOR FONDO			
REGISTRO 8	INTERRUP-CIONES	5 SPRITES	COLISIÓN SPRITES	NÚMERO DEL 5º SPRITE				

Tabla 3.2: Los registros del VDP

- **Registro 0**: en los ordenadores MSX de primera generación sólo los dos bits menos significativos (M3 y VE) de este registro son utilizados. El resto de bits están reservados para revisiones futuras y deben, por tanto, estar puestos a '0'. VE (*bit* 0) está preparado para la conexión de un VDP externo y M3 (*bit* 1) está implicado en la selección del modo de pantalla conjuntamente con registro 1.
- **Registro 1**: Registro de control principal del VDP. El *bit* 2 está reservado y tiene que estar siempre a '0'. El resto de bits se usan para:
  - MAG (*bit* 0): define la extensión de los *sprites*: nomal —resolución de 1 x 1 puntos— (MAG=0) o ampliado —resolución de 2 x 2 puntos— (MAG=1).

- SIZE (*bit* 1): define el tamaño de los *sprites*: 8 x 8 *píxeles* (SIZE=0) ó 16 x 16 *píxeles* (SIZE=1).

  La Tabla 3.3 muestra como MAG y SIZE se combinan para obtener los distintos tipos de *sprites*.

SIZE	MAG	
0	0	*sprite* de 8x8 sin ampliar
0	1	*sprite* de 8x8 ampliado
1	0	*sprite* de 16x16 sin ampliar
1	1	*sprite* de 16x16 ampliado

Tabla 3.3: Bits de control de sprites

- M2 (*bit* 3): segundo *bit* del modo de pantalla.
- M1 (*bit* 4): primer *bit* del modo de pantalla.

La Tabla 3.4 muestra qué combinación de M1, M2 y M3 proporciona cada modo de pantalla.

M1	M2	M3	Modo de pantalla
0	0	0	Modo 1 (texto)
0	0	1	Modo 2 (gráficos alta resol.)
0	1	0	Modo 3 (gráficos multicolor)
1	0	0	Modo 0 (texto)

Tabla 3.4: Bits de control del modo de pantalla

- IE (*bit* 5): permite activar (IE=1) o desactivar (IE=0) las interrupciones del VDP. Por defecto, al iniciarse el sistema, se activan las interrupciones.
- BL (*bit* 6): permite activar (BL=1) o desactivar la pantalla (BL=0).
- VRAM (*bit* 7): indica al VDP el tamaño de memoria de los *chips* que se usan para la VRAM. En general, para MSX son de tamaño de 16 KB (VRAM=1).

- **Registro 2**: Ayuda a proporcionar la dirección base de la VRAM donde comienza la TABLA DE NOMBRES para un determinado modo de pantalla. Para ello, se usa su *nibble* bajo (siendo su rango de posibles valores de 0 a 15). Su valor ha de multiplicarse por 400h ya que este *nibble* bajo son los 4 bits más significativos de los 14 de la dirección de la TABLA DE NOMBRES. La Tabla 3.5 muestra los valores por defecto de este registro.

	Binario				Decimal
Screen 0	0	0	0	0	0
Screen 1	0	1	1	0	6
Screen 2	0	1	1	0	6
Screen 3	0	0	1	0	2

Tabla 3.5: Valores por defecto del registro 2 del VDP

- **Registro 3**: Ayuda a proporcionar la dirección base de la VRAM donde comienza la TABLA DE COLORES para un determinado modo de pantalla. Este registro contiene un valor entre 0 y 255 que debe multiplicarse por 40h ya que representan los 8 bits más significativos de la dirección de la TABLA DE COLORES. La Tabla 3.6 muestra los valores por defecto de este registro.
- **Registro 4**: Define la dirección de comienzo en VRAM de la TABLA GENERADORA DE PATRONES para un modo de pantalla dado. Los tres bits menos significativos de este registro (por tanto, su valor oscilará entre 0 y 7) representan los tres bits más significativos de la dirección de la TABLA GENERADORA DE PATRONES. Para conseguir esta dirección, el contenido del registro ha de multiplicarse por 800h. La Tabla 3.6 muestra los valores por defecto de este registro.
- **Registro 5**: Define la dirección de comienzo en VRAM de la TABLA DE ATRIBUTOS DE SPRITES. Se usan los siete bits menos significativos de este registro (por tanto, su valor oscilará entre 0 y 127) representan los siete bits más significativos de los 14 de la dirección de la TABLA DE ATRIBUTOS DE SPRITES. Para conseguir esta dirección, el contenido del registro ha de multiplicarse por 80h. La Tabla 3.6 muestra los valores por defecto de este registro.
- **Registro 6**: Define la dirección de comienzo en VRAM de la TABLA GENERADORA DE SPRITES. Se usan los tres bits menos significativos de este registro (por tanto, su valor oscilará entre 0 y 7). Representan los tres bits más significativos de los 14 de la dirección de la TABLA GENERADORA DE SPRITES. Para conseguir esta dirección, el contenido del registro ha de multiplicarse por 800h. La Tabla 3.6 muestra los valores por defecto de este registro.

	Screen 0	*Screen 1*	*Screen 2*	*Screen 3*
Registro 3	0	128	255	0
Registro 4	1	0	3	0
Registro 5	0	54	54	54
Registro 6	0	7	7	7

Tabla 3.6: Valores por defecto de los registros 3-6 del VDP

- **Registro 7**: Este registro define el color de la tinta para el modo 0 de pantalla y el color de fondo para todos los modos. Su *nibble* alto marca el color de la tinta y su *nibble* bajo el del fondo.
- **Registro 8**: Corresponde con el *registro de estado* y es de sólo lectura. Su lectura sin sincronía con la interrupción de barrido de la pantalla puede hacer que se salten algunas interrupciones. Por ello, es preferible leer la copia de este registro desde la variable de sistema.

- INTERRUPCIONES (*bit* 7): sirve para controlar las interrupciones. Se activa a nivel alto (a '1') cuando el VDP termina el análisis de la pantalla (*overscan*). Se pone a '0' después de la lectura del registro.
- 5º SPRITE (*bit* 6): activado a nivel alto indica que se rompe la regla del 5º *sprite*, es decir, indica la presencia de más de 4 *sprites* en una misma línea.
- COLISIÓN DE SPRITES (*bit* 5): se activa a '1' cada vez que el VDP detecta la colisión de dos o más *sprites* al analizar la pantalla. Este bit vuelve a ser '0' tras la lectura del registro. No se tienen en cuenta, para la detección de colisiones, aquellos *sprites* que se encuentran fuera del puntero que señala el fin de la TABLA DE ATRIBUTOS DE SPRITES (Doh).
- NÚMERO DEL 5º SPRITE (*bits* 0-4): indica el número del *sprite* —el número del plano de *sprite*, para ser más exactos— que rompe la regla del 5º *sprite*.

3.4.6. Sprites

El VDP del MSX tiene la capacidad de mostrar en pantalla hasta 32 *sprites* por *hardware* de forma simultánea. Estos *sprites* son un tipo de gráfico basado en caracteres o patrones que, además de definírseles la forma, pueden ser movidos por la pantalla. Tienen una serie de cualidades que los hace muy útiles a la hora de realizar videojuegos:

- Admiten transparencias en su diseño, lo que les permite superponerse a cualquier gráfico sin borrar los caracteres de la pantalla sobre el que se sobrepone.
- Pueden aparecer o desaparecer en cualquier parte de la pantalla pudiéndose detectar la colisión de dos o más *sprites* en su recorrido por la pantalla.
- Pueden solaparse o unirse para construir un *sprite* de mayor tamaño que los caracteres que componen el fondo de la pantalla.

La limitación de hasta 32 *sprites* simultáneos en pantalla se debe a que la pantalla se organiza en una serie de planos superpuestos, donde cada plano sólo puede contener un único *sprite*. Cada plano tiene una prioridad asignada que marca el orden de preferencia a la hora de visualizarse superpuesto al resto de planos. Treinta y dos de estos planos corresponden a los planos de *sprite*, numerados desde el 0 al 31 y que se superponen al plano multicolor y al de fondo. El número del plano marca su prioridad, siendo el plano número 0 el más prioritario y más alejado del plano de fondo; y el plano número 31 el plano menos prioritario y más próximo al plano de fondo. El plano multicolor contiene los gráficos de caracteres que componen el fondo y el texto. El plano de fondo define el marco o borde de la pantalla (véase la Figura 3.10)[30].

30 Fuente: [7], página 120.

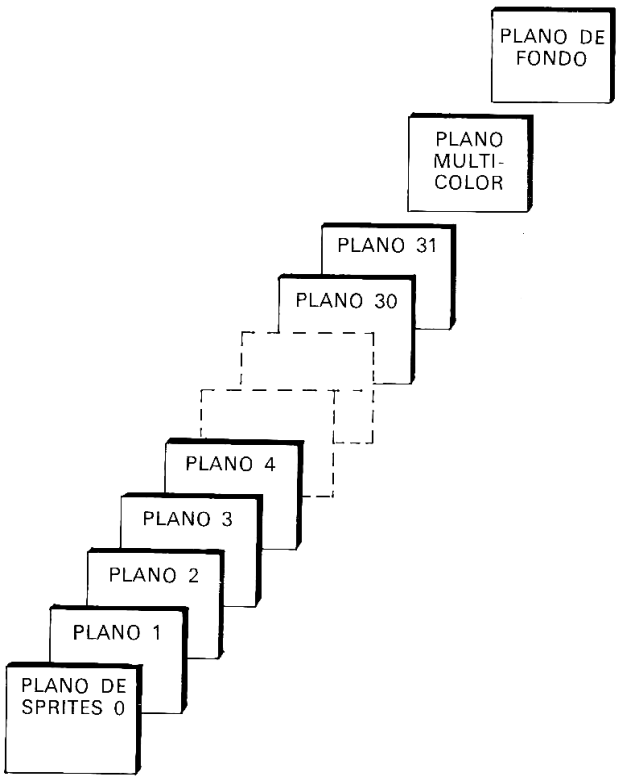

Figura 3.10: Planos de sprite y de fondo

Existen hasta cuatro tamaños distintos de *sprites*: 8 x 8 *píxeles*, 8 x 8 *píxeles* ampliados, 16 x 16 *píxeles* y 16 x 16 *píxeles* ampliados. El *sprite* de tamaño 8 x 8 *píxeles* tiene el mismo tamaño que los caracteres o patrones y una resolución de 1 x 1 punto. Los tamaños ampliados tienen una resolución de 2 x 2 puntos. Así pues, un *sprite* con tamaño 8 x 8 ampliado tiene un tamaño final de 16 x 16 y un *sprite* 16 x 16 ampliado, uno de 32 x 32. El tamaño de los *sprites* se controla a través del registro 1 del VDP. Existe una limitación respecto al visualizado de *sprites* de diferentes tamaños en pantalla y es que, simultáneamente, sólo puede mostrarse en pantalla *sprites* del mismo tamaño.

Existe una segunda limitación relativa a los *sprites* y es que sólo admiten un único color para todo el *sprite*.

Como se ha visto en el punto *3.4.3 Memoria de vídeo VRAM*, las tablas de la VRAM que almacenan la información relativa a los *sprites* son la TABLA GENERADORA DE SPRITES y la TABLA DE ATRIBUTOS DE SPRITES.

DEFINICIÓN DE SPRITES

La base para definir un *sprite* es una matriz de 8 x 8 puntos o *píxeles* (véase la Figura 3.11)[31], donde cada punto marcado indica que este debe ser dibujado. En caso contrario, ofrecerá transparencia sobre los planos menos prioritarios al que ocupa el *sprite*. Las columnas de la matriz marcan el valor, en potencia de 2, del punto a dibujar.

31 FUENTE: [7], página 123.

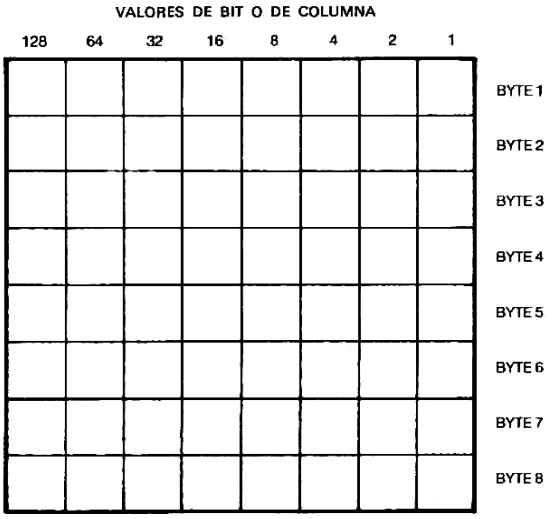

Figura 3.11: Matriz de sprites 8 x 8

Los valores de los bytes que componen el *sprite* se obtienen sumando los valores de las columnas de los puntos a dibujar en cada fila de la matriz. La Figura 3.12[32] muestra un ejemplo de la definición de un *sprite* de un hombre de 8 x 8 *píxeles* con los correspondientes valores para los bytes que lo componen.

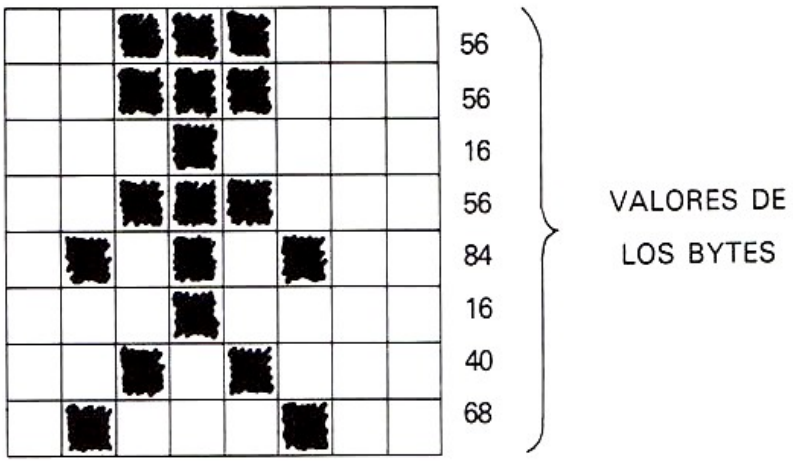

Figura 3.12: Ejemplo de definición de un sprite

Como se ha comentado, cada *sprite* sólo puede ser de un único color. En caso de querer disponer de un *sprite* multicolor, este puede componerse por la fusión de dos o más *sprites*. Por ejemplo, para realizar un *sprite* de un hombre, este podría componerse por los siguientes tres *sprites*:

- Un *sprite* para el pelo, torso y brazos—jersey— (*sprite* 0)
- Un *sprite* para cara y manos (*sprite* 1)
- Un *sprite* para piernas y pies (*sprite* 2)

32 FUENTE: [7], página 125.

A la hora de dibujar el *sprite* del hombre, habría de dibujarse en pantalla los tres *sprites* que los componen.

Los *sprites* de 16 x 16 puntos de tamaño se componen de cuatro *sprites* de 8 x 8 puntos de tamaño. A cada *sprite* de 8 x 8 que lo componen se le denomina cuadrante. Por tanto, un *sprite* de 16 x 16 puede verse como una matriz de 16 x 16 compuesta por 4 submatrices o cuadrantes de 8 x 8. Se necesitan 32 bytes para almacenar un *sprite* de 16 x 16 puntos que son escritos como una entrada en la tabla generadora de *sprites*. La Figura 3.13[33] ilustra como se organizan los cuadrantes de un *sprite* de 16 x 16 puntos y la Figura 3.14[34] cómo se almacenan en memoria los 32 bytes que lo componen.

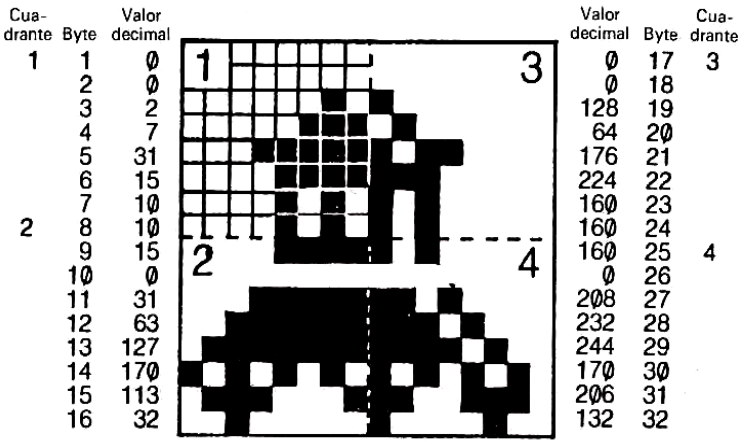

Figura 3.13: Definición de un sprite de 16x16

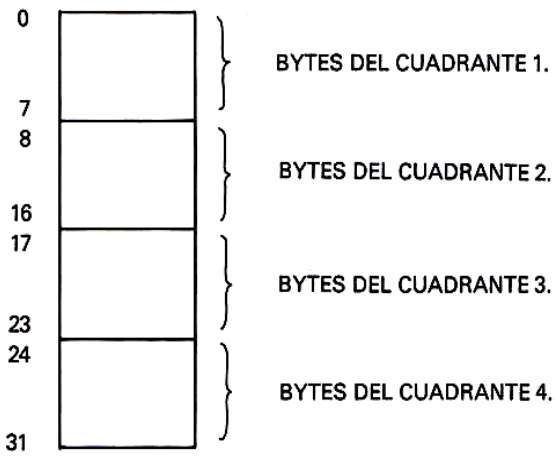

Figura 3.14: Relación byte-cuadrante de un sprite 16x16

33 FUENTE: [7], página 125.

34 FUENTE: [8], página 179.

LA REGLA DEL QUINTO SPRITE

El VDP del MSX-1 tiene la limitación de sólo poder mostrar hasta 4 *sprites* simultáneos en una misma línea de la pantalla. A esta limitación se la conoce como *la regla del quinto sprite*. Si esta regla no se respeta, sólo se verán en una misma línea de la pantalla los 4 *sprites* de mayor prioridad. Así mismo, el bit 6 del *registro de estado* del VDP (registro 8) —la bandera del quinto *sprite*— se activará a '1' y los cinco bits de menor peso de dicho registro indicarán el número del quinto *sprite* —del plano, para ser exactos— que incumple la regla (véase la *Tabla 3.2: Los registros del VDP*).

Esta regla ha de tenerse presente a la hora de definir *sprites* multicolor pues si no se definen con cautela, *la regla del quinto sprite* se incumplirá rápidamente. Por ejemplo, si se está definiendo la cabeza del héroe del videojuego y se usan dos *sprites* para su composición: uno para el pelo y ojos, y otro para la cara; la cabeza, al ser la superposición de ambos *sprites*, en la línea de la pantalla donde coincidan pelo y cara o cara y ojo ya habrán dos *sprites*. Si al moverse el héroe, la línea de su cabeza donde coincide pelo y cara, también coincide en la misma línea con algunos enemigos que también tengan en su composición coincidencia de más de un *sprite* en la misma línea, *la regla del quinto sprite* se incumplirá fácilmente y la línea del *sprite* que incumple la regla no será dibujada.

MOVER SPRITES

Para mover un *sprite* basta con modificar los valores de las coordenadas vertical (*y*) y horizontal (*x*) de su posición en pantalla que se hallan almacenados en su subtabla de la TABLA DE ATRIBUTOS DE SPRITES (véase la Figura 3.9 *Organización de la Tabla de Atributos de Sprites,* del punto *3.4.3 Memoria de vídeo VRAM*).

El primer byte (byte 0) de la subtabla corresponde con su coordenada *y* de la pantalla. Los valores de *y* para mostrar el *sprite* en pantalla oscilan dentro del rango [-31, 191]. Los valores negativos significan que el *sprite* está fuera de la pantalla, así pues, si los valores de *y* varían desde -31 hasta 0, el *sprite* aparecerá lentamente por la parte superior de la pantalla hacia abajo. Existen otros dos valores significativos para la coordenada *y*. Cuando el byte *0* toma el valor de 208, el propio *sprite* y todos los demás que ocupan planos con prioridad inferior al suyo, desaparecen de la pantalla. Por contra, cuando toma el valor de 209, sólo el propio *sprite* es el que desaparece. Estos dos valores para la coordenada *y* permiten esconder *sprites*.

El segundo byte (byte 1) de la subtabla corresponde con la coordenada *x* del *sprite*. El rango de valores que puede tomar esta coordenada para que el *sprite* se mueva por la pantalla es [0, 255]. Para que este desaparezca horizontalmente de la pantalla, bastará con dar el valor -32 a la coordenada *x*.

Para mover el *sprite* por pantalla hay que tener presente:

- El punto origen de la pantalla (coordenadas *x*=0, *y*=0) se corresponde con la esquina superior izquierda.
- Las coordenadas vertical (x) y horizontal (y) de un *sprite* toman como posición origen el *pixel* superior izquierdo del mismo.

- De inicio, se fija la coordenada vertical de los *sprites* con el valor 209 para evitar que aparezcan en pantalla. La coordenada horizontal se fija previamente al valor 255 (el valor más alto posible).

DETECCIÓN DE COLISIÓN DE SPRITES

El VDP puede detectar colisión de *sprites* por *hardware*. Si un *pixel* de dos *sprites* se superponen, aunque sean transparentes, se considera que ha habido una colisión. En caso de detectarse una colisión, el bit *5* del *registro de estado* del VDP (registro 8) se activará a '1'. El VDP actualiza el estado de este bit cada 0,02 segundos.

Existe el inconveniente de que el VDP no es capaz de informar sobre qué *sprites* han colisionado ni en qué coordenadas se ha producido la colisión. Esto implica que debe realizarse una función por *software* que lo detecte. Para no caer en un bucle infinito que llame a dicha función mientras se está tratando una colisión, es recomendable deshabilitar la detección de colisiones y volverla a habilitar al salir del tratamiento de la colisión.

3.4.7. Modo 2 de pantalla (gráficos en alta resolución)

Este modo de pantalla, al disponer de gráficos en alta resolución, es el ideal para la realización de videojuegos. Por ello, se realizará un análisis profundo de las características de este modo de pantalla.

La principal característica es la dimensión de la pantalla del modo gráfico 2 cuyo tamaño es de 256 *píxeles* de ancho y 192 *píxeles* de alto y permite el uso de *sprites*. La segunda característica a tener en cuenta es la paleta de colores, formada por un total de 16 colores (15 colores + transparencia). La tercera característica a tener en cuenta y principal limitación existente a la hora de pintar en el modo gráfico 2 es que sólo podemos utilizar un máximo de 2 colores —uno para la tinta y otro para el fondo— por cada bloque horizontal de 8 *píxeles* (byte), debido al efecto denominado *attribute clash*[35].

Los valores iniciales de los registros del VDP para el modo 2 de pantalla están representados en la Tabla 3.7.

35 https://en.wikipedia.org/wiki/Attribute_clash

	Binario									
	7	6	5	4	3	2	1	0	Hexadecimal	Decimal
REGISTRO 0	0	0	0	0	0	0	1	0	02h	2
REGISTRO 1	1	1	1	0	0	0	0	0	E0h	224
REGISTRO 2	0	0	0	0	0	1	1	0	06h	6
REGISTRO 3	1	1	1	1	1	1	1	1	Ffh	255
REGISTRO 4	0	0	0	0	0	0	1	1	03h	3
REGISTRO 5	0	0	1	1	0	1	1	0	36h	54
REGISTRO 6	0	0	0	0	0	1	1	1	07h	7
REGISTRO 7	0	0	0	0	0	1	1	1	07h	7

Tabla 3.7: Modo 2: Valores iniciales de los registros del VDP

La pantalla de 256 × 192 *píxeles* se organiza en una matriz (matriz pantalla) de 32 x 24 caracteres o *tiles* —un total de 768 *tiles*— de 8 x 8 *píxeles* cada uno. Esto da un tamaño de memoria, para la matriz pantalla, de 6 KB (768 x 8 bytes = 6144 bytes). Así mismo, la matriz pantalla se subdivide en tres bancos o áreas de 32 x 8 caracteres (un total de 256 caracteres por banco): banco 0, banco 1 y banco 2.

TABLA DE NOMBRES

Su dirección de inicio en VRAM es la 1800h y ocupa un tamaño de 768 bytes. Representa el mapa-mosaico de la imagen a mostrar. Cada uno de los valores de esta tabla es el índice apuntador del carácter de la TABLA DE GENERACIÓN DE PATRONES y de la TABLA DE COLORES que se ha de mostrar en ese *tile* de la pantalla.

Por ejemplo, si el valor de la dirección 1800h —primer elemento de la TABLA DE NOMBRES— es un '6', significa que el caracter a dibujar en el *tile* cero de la pantalla (posición 0) es el que ocupa la posición 6 de la TABLA GENERADORA DE PATRONES y de la TABLA DE COLORES. Es decir, los 8 bytes comprendidos entre las posiciones 30h y 37h de estas tablas. Por contra, si el valor 256 de la TABLA DE NOMBRES es un '0', en el *tile* 256 (primer *tile* del banco 1) de la matriz pantalla, se dibujará el patrón cero del banco 1 de la TABLA GENERADORA DE PATRONES, es decir, los 8 bytes comprendidos entre las posiciones 0800h y 0807h.

TABLA GENERADORA DE PATRONES

Su dirección de inicio en VRAM es la 0000h y ocupa un tamaño de 6144 bytes. Está subdivida en tres bancos de 256 caracteres o patrones cada uno. Esto proporciona tres conjuntos de 256 patrones distintos en correspondencia con cada uno de los bancos en los que se subdivide la matriz pantalla. Es decir, los primeros 2 KB definen los patrones que aparecen para los primeros 256 valores que se pueden asignar en la TABLA DE NOMBRES y que se corresponden con los *tiles* 0..255 de la matriz pantalla. Los segundos 2 KB definen los patrones para los valores comprendidos entre 256 y 511 de la TABLA DE NOMBRES y que se corresponden con los *tiles* 256..511 de la matriz pantalla. Y los últimos 2 KB definen los patrones que aparecen para los últimos 256 valores que se pueden asignar en la TABLA DE NOMBRES y que se corresponden con los *tiles* 512..767 de la matriz pantalla. Dicho de otro modo, que en el banco 1 de la

matriz pantalla sólo puede dibujarse —a través de la TABLA DE NOMBRES— un patrón que esté definido en el banco 1 de la TABLA GENERADORA DE PATRONES. En el caso de que se necesite dibujar el mismo patrón en bancos distintos de la matriz pantalla, ese patrón deberá estar definido en cada uno de los bancos correspondientes de la TABLA GENERADORA DE PATRONES.

La fórmula para calcular qué byte de la tabla contiene el punto que corresponde con la coordenada X,Y de la pantalla, siendo la coordenada 0,0 de la pantalla la esquina inferior izquierda, es:

(Dirección inicio tabla)+((((191-Y) div 8)*256)+((191-Y) mod 8)+(X AND 0F8h))

sabiendo el byte, el bit se obtiene haciendo:

7-(X AND 7)

Esta tabla está sincronizada con la TABLA DE COLORES, de manera que cada byte de la TABLA DE GENERACIÓN DE PATRONES tiene su exacta correspondencia en la TABLA DE COLORES.

TABLA DE COLORES

Su dirección de inicio en VRAM es la 2000h y ocupa un tamaño de 6144 bytes. Está, al igual que la TABLA DE GENERACIÓN DE PATRONES, subdivida en tres bancos de 256 bytes cada uno. Cada bloque de 8 bytes de esta tabla define la combinación de colores para un sólo patrón —el que ocupa su mismo índice o direcciones de memoria— de la TABLA DE GENERACIÓN DE PATRONES, consiguiéndose una resolución de ocho colores verticales por dos colores distintos horizontalmente —*nibble* alto para el primer plano y *nibble* bajo para el fondo— en cada patrón.

La Figura 3.15 resume gráficamente lo expuesto sobre la configuración de pantalla y de estas tres tablas en este modo.

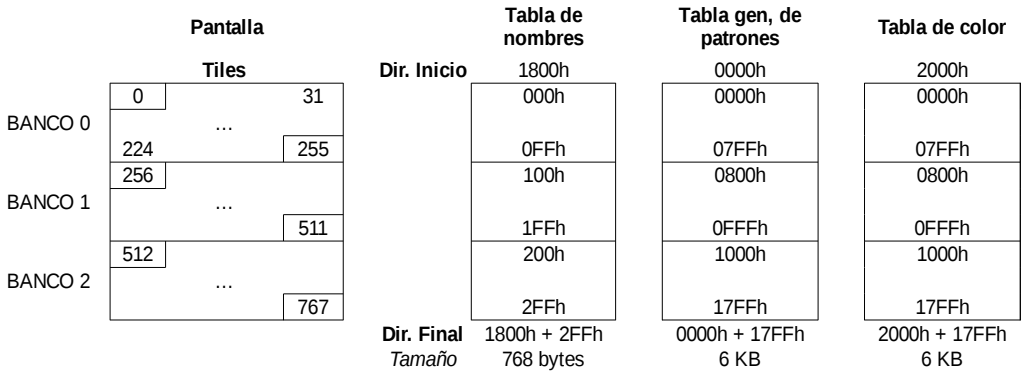

Figura 3.15: Modo 2: Tabla de nombre, de patrones y de colores

TABLA GENERADORA DE SPRITES

Como se ha visto en el punto *3.4.3 Memoria de video VRAM*, contiene las definiciones de los *sprites* disponibles. La dirección base de inicio de la TABLA GENERADORA DE SPRITES es la 3800h y tiene un tamaño de 2 KB.

En caso de usar *sprites* de tamaño 8 x 8 *píxeles* (8 bytes por *sprite*), es posible definir hasta 256 *sprites*. Si por contra, se usan *sprites* de tamaño 16 x 16 *píxeles* (32 bytes continuos por *sprite*), sólo pueden definirse hasta 64 *sprites*.

Los primeros 8 bytes de la tabla corresponderían a la definición del *sprite* 0 y los últimos 8 bytes de la tabla corresponderían a la definición del *sprite* 255, asumiendo un tamaño de *sprites* de 8 x 8 *píxeles*. En caso de *sprites* de tamaño 16 x 16 *píxeles*, los 32 primeros bytes de la tabla corresponderían con el *sprite* 0 y los últimos 32 bytes corresponderían con el *sprite* 63.

TABLA DE ATRIBUTOS DE SPRITES

Como se ha visto en el punto *3.4.3 Memoria de video VRAM,* contiene la información de los colores de los *sprites* diseñados, su ubicación en la pantalla y el número identificativo del *sprite*. La dirección base de inicio de la TABLA DE ATRIBUTOS DE SPRITES es la 1B00h y tiene un tamaño de 128 bytes que se dividen en 32 subtablas —una subtabla por cada uno de los 32 planos disponibles— de 4 bytes de largo.

En los puntos *3.4.3 Memoria de vídeo VRAM* y *3.4.6 Sprites* ya se ha tratado con profundidad la organización y significado de los 4 bytes de cada subtabla (véase la Figura 3.9).

MAPA DE MEMORIA VRAM

Las cinco tablas: Nombre (*map*), Generadora de patrones (*tileset pattern*), Color (*tileset color*), Generadora de *sprites* (*sprite patterns*) y Atributos de *sprites* (*sprite attributes-OAM*) constituyen el mapa de memoria en el que se organizan los 16 KB de memoria VRAM. Cada tabla tiene su dirección de inicio y su tamaño en bytes. La Figura 3.16[36] muestra el mapa de memoria para el modo 2 de pantalla de gráficos en alta resolución.

36 FUENTE: https://aorante.blogspot.com/2016/04/mapas-de-memoria-del-tms9918.html

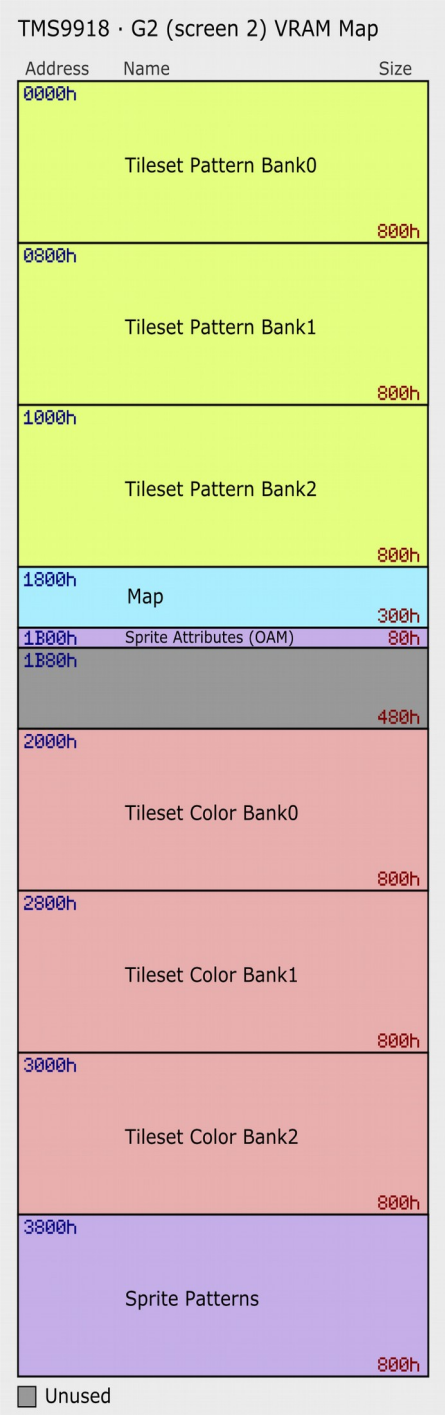

Figura 3.16: Mapa de memoria para el modo 2 de pantalla de gráficos en alta resolución

3.5 . Generador de sonido programable (PSG)

Es el circuito integrado encargado de generar la amplia gama de sonidos que el MSX puede emitir liberando de dicha tarea al procesador Z80, que sólo debe informar al PSG (acrónimo de *Programmable Sound Generator*) sobre el tipo de sonido y el cuándo emitir. La norma especifica que para la primera generación de MSX, el PSG usado sea el AY-3-8910 fabricado por General Instruments u otro equivalente. El PSG se organiza en varios bloques, cada uno con una función especifica. Los principales son:

- **Tres canales de sonido independientes (A, B y C)**: los cuales generan, cada uno y de forma simultánea, sonido que corresponde a las notas musicales.
- **Un generador de ruido**: pensado para generar efectos especiales.
- **Un generador de envolventes**: permite dar diferentes formas y calidades a los sonidos producidos, simulando instrumentos musicales.
- **Un controlador de amplitud** o volumen de cada canal.
- **Un mezclador**: mezcla los sonidos de los canales con el ruido.
- **Un conversor digital/analógico** que transforma las señales digitales del PSG en analógicas que puedan ser reproducidas en un altavoz.
- **16 registros internos de 8-bits**: 14 de ellos por los que accede el Z80 para dar las órdenes pertinentes y comenzar la generación de sonido. Los otros 2 registros no tienen relación con la generación de sonido, son dos puertos de E/S que se utilizan para controlar los mandos de juego (*joysticks*).

La Figura 3.17[37] muestra el diagrama de bloques que componen el AY-3-8910.

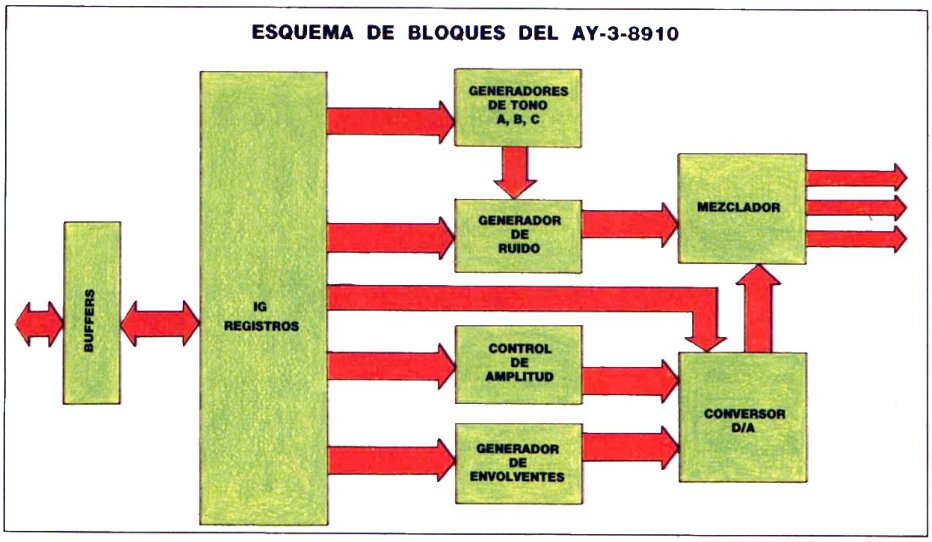

Figura 3.17: Diagrama de bloques del PSG

37 Fuente: [17]

3.5.1. Acceso al PSG

El acceso a los registros del PSG se hace a través de tres puertos del mapa de E/S (véase la Figura 3.7). Estos tres puertos son:

- puerto de direcciones (A0h): En este puerto se escribe el registro (0..15) al que se quiere acceder.
- puerto de escritura (A1h): Este puerto se utiliza para escribir un dato en el registro seleccionado en el *puerto de direcciones*.
- puerto de lectura (A2h): Este puerto se utiliza para leer el registro seleccionado en el *puerto de direcciones*.

3.5.2. Los registros del PSG

El AY-3-8910 tiene 14 registros a los que asignando valores adecuados puede seleccionarse los canales, ondas de sonido, frecuencia de tono, sonido y volumen.

- **Registros 0 y 1**: Estos dos registros se usan conjuntamente formando un "registro" de 12 bits —los 8 bits del registro 0 y los 4 bits menos significativos del registro 1 (véase la Figura 3.18)— para controlar la frecuencia del sonido del canal A. Los 4 bits del registro 1 que conforman los bits de mayor peso del "registro" de 12 bits, tienen mayor efecto en la frecuencia del sonido, por ello, se le denomina *ajuste grueso o de graves*. Y los 8 bits del registro 0 controlan la frecuencia del *ajuste fino o de agudos*.

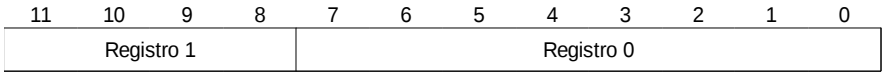

Figura 3.18: Registro de 12 bits del canal A del PSG

La frecuencia del tono generado en el canal depende del valor del "registro" de 12 bits y de la frecuencia de reloj aplicada. La frecuencia (f) de cada nota se calcula a partir de la fórmula:

$$f = \text{frecuencia\_reloj\_PSG} / (16 \cdot ((256 \cdot \textit{registro 1}) + \textit{registro 0}))$$

siendo la frecuencia de reloj del PSG del MSX de 1,78977 Mhz. Cuanto mayor sea el valor del registro de 12 bits, menor será el grado del tono generado por el canal A.

- **Registros 2 y 3**: Estos dos registros son análogos a los registros 0 y 1 pero operan sobre el canal B del PSG. El registro 2 controla el *ajuste fino o de agudos* y el registro 3 controla el *ajuste grueso o de graves*.
- **Registros 4 y 5**: Estos dos registros son, también, análogos a los registros 0 y 1 pero operan sobre el canal C del PSG. El registro 4 controla el *ajuste fino o de agudos* y el registro 5 controla el *ajuste grueso o de graves*.

- **Registro 6**: Este registro controla la frecuencia de la señal o el grado de ruido blanco que emitirá el PSG. El que exista un único registro para ello implica que todos los canales emitirán un ruido con la misma frecuencia. Sólo se hace uso de sus 5 bits de menor peso, por lo que el intervalo de valores que puede alcanzar es de [0..31], siendo el 31 el grado más bajo de ruido y el 0 el más alto.
- **Registro 7**: Este registro habilita/deshabilita el tono (sonido) de un canal. Es decir, controla la mezcla de señales en los distintos canales. Cada uno de los bits que lo componen controla una parte del funcionamiento del PSG. Cuando el bit está a '1' se deshabilita el tono y cuando está a '0' se habilita. Según la norma del estándar para el MSX de primera generación, el bit 6 debe estar a '0' y el bit 7 a '1' para el correcto funcionamiento del ordenador. La Figura 3.19[38] muestra la función de cada uno de los bits de este registro.

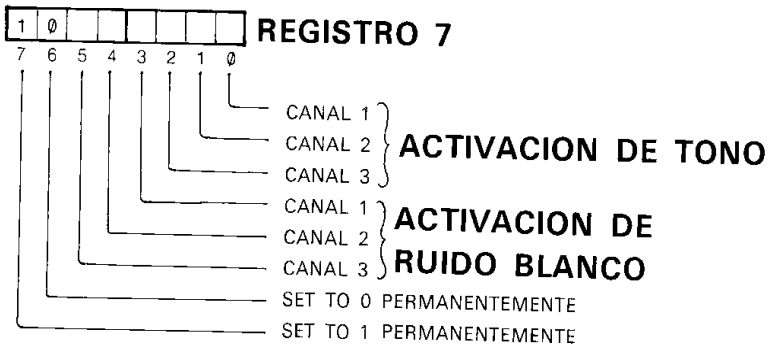

Figura 3.19: Función de los bits del registro 7 del PSG

- **Registros 8, 9 y 10**: Estos registros controlan la amplitud de onda o volumen de cada canal. El registro 8 corresponde con el volumen del canal A, el registro 9 con el volumen del canal B y el registro 10 con el volumen del canal C. Sólo se utilizan los 5 bits de menor peso. Los bits 0 al 3 determinan el volumen, por lo que rango de valores admitidos oscila entre 0 y 15. El valor de '0' implica silencio y el de '15' máxima intensidad de volumen. Cuando el valor de estos registros es 16 — el bit 4 toma el valor de '1'— se activa en el canal el generador de envolventes, es decir, el sonido varia su volumen según la forma de onda y la frecuencia.
- **Registros 11 y 12**: Estos registros determinan la frecuencia de repetición de la envolvente, en caso de que sea periódica. Entran en función para cada canal cuando los registros 8, 9 o 10 tienen el valor de 16. Los dos juntos forman un registro de 16 bits llamado *registro de control de periodo de la envolvente*. El registro 11 tiene los 8 bits de menor peso de dicho registro de 16 bits —*ajuste fino del periodo*— y el registro 12 contiene los 8 bits de mayor peso —*ajuste grueso del periodo*—. Cuanto más grande sea el valor del *registro de control de periodo de la envolvente*, mayor será el periodo de la envolvente. La Figura 3.20[39] muestra la amplitud o volumen de la onda desde su valor mínimo hasta su máximo, permaneciendo constante hasta que vuelve a su valor mínimo y el

38 Fuente: [7]

39 Fuente: [8]

periodo de la envolvente que dibuja una forma de onda con un incremento gradual de la amplitud hasta alcanzar su máximo seguido por una disminución hasta llegar su valor mínimo.

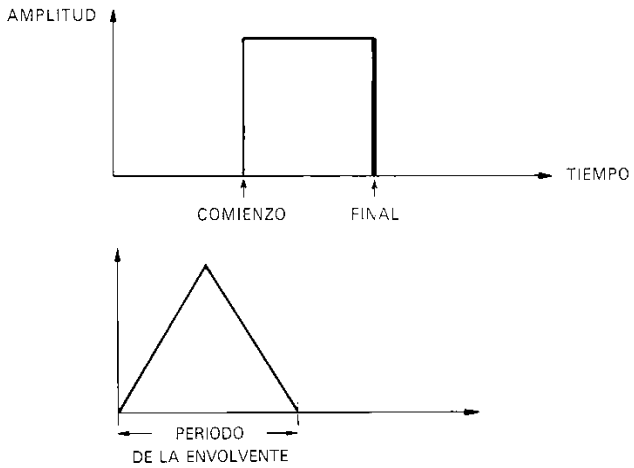

Figura 3.20: Amplitud y periodo de la envolvente de onda

El periodo de la onda, es decir, la frecuencia de cambio del volumen se calcula según la fórmula:

periodo = (registro 12) + 256 · (registro 11)

- **Registro 13**: Este registro controla la forma de la envolvente del PSG y sólo se utilizan los 4 bits de menor peso significativo, lo que hace que el rango de valores que puede tomar sea entre 0 y 15. El PSG tiene duplicadas algunas formas de onda, por lo que en lugar de tener disponibles 16 formas sólo existen 8 formas de onda diferentes. El hecho de que exista un sólo registro para el control de la forma de la envolvente hace que todos los sonidos emitidos por cualquiera de los tres canales tengan la misma envolvente.

  Los 4 bits usados en este registro reciben los nombres de:

  - **HOLD** (bit 0): determina si la envolvente mantiene o no el valor final del primer ciclo. Si el bit está a '0', la envolvente se repetirá periódicamente con la frecuencia determinada por los registros 11 y 12. En cambio, si el bit está a '1', la envolvente no se repetirá y el volumen del sonido se sostendrá al máximo o al mínimo según si el primer ciclo de la envolvente es ascendente o descendente.
  - **ALT** (bit 1): Si está a '1' se sucederán alternativamente rampas ascendentes y descendentes que darán a la envolvente una forma triangular.
  - **ATT** (bit 2): Este bit marca el periodo inicial de la producción de sonido, es decir, si el volumen del sonido aumenta desde el valor mínimo al máximo poco a poco o si el volumen es descendente desde el valor máximo al valor mínimo.

- **CONT** (bit 3): determina si la envolvente se repite periódicamente o no después del primer ciclo. Si su valor es '0', tras el primer ciclo, el volumen caerá a cero.

La Figura 3.21[40] muestra las 8 distintas formas de envolvente y la Figura 3.22[41] muestra de forma resumida la función de los 16 registros del PSG.

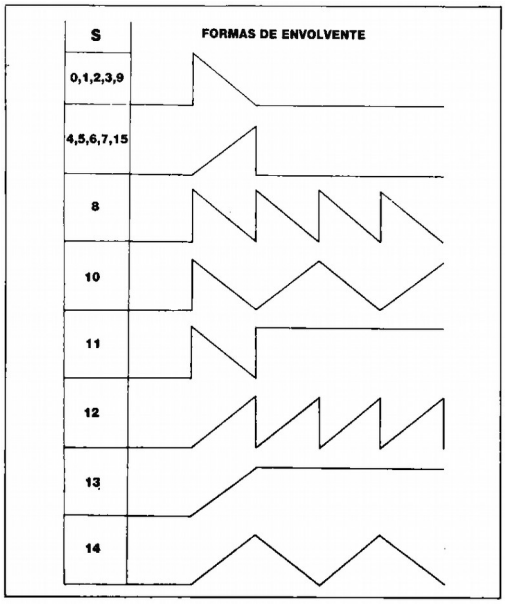

Figura 3.21: Formas de onda envolvente del PSG

REGISTROS DEL AY-3-8910

REGISTRO \ BIT		B7	B6	B5	B4	B3	B2	B1	B0
0	PERIODO DE LA SEÑAL DEL CANAL A	8 BITS DE AJUSTE FINO							
1						4 BITS DE AJUSTE GRUESO			
2	PERIODO DE LA SEÑAL DEL CANAL B	8 BITS DE AJUSTE FINO							
3						4 BITS DE AJUSTE GRUESO			
4	PERIODO DE LA SEÑAL DEL CANAL C	8 BITS DE AJUSTE FINO							
5						4 BITS DE AJUSTE GRUESO			
6	PERIODO DEL RUIDO				5 BITS DE AJUSTE				
7	ACTIVACION	E/S			RUIDO			TONO	
		IOB	IOA	C	B	A	C	B	A
8	AMPLITUD CANAL A				M	V3	V2	V1	V0
9	AMPLITUD CANAL B				M	V3	V2	V1	V0
10	AMPLITUD CANAL C				M	V3	V2	V1	V0
11	PERIODO DE LA ENVOLVENTE	8 BITS DE AJUSTE FINO							
12		8 BITS DE AJUSTE GRUESO							
13	FORMA DE ENVOLVENTE					CONT	ATT	ALT	HOLD
14	E/S PORT A	8 BITS DE E/S PARALELO							
15	E/S PORT B	8 BITS DE E/S PARALELO							

Figura 3.22: Registros del AY-3-8910

40 Fuente: [17]

41 Fuente: [17]

3.6 . Sistema de E/S (PPI)

Las operaciones de entrada y salida del MSX con los periféricos interconectados al ordenador (teclado, cartuchos insertados en bancos primarios externos como cartuchos de sonido o cartuchos gráficos, impresora, unidad de cintas magnéticas —casete—, unidad de disco...) están controladas por el componente Interfaz periférico programable (PPI).

El PPI es el circuito integrado Intel 8255 o equivalente y como se ha dicho, es el que permite conectar el ordenador con el mundo exterior. Dispone de cuatro registros internos, tres de los cuales son de entrada/salida —A, B y C— y un cuarto, de modo —*registro de selección de modo*—, que controla si los registros de E/S están configurados para entrada, salida o entrada y salida. El acceso a estos cuatro registros se realiza a través del rango de puertos A8h-ABh del mapa de entrada/salida. La BIOS, al inicio, activa el *registro de selección de modo* y configura el modo de los otros tres registros. No es recomendable modificar el valor del *registro de selección de modo* pues puede desconectarse el teclado o la memoria. La Tabla 3.8 muestra la configuración que realiza la BIOS de los registros de E/S del PPI.

Registro PPI	Puerto E/S	Modo	Uso
A	A8	Salida	Sistema de bancos primarios de memoria-E/S
B	A9	Entrada	Lectura del teclado
C	AA	Salida	Lectura del teclado, casete y puerto sonido de 1 bit
Reg. Sel. Modo	AB	-	Configura el modo de los registros A, B y C

Tabla 3.8: Registros del PPI

Uso de los registros de E/S del PPI:

- **Registro A**: se utiliza para controlar la asignación de las páginas lógicas de memoria a las páginas físicas de los bancos primarios de memoria. Ya se comentó su uso con profundidad en el punto *3.3 Mapa de memoria.*
- **Registro B**: devuelve un valor relacionado con el teclado que adquiere significado junto con el *nibble* bajo del registro C. Bits con valor a '0' indica la pulsación de la tecla correspondiente. Un valor a '1', por contra, indica que la tecla asociada no ha sido pulsada.
- **Registro C**: Su función principal es la de ayudar a la lectura del teclado. Al *nibble* bajo se le denomina *señal de exploración del teclado (Keyboard scan signal)*. Al fijar un valor en el *nibble* bajo se determina la fila de la matriz del mapa de teclado que será leída en el registro B. Observando los bits puestos a '0' se puede detectar que tecla o teclas han sido pulsadas. La Tabla 3.9[42] muestra como se lee el mapa de teclado del MSX haciendo uso de los registros B y C. Los bits 4 y 5 están relacionados con el control de la unidad de cinta magnética. El bit 6 controla el encendido o apagado de la luz de bloqueo de las mayúsculas —el valor '0' enciende la luz y un '1' la apaga— y el bit 7 es denominado *puerto de*

42 Fuente: [7]

sonido de 1 bit pues cuando este bit pasa de '1' a '0' el sistema de sonido emite un sonido en forma de *click*.

REGISTRO B* Modelos de bits leídos en el registro B								REGISTRO C** Modelos de bits escritos en el registro C
7	6	5	4	3	2	1	0	3 2 1 0
7	6	5	4	3	2	1	0	0000
;	[	@	¥	^	—	9	8	0001
B	A	—	/	•	,	]	:	0010
J	I	H	G	F	E	D	C	0011
R	Q	P	O	N	M	L	K	0100
Z	Y	X	W	V	U	T	S	0101
F3	F2	F1		CAPS	GRAPH	CTRL	SHIFT	0110
RE-TURN	SELECT	BS	STOP	TAB	ESC	F5	F4	0111
→	↓	↑	←	DEL	INS	HOME CLS	SPACE	1000

Tabla 3.9: Tabla del teclado MSX-1

La E/S es recomendable manejarla utilizando las rutinas de la BIOS porque no existe la garantía de que las localizaciones del sistema sean homogéneas entre los distintos aparatos.

En el apartado *3.3 Mapa de memoria* se ha comentado la existencia, por lo general, de dos de los bancos primarios como bancos externos destinados a la inserción, entre otros, de cartuchos que fueran periféricos como, por ejemplo, unidades de disco o los recientes cartuchos con nuevos chips gráficos como la *Power Graph Ligth*[43] o sonoros como la *OPL4 Shockwave 2*[44] ambos de Tecnobytes. La comunicación entre estos periféricos y el ordenador será a través de puertos propios del mapa de E/S, visto en el punto *3.3.3 Mapa de entrada/salida*, en lugar del puerto A8 (*registro selector de banco*) que corresponde con registro A del PPI.

43 http://www.tecnobytes.com.br/p/v9990-powergraph-light.html

44 http://www.tecnobytes.com.br/2018/09/new-opl4-shockwave-2-batch-22018.html

3.7 . El sistema de Bus

El sistema MSX dispone de tres *buses* (datos, direcciones y control) que permiten la comunicación y la transferencia de datos entre los distintos componentes que lo conforman.

El *bus* de datos es el canal por el que el Z80 transfiere datos a la memoria, al VDP, al PSG y al PPI. El tamaño del *bus* de datos es de 8 bits y el modo en el que el Z80 transfiere los datos es volcando secuencialmente los bytes de datos en el *bus*.

El *bus* de direcciones es el canal por el que el Z80 accede a las posiciones de memoria. Su tamaño es de 16 bits y es el que fija la cantidad máxima de memoria que el microprocesador puede gestionar. Cuando el Z80 coloca un byte en el *bus* de datos, también coloca dos bytes en el *bus* de direcciones para indicar la dirección en la que el byte de datos debe ser escrito. En caso de una operación de lectura, coloca los bytes de la dirección del dato a ser leído en el *bus* de direcciones y posteriormente accede al dato desde el *bus* de datos. Esta dirección puede referirse tanto a una posición de memoria como a un puerto de E/S del PPI, del VDP o del PSG.

El *bus* de control es el canal por el cual el Z80 gobierna todas las transferencias de datos e informa a los componentes y periféricos conectados a los *buses* de datos y direcciones que quiere leer o escribir datos en ellos.

Existe un *bus* exclusivo del VDP que lo conecta a la memoria de vídeo (VRAM). Este *bus* está compuesto por uno de datos, otro de direcciones y otro de control. El Z80 no tiene acceso a este *bus*.

3.8 . MSX-DOS *(Disk Operating System)*

MSX-DOS es un sistema operativo de disco para los ordenadores MSX. Existen dos versiones, el MSX-DOS1, que requiere un mínimo de 64 KB de memoria RAM y admite hasta seis unidades simultáneas, y el MSX-DOS2, que requiere de memoria mapeada y un mínimo de 128 KB de memoria, admite hasta ocho unidades simultáneas y es capaz de manejar subdirectorios y discos duros.

El MSX-DOS se compone de dos módulos:

- la *interfaz o controladora de disco* almacenada en su propia memoria ROM, llamada BDOS (Basic DOS) o *Disk-ROM*. Esto es así porque la BIOS estándar de un MSX no tiene soporte para disco, así que hubo de expandirla añadiendo la controladora en un banco primario expandido —en aquellos ordenadores que incluyeran alguna unidad— o externamente a través de un cartucho que se insertara en algún banco de cartucho —en aquellos ordenadores sin unidad de disco—. Incluye rutinas para accionar la unidad de disco, el núcleo del DOS y el

intérprete de BASIC ampliado con soporte de disco (*Disk-Basic*). Se ubica en la página lógica 1 de la memoria sustituyendo al Intérprete de Basic original.

- Dos archivos de sistema ubicados en los discos: MSXDOS.SYS y COMMAND.COM (MSXDOS2.SYS y COMMAND2.COM para MSX-DOS2). El archivo MSXDOS.SYS sirve para configurar los parámetros necesarios para el funcionamiento del COMMAND.COM en el arranque del MSX-DOS y es el intermediario entre las operaciones de E/S realizadas por la BDOS a requerimiento del COMMAND.COM. El archivo COMMAND.COM es el interfaz de línea de comandos responsable de la ejecución de las órdenes MSX-DOS, que pueden ser: internas —ya implementadas en el COMMAND.COM—, externas —en archivos con extensión .COM— o archivos de proceso por lotes —ficheros *batch* con extensión .BAT—.

No sólo agrega comandos de soporte de disco al intérprete de BASIC sino también un sistema de arranque, con el cual es posible arrancar un sistema operativo de disco real ya que los MSX están diseñados para cargar el sistema operativo de disco y transferirle el control siempre que encuentren una copia de los archivos de sistema en el disco. En ese caso, la BDOS omite la ROM básica, de modo que los 64 KB completos del espacio de direcciones del microprocesador Z80 pueden usarse para el DOS o para otros discos autoarrancables, por ejemplo, juegos basados en disco. Al mismo tiempo, se puede acceder a la ROM de la BIOS original a través del mecanismo de "cambio de banco primario de memoria" (descrito en el punto *3.3 Mapa de memoria*), de modo que el *software* basado en DOS aún puede usar llamadas a la BIOS. Además, debido a la ROM de BDOS, las capacidades básicas de acceso a archivos están disponibles incluso sin un intérprete de comandos mediante el uso de comandos BASIC extendidos.

La BDOS, para dejar a disposición del Z80 los 64 KB de espacio direccionable como memoria RAM, necesita, primero, localizar su ubicación en el ordenador. Para ello, aprovecha que la BIOS, al iniciarse el ordenador, ya hace una búsqueda para localizar las páginas 2 y 3 lógicas, por tanto sólo busca la memoria RAM oculta al Z80, es decir, la que se ubica en las páginas físicas 0 y 1. Tras realizar la búsqueda y en caso de encontrar memoria RAM en dichas páginas, almacena su ubicación en la memoria RAM reservada para el sistema. En la dirección de memoria F341h guarda el banco primario y segmento para la página lógica 0, en F342h para la página lógica 1, en F343h para la página lógica 2 y en F344h para la página lógica 3. El formato en el que almacena dicha información es:

- bit 0-1 = Número del banco primario de memoria
- bit 2-3 = Número del segmento expandido (opcional)
- bit 4-6 = Sin uso
- bit 7 = 1 → El banco primario está expandido

En caso de no poder encontrar un mapa de memoria lineal de 64 KB, el MSX-DOS no se ejecuta y salta directamente al *Disk-Basic*.

Hoy en día es posible disponer de MSX-DOS en ordenadores MSX que originariamente no estaban equipados para soportarlo gracias a nuevo *hardware* en cartucho que, además de incorporar un expansor de banco primario que permite incorporar memoria RAM con mapeadores de memoria, procesadores de sonido

avanzados y unidades de almacenamiento masivo como tarjetas microSD —véase por ejemplo la MegaFlashROM SCC+ SD[45] de Manuel Pazos, la Flashjack[46] de RetroMSX o la Rookie Drive[47] de Xavier Peirats (alias Xavirompe)—, incorporan el sistema operativo de disco NEXTOR[48] de Néstor Soriano (alias Konamiman) que es una versión mejorada del MSX-DOS.

3.8.1 Mapa de memoria del MSX-DOS

El MSX-DOS reserva para el sistema dos zonas de memoria. De la zona baja de memoria se reserva el espacio comprendido por los doscientos cincuenta y seis primeros bytes (direcciones 0000 h – 0099 h), zona conocida como *System Scratch*. Esta zona de memoria contiene algunos datos usados por el MSX-DOS para comunicarse con los programas de usuario así como la dirección de salto para las llamadas al sistema.

De la parte alta de la memoria se reserva entorno a 10 KB (D806h – FFFFh) para uso del sistema (unidad de disco u otros dispositivos conectados, etc.). Esta zona, además de las variables del sistema, contiene las rutinas de MSX-DOS para el correcto funcionamiento del MSX-DOS y del programa de usuario.

Entremedias de ambas zonas se ubica el espacio de memoria disponible para programas de usuario que funcionan bajo MSX-DOS. Esta zona es conocida como *Área Transitoria del Programa* o TPA (por sus siglas en inglés *Transient Program Area*). Se ubica a continuación de la zona de *System Scratch*, siendo su dirección más alta variable ya que depende de la configuración *hardware* del ordenador. Esta dirección puede conocerse puesto que se almacena en las posiciones 0006 hy 0007h (TPA) del *System Scratch*.

En la zona alta del TPA se ubica la pila de ejecución del programa cuyo tamaño depende de la cantidad de variables y llamadas a funciones usadas en los programas. Al aumentar el tamaño de la pila, decrece el tamaño de memoria disponible. Así pues, el espacio de memoria disponible se calcula como:

$$\text{memoria libre para programas} = \text{TPA} - \text{tamaño de la pila} - 0100h$$

La Figura 3.23[49] muestra la organización que hace el MSX-DOS de la memoria RAM.

45 MegaFlashROM SCC+ SD: https://www.msxcartridgeshop.com/

46 Flashjack: https://www.retromsx.com/flashjacks-spec/

47 Rookie Drive: http://rookiedrive.com/es/

48 NEXTOR: https://www.konamiman.com/msx/msx-s.html#nextor

49 Fuente: [28]

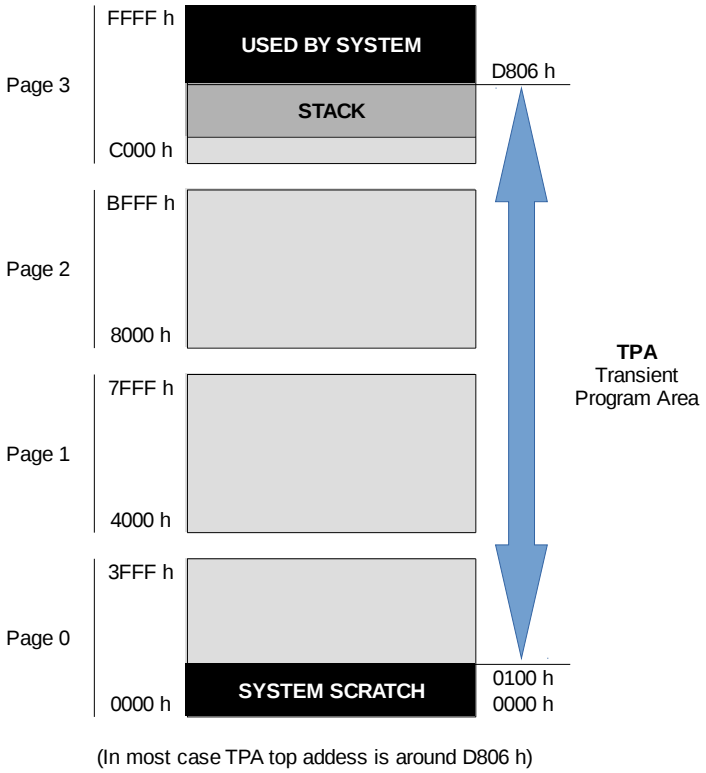

Figura 3.23: Mapa de memoria del MSX-DOS

3.8.2 Acceso a los archivos de disco

El acceso a los archivos y directorios de disco a través de la FAT es bastante complejo, por tanto se realiza a través del *Bloque de Control de Archivo* o FCB —acrónimo anglosajón de *File Control Block*—. El FCB es el área de memoria donde se almacena la información necesaria para manejar archivos mediante llamadas al sistema en MSX-DOS. La dirección de memoria destinada a localizar el FBC puede ser cualquiera pero el MSX-DOS usa la dirección 005Ch, ocupando un tamaño de 37 bytes. Al programador le es suficiente con facilitar la unidad de disco y el nombre del archivo —compuesto por hasta 8 caracteres y, separados por un '.', una extensión opcional de 3 caracteres— para que el FCB pueda acceder a él. La Tabla 3.10[50] muestra la estructura del FCB.

Las operaciones básicas realizables sobre los archivos de disco son las mismas que pueden encontrarse en otros sistemas operativos de disco: crear, abrir, cerrar, leer y escribir.

50 Fuente: [12]

Desplazamiento (Offset)	COMENTARIOS
+0	Número de unidad (0=por defecto, 1=A:, 2=B:, etc.)
+1/+11	Nombre del archivo y extensión
+12/+13	Bloque actual
+14/+15	Tamaño del registro aleatorio (1 a 65535)
+16~+19	Tamaño del archivo en bytes (1 a 4.294.967.297)
+20/+21	Data (mismo formato que directorio)
+22/+23	Hora (mismo formato que directorio)
+24	+ID del dispositivo
+25	Localización del directorio
+26/+27	Primer clúster del archivo
+28/+29	Último clúster accedido
+30/+31	Localización relativa del clúster
+32	Registro secuencial actual
+33/+36	Número del registro aleatorio

Tabla 3.10: Estructura del FCB

El MSX-DOS1 no soporta subdirectorios, por lo que todos los ficheros que se hallan en el disco necesariamente deben ubicarse dentro del directorio raíz del disco. El directorio raíz puede albergar un máximo de 255 ficheros.

4 . El entorno de desarrollo

En este capítulo se van a presentar las herramientas, libres y gratuitas, a utilizar en un ordenador actual y que conforman un completo entorno de desarrollo cruzado para la realización *homebrew* de videojuegos destinados a ordenadores MSX. Estas herramientas abarcan todos los aspectos que integran un videojuego, desde la programación en sí hasta los apartados gráficos y sonoros del mismo. Alguna de estas aplicaciones sólo existe para el sistema operativo Windows, por lo que la versión que se presenta de ellas es para dicho sistema operativo.

El capítulo se ha dividido en dos secciones. La primera describe las aplicaciones que acompañan a la librería FUSION-C para la programación y verificación del videojuego. La segunda sección presenta aplicaciones terceras adicionales para el desarrollo de los apartados gráficos y sonoros de un videojuego.

El proceso de instalación y configuración de estas aplicaciones, así como el proceso de compilación, se anexan en los apéndices 1 (ANEXO I - Instalación del SDK) y 2 (ANEXO II - Configurar la aplicación GIMP para pintar imágenes SCREEN 2).

La principal fuente bibliográfica consultada para la elaboración de este apartado ha sido [28], además de las propias páginas web de las aplicaciones [29][30][31][32][33][34].

4.1 . Fusión-C

FUSION-C es una librería libre y gratuita de C dedicada al MSX para el compilador cruzado SDCC. Implementa funciones para manejar el *hardware* del MSX facilitando el desarrollo de programas y juegos de manera sencilla para los ordenadores MSX de cualquier generación. Está pensada para que los programas compilados se ejecuten bajo MSX-DOS pero permite la transformación posterior a formato ROM para aquellos ordenadores sin sistema operativo de disco.

El nombre de la librería proviene de la fusión de varías librerías ya existentes desarrolladas parcialmente, principalmente de la librería *Solid-C* desarrollada por Ego Voznessenski y de las funciones desarrolladas por T. Hara, Néstor Soriano, Jorge Torres Chacón, Alberto Orante, Eric Boez y Fernando García.

Entre las funciones que implementa se encuentran:

- Manejo de modos de pantalla de *Screen* 0 a 8.
- Dibujo.

- Copia de un área gráfica en VRAM o RAM.
- Manejo de *sprites*.
- Manejo de ficheros bajo MSX-DOS.
- Control de teclado y *joystick*.
- Manejo de memoria mapeada bajo MSX-DOS2.
- Manejo del PSG para la parte de sonido.
- Reproductor para ficheros musicales .PT3.
- Reproductor de ficheros AYFX para efectos de sonido.
- etc.

La última versión realizada del la librería hasta la fecha es la 1.2, de septiembre de 2019. La librería viene acompañada de documentación a modo de manual rápido y de herramientas y *scripts* que automatizan la compilación para los sistemas operativos Windows / Linux / MacOs. Los programas principales del entorno de desarrollo de FUSION-C están disponibles libremente en Internet, aunque también existe una compilación que los incluye llamada *Fusion-C Tools-Chain*. Estos programas son el SDCC 3.6, el módulo *Hex2Bin* que convierte el código de SDCC en código utilizable en MSX-DOS, el editor de código *Sublime Text* y el emulador *openMSX*.

La versión recomendada del compilador SDCC es la 3.6 —SDCC (*Small Device C Compiler*) es un compilador C estándar optimizado que puede producir código Zilog Z80 para múltiples computadoras basadas en Z80. El compilador incluye una biblioteca C estándar parcialmente compatible con el MSX—.

FUSION-C viene configurado con una estructura definida de directorio de trabajo sobre el que desarrollar los proyectos. Esta estructura de árbol cuelga del directorio *Woking Folder* y es la siguiente (la letra negrita diferencia directorios de ficheros):

Working Folder
|_**dsk**
|_command.com
|_msxdos.sys
|_**fusion-c**
|_**examples**
|_**header**
|_**include**
|_**lib**
|_**source**
|_**lib**
|_**openMSX**
|_**Tools**
|_Makefile
|_compil.bat

El contenido y función de los directorios y ficheros es:

- **dsk**: directorio destino de los programas compilados. Representa el disco que está insertado en el MSX, por ello incorpora los dos archivos de sistema que se ubican en los discos: MSXDOS.SYS y COMMAND.COM (MSXDOS2.SYS y COMMAND2.COM para MSX-DOS2).
- **fusion-c**: directorio principal donde se encuentra la librería.
  - **examples**: directorio con ejemplos de programas compilados con su código fuente.
  - **header**: directorio con todos los ficheros cabecera de C ('.h') que componen la librería FUSION-C.
  - **include**: directorio con los ficheros binarios usados por el compilador.
  - **lib**: directorio que incluye la librería dinámica FUSION-C.
  - **source**: directorio que contiene códigos fuente de algunas funciones de la librería.
    - **lib**: contiene todo el código fuente de la librería y el *script* de compilación de la misma.
- **openMSX**: directorio que contiene el emulador openMSX[51]. La emulación se realiza sobre un MSX-2 (Philips NMS 8255) con MSX-DOS2.
- **Tools**: directorio que contiene herramientas útiles para el desarrollo de videojuegos.
- *Makefile*: fichero de proceso por lotes (*script*) para sistemas operativos Linux y MacOS que automatiza la compilación.
- *Compil.bat*: fichero de proceso por lotes (*BAT*) para sistema operativo Windows que automatiza la compilación.

El proceso de compilación, además de compilar el código fuente del programa y copiar el fichero ejecutable al directorio **dsk**, lanzará el emulador openMSX con el directorio **dsk** como disco insertado. Si el emulador ya se encuentra en ejecución, la compilación no volverá a lanzar otra instancia del emulador. Bastará con seguir usando el instancia en ejecución del emulador.

El directorio **Tools** contiene variadas herramientas externas a la librería que ayudan al desarrollo de videojuegos, entre ellas figuran:

- **AYFX Editor**: editor cruzado de efectos de sonido para los chips AY-3-8910 / YM2149F y compatibles. Incluye algunos efectos de sonido.
- **MSX Disk image**: aplicación que permite crear ficheros imagen de disco (.DSK). Permite, también, añadir o eliminar ficheros dentro de la imagen de disco.
- **MSX-DOS**: contiene los ficheros de sistema tanto del MSX-DOS1 como del MSX-DOS2.
- **DSK2ROM**: programa conversor de ficheros imagen de disco (.DSK) a ficheros imagen ROM (.ROM), creado por Vicent Van Dam.

51 Emulador openMSX: https://openmsx.org/

4.2 . Programas adicionales

4.2.1 GIMP

GIMP, acrónimo de las siglas inglesas *GNU Image Manipulation Program*, es un programa libre y gratuito de edición y retoque de imágenes disponible para los sistemas operativos Windows, MacOS y GNU/Linux.

4.2.2 MSX tiles devtool

MSX tiles devtool es una aplicación libre y gratuita disponible para el sistema operativo Windows. Desarrollada por Alberto Orante para trabajar con imágenes en el modo 2 de gráficos en alta resolución del VDP TMS9918 y como complemento a la aplicación *nMSXtiles*. Incluye herramientas que permiten cambiar los patrones y colores de la imagen por otro, visualizar la pantalla o los bancos de patrones/colores, además de poder exportar la imagen resultante a diferentes formatos de código: C, ensamblador o Basic.

Permite tanto la importación de proyectos de la aplicación *nMSXtiles* y de imágenes en formatos *bitmap* (PNG/GIF) y SC2 (binario del MSX Basic) como su exportación a dichos formatos, además de los formatos de código.

4.2.3 NMSX tiles

NMSX tiles es un editor de patrones y pantallas para el modo 2 de los ordenadores MSX. También permite editar *sprites*. Desarrollada por Ramón de las Heras y Gerardo Herce, está disponible para los sistemas operativos Windows y MacOs.

Permite la exportación tanto de la pantalla como de los bancos que conforman los patrones y sus colores a formato *bitmap* (PNG), a código ensamblador y SC2 (binario del MSX Basic).

4.2.4 SpriteSX devtool

SpriteSX devtool es una herramienta libre y gratuita para crear colecciones de *sprites* para videoprocesadores de la familia TMS9918, simplificando el trabajo de creación y edición de *sprites*. Desarrollada por Alberto Orante, permite exportar en código fuente (C, Ensamblador y Basic) los *sprites* generados. Está disponible para sistema operativo Windows.

4.2.5 Vortex Tracker 2

Vortex Tracker 2 es un completo editor de música para el AY-3-8910. Permite componer música mediante muestras de sonido y un editor/secuenciador de tramas de tiempo que se reparten en los distintos canales. Es capaz de importar diversos formatos de audio de otros editores —como el Amadeus (formato AY)— y exporta la música compuesta a formato PT3 y formato texto.

Es una herramienta libre y gratuita creada por Sergey Bulba y está disponible para el sistema operativo Windows.

5 . La librería FUSION-C

En este capítulo se van a presentar algunas de las funciones, agrupadas en módulos, que integran la librería FUSION-C y que implementan el acceso y el tratamiento de los componentes que conforman el *hardware* del sistema MSX de primera generación.

El capítulo se ha dividido en seis secciones. La primera sección enumera y comenta brevemente los módulos que integran la librería necesarios para programar los ordenadores MSX-1. El resto de secciones lista las funciones dedicadas a cada componente (según la estructura del capítulo 3), indicando a qué módulo pertenecen.

La principal fuente bibliográfica consultada para la elaboración de este apartado ha sido [28].

5.1. Módulos de FUSION-C

- **msx_fusion.h**: contiene la mayoría de funciones que componen la librería, incluyendo funciones de acceso a la consola, al teclado y *joysticks*, a los puertos del mapa de E/S, a la memoria principal, al VDP y PSG, a ficheros y el control de las interrupciones.
- **vdp_graph1.h**: contiene las funciones gráficas para el TMS9918A (VDP del MSX-1).
- **vdp_sprites.h**: contiene las funciones relativas al tratamiento de *sprites* por *hardware*.
- **psg.h**: contiene las funciones relativas al sonido.
- **ayfx_player.h**: contiene las funciones que permiten reproducir efectos de sonido almacenados en ficheros de sonido de tipo AYFX.
- **pt3player.h**: contiene las funciones que permiten reproducir ficheros de sonido almacenados en ficheros con formato PT3.
- **io.h**: contiene las funciones relativas a la entrada y salida de la unidad de disco.

5.2. Mapa de memoria

Las funciones dedicadas a los accesos a la memoria principal están en el archivo cabecera **msx_fusion.h**. A continuación se listan las funciones y su propósito.

- **Poke**: escribe un byte de tipo *char* en una dirección de memoria dada.
- **Pokew**: escribe un dato de tipo *int* (2 bytes) en una dirección de memoria dada.
- **Peek**: lee un byte de tipo *char* de una dirección de memoria dada.
- **Peekw**: lee un dato de tipo *int* (2 bytes) de una dirección de memoria dada.
- **MemChr**: devuelve un puntero de tipo *char* en n bytes, o NULL si no se encuentra.
- **MemFill**: rellena la memoria comprendida entre un rango de direcciones con un valor concreto.
- **MemCopy**: copia un número de bytes de una dirección a otra.
- **MemCopyReverse**: copia un número de bytes de una dirección a otra pero, a diferencia de MemCopy, comienza a copiar desde la dirección final del origen.
- **MMalloc**: versión del compilador SDCC de la función *malloc* del lenguaje C.
- **ReadTPA**: devuelve la dirección TPA del sistema actual de ejecución de MSX-DOS.
- **ReadSP**: devuelve la dirección inferior de la pila. La pila está dentro de la zona TPA. La pila crece y disminuye mientras el programa se está ejecutando.

5.3. Procesador de vídeo (VDP)

Las funciones dedicadas al procesador gráfico se dividen en varios módulos. Del archivo cabecera **MSX_FUSION.h**:

- **Screen**: fija el modo de pantalla (0..3).
- **VDPstatus**: lee el *registro de estado* del VDP. Previo a la lectura del registro se desactivan las interrupciones, volviendo a ser activadas tras la operación de lectura.
- **VDPstatusNi**: lee el *registro de estado* del VDP sin desactivar las interrupciones.

- **VDPwrite**: escribe un dato en un registro dado del VDP. Previo a la escritura del registro se desactivan las interrupciones volviendo a ser activadas tras la operación de escritura.
- **VDPwriteNi**: escribe un dato en un registro dado del VDP sin desactivar las interrupciones.
- **Vpoke**: escribe un byte en una dirección concreta de la VRAM.
- **VpokeFirst**: fija la primera dirección de VRAM a partir de la que se va a escribir un bloque consecutivo de bytes.
- **VpokeNext**: escribe el bloque de bytes fijado por la instrucción VpokeFirst. El incremento de las direcciones donde se escribe es automático.
- **Vpeek**: lee un byte en una dirección concreta de la VRAM.
- **VpeekFirst**: fija la primera dirección de VRAM a partir de la que se va a leer un bloque consecutivo de bytes.
- **VpeekNext**: lee el byte de la dirección fijada por la instrucción VpeekFirst e incrementa la dirección para próximos usos de VpeekNext.
- **Width**: fija el número de columnas (1 – 80) de la pantalla para los modos de texto.
- **SetColors**: fija los colores de la tinta, del fondo y de los bordes de la pantalla.
- **HideDisplay**: oculta la visualización de la pantalla, mostrando una pantalla en negro.
- **ShowDisplay**: muestra la pantalla si ha sido ocultada previamente.
- **FillVram**: rellena un espacio de direcciones de la VRAM con un dato concreto a partir de una dirección de inicio y la cantidad de direcciones contiguas a rellenar.
- **GetVramSize**: devuelve el tamaño de la VRAM.
- **CopyRamToVram**: copia un bloque datos desde la memoria principal a la memoria gráfica.
- **CopyVramToRam**: copia un bloque datos desde la memoria gráfica a la memoria principal.

Del archivo cabecera **VDP_GRAPH1.h**:

- **SC2ReadScr**: vuelca el contenido de la memoria VRAM en la memoria principal.
- **SC2WriteScr**: escribe en la VRAM desde la memoria RAM (2x6144 bytes).

- **ReadBlock**: copia un bloque de memoria VRAM a la memoria RAM dadas unas coordenadas origen y tamaño de la pantalla.
- **WriteBlock**: copia un bloque de memoria RAM a la memoria VRAM a partir de unas coordenadas origen de la pantalla dadas.
- **Get8px**: devuelve el byte del *píxel* que ocupa una coordenada concreta de la pantalla.
- **Set8px**: escribe el byte del *píxel* que ocupa una coordenada concreta de la pantalla.
- **Clear8px**: pone a cero el byte del *píxel* que ocupa una coordenada concreta de la pantalla.
- **GetCol8px**: devuelve el byte de la Tala de Color correspondiente al patrón de la Tabla de Patrones que ocupa una coordenada concreta de la pantalla.
- **SetCol8px**: escribe el byte en la Tala de Color que corresponde al patrón de la Tabla de Patrones que ocupa una coordenada concreta de la pantalla.
- **SC2Pset**: dibuja un punto en las coordenadas de la pantalla dadas estableciendo el color de patrón completo.
- **SC2Line**: dibuja una línea desde un punto a otro de la pantalla con un color concreto sin variar el color de fondo.

Del archivo cabecera **VDP_SPRITES.h**:

- **SpriteOn**: habilita los *sprites* por *hardware*.
- **SpriteOff**: deshabilita los *sprites* por *hardware*.
- **Sprite8**: fija el tamaño de 8x8 puntos a los patrones de los *sprites*.
- **Sprite16**: fija el tamaño de 16x16 puntos a los patrones de los *sprites*.
- **SpriteSmall**: fija el tamaño de los *sprites* a normal (sin ampliar).
- **SpriteDouble**: fija el tamaño de los *sprites* a modo ampliado.
- **SpriteReset**: restablece todos los atributos y patrones de *sprites*.
- **SpriteCollision**: devuelve el valor de '1' en caso de detectarse colisión de *sprites*.
- **PutSprite**: dibuja en pantalla un *sprite* concreto con un determinado color a partir de unas coordenadas dadas.
- **Sprite32Bytes**: define patrones de *sprites* de tamaño 16x16 puntos (4 cuadrantes).

5.4. Generador de sonido programable (PSG)

Las funciones dedicadas al procesador de sonido se dividen en varios módulos. Del archivo cabecera **MSX_FUSION.h**:

- **Beep**: emite un pitido.
- **InitPSG**: Inicializa todos los registros del PSG a cero, parando todos los sonidos y ruidos.
- **PSGread**: lee el contenido de un registro concreto del PSG.
- **PSGwrite**: escribe un byte en un registro concreto del PSG.

Del archivo cabecera **PSG.h**:

- **Sound**: escribe un valor en uno de los registros del PSG.
- **SoundFX**: hace sonar un efecto sonoro en un canal concreto.
- **SetChannelA**: activa/desactiva el sonido (tono) y el ruido de un canal concreto.
- **SilencePSG**: silencia los tres canales del PSG.
- **GetSound**: lee el contenido de un registro concreto del PSG.
- **SetTonePeriod**: fija el periodo del sonido (tono) de un canal.
- **SetNoisePeriod**: fija el periodo del ruido de un canal.
- **SetEnvelopePeriod**: fija el periodo de la envolvente.
- **SetVolume**: fija el volumen de un canal (valores entre 0 y 15). El valor 16 activa la envolvente.
- **SetChannel**: activa/desactiva el sonido (tono) y el ruido de un canal.
- **PlayEnvelope**: reproduce el sonido del canal que tiene por valor 16 con la onda envolvente indicada.

Del archivo cabecera **AYF_PLAYER.h** (reproductor AYFX_player):

- **InitFX**: inicializa el reproductor de efectos sonoros de formato AY.
- **PlayFX**: reproduce el efecto sonoro indicado del banco de sonidos.
- **UpdateFX**: actualiza la reproducción del efecto sonoro.
- **TestFX**: comprueba si se está reproduciendo algún efecto sonoro.
- **FreeFX**: libera la memoria usada por el reproductor de efectos sonoros de formato AY.

Del archivo cabecera **PT3REPLAYER.h** (reproductor de música en formato PT3):

- **PT3Init**: inicializa el reproductor de música, carga el fichero almacenado en formato PT3. Puede indicarse si se desea que reproduzca la música en bucle.
- **PT3Play**: actualiza la reproducción de la música.
- **PT3Rout**: prepara los datos para ser reproducidos.
- **PT3Mute**: silencia la música.

5.5. Sistema de Entrada / Salida (PPI)

5.5.1. Teclado y mandos de juego

Las funciones dedicadas al teclado y a los mandos de juego se encuentran en el archivo cabecera **msx_fusion.h**:

- **GetKeyMatrix**: devuelve el valor de una línea concreta de la matriz del teclado. Cada línea proporciona el estado (pulsada/no pulsada) de 8 teclas.
- **KillKeyBuffer**: vacía la memoria (*buffer*) del teclado.
- **InKey**: comprueba si alguna tecla ha sido pulsada.
- **WaitKey**: espera hasta que una tecla es pulsada.
- **KeyboardRead**: devuelve el estado de algunas teclas del teclado. Puede informar de si más de una tecla ha sido pulsada.
- **JoystickRead**: lee el estado de uno de los mandos de juego (*joystick*).
- **TriggerRead**: lee el estado de los botones de disparo de uno de los mandos de juego.

5.5.2. Mapa de E/S

Las funciones dedicadas al acceso a los puertos que componen el mapa de entrada y salida se encuentran en el archivo cabecera **msx_fusion.h**:

- **InPort**: lee un byte de un puerto concreto.
- **OutPort**: escribe un byte en un puerto concreto.
- **OutPorts**: envía varios bytes a través de un puerto concreto.

5.6. MSX-DOS

Las funciones dedicadas al sistema operativo de disco se dividen en varios módulos. El archivo cabecera **MSX_FUSION.h** detalla la estructura de datos del FCB además de las funciones para operar con ella:

- **fcb_open**: abre un fichero.
- **fcb_close**: cierra un fichero previamente abierto.
- **fcb_create**: crea un fichero nuevo.
- **fcb_read**: lee un número indicado de bytes del fichero abierto.
- **fcb_write**: escribe un número indicado de bytes en el fichero abierto.
- **Exit**: finaliza la ejecución del programa devolviendo el control al sistema operativo.

Del archivo cabecera **IO.h**:

- **GetOSVersion**: devuelve la versión del MSX-DOS.
- **GetDisk**: devuelve el número identificador de la unidad de disco.
- **Open**: abre un fichero en el modo indicado.
- **Close**: cierra un fichero previamente abierto.
- **Read**: lee un número indicado de bytes de un fichero abierto.
- **Write**: escribe un número indicado de bytes en un fichero abierto.
- **Create**: crea un fichero con un nombre dado.
- **Remove**: elimina un fichero del disco.

6 . Bomb Jack

En este capítulo se aborda el proyecto-aprendizaje basado en el desarrollo y programación de una versión para ordenadores MSX de primera generación del mítico videojuego *Bomb Jack* de Tehkan. Para ello se sigue la metodología basada en el modelo clásico o en cascada del ciclo de vida del *software* ya definida en la estructura de desglose de trabajo planificada para esta tarea.

El capítulo se ha dividido en cuatro secciones. En cada una de ellas se desarrolla una de las fases del modelo clásico de ciclo de vida del *software*. La primera corresponde con la fase de *análisis*, que tratándose de un videojuego ya existente, se fundamentará en investigar sobre la descripción y documentación de dicho videojuego y sobre la existencia de versiones anteriores para ordenadores MSX. El segundo capítulo corresponde con la fase de *diseño*, donde se definen los elementos que compondrán el documento de diseño del videojuego: historia, guión, arte conceptual, sonido, mecánica de juego y la manera en la que interactuarán jugador y videojuego a través de la máquina. El tercer capítulo tratará la fase de *implementación* en base al diseño y el capítulo cuarto, la fase de *pruebas* donde se comprueba la corrección del *software* desarrollado así como de su integración y funcionamiento en ordenadores MSX-1 reales.

Las fuentes bibliográficas consultadas para su elaboración han sido [35][36][37][38][39][40][41][42][43][44].

6.1. Investigación y preparación

6.1.1. Descripción

Bomb Jack es un videojuego creado el año 1984 por la compañía japonesa Tehkan como máquina recreativa (arcade) para los salones de juego. Pertenece al género "plataformas" y tiene una representación gráfica en dos dimensiones. Presenta dos modalidades de juego: un jugador y dos jugadores alternándose la partida de juego.

Dado el éxito alcanzado por el videojuego en los salones de juego, se realizaron versiones para los ordenadores domésticos de la época: Amstrad CPC, Atari ST, Commodore 16, Commodore 64, Commodore Amiga, NEC PC-8801 y ZX Spectrum. También se realizó la versión para las videoconsolas: Game Boy y SG-1000.

Con posterioridad, se han realizado versiones para videoconsolas más modernas: PS2 y Xbox, para ordenadores actuales[52] con sistemas operativos Linux, Windows o MacOSX realizadas por aficionados. También, realizadas por aficionados, se han desarrollado versiones para ordenadores clásicos que no aún no disponían de versión propia: Atari XL y MSX.

6.1.2. Antecedentes

En el año 2004 aparecieron dos versiones *homebrew* del videojuego *Bomb Jack* para ordenadores MSX. Una de ellas para ordenadores MSX-1 realizada por José Lucio, conocido como "Slotman", de *Icon games*. Esta versión[53] es una conversión directa de la versión realizada para la videoconsola SG-1000 de SEGA. La otra versión[54] fue realizada para ordenadores MSX-2 por el grupo de desarrollo español *Kralizec*. [44]

La Figura 6.1 muestra la versión del videojuego para la máquina arcade original, la Figura 6.2, la versión realizada para ordenadores MSX-1 y la Figura 6.3, la realizada para ordenadores MSX-2.

Figura 6.1: Bomb Jack versión arcade

Figura 6.2: Bomb Jack versión MSX-1

Figura 6.3: Bomb Jack versión MSX-2

Si bien ya existe una versión para ordenadores MSX-1, esta, al tratarse de una conversión directa de la versión de la videoconsola SG-1000 y no de una versión nativa para ordenadores MSX-1, no aprovecha el potencial del *hardware* del MSX-1 ni tampoco puede decirse estrictamente que el MSX-1 tenga una versión propia. Por lo que la versión a desarrollar en este proyecto-aprendizaje podría cubrir esta carencia a la par que trata de exprimir el potencial del *hardware* del MSX-1.

52 http://www.pouet.net/prod.php?which=83722

53 Versión para MSX-1: http://www.icongames.com.br/msxfiles/bjack/index.htm

54 Versión para MSX-2: https://www.generation-msx.nl/software/kralizec/bomb-jack/3446/

6.2. Diseño

El jugador controla al superhéroe Jack que tiene por misión salvar el mundo recolectando, a lo largo de diversos escenarios, las 24 bombas que hay distribuidas por la pantalla antes de que estas exploten y sin ser eliminado por algún enemigo.

Para esta nueva versión del videojuego se va a partir de la versión original y su diseño, aunque para algunos elementos se tomará como referencia principal la versión de MSX-2.

6.2.1. Entrada y salida

6.2.1.1. Teclado y palanca de juego (mecánicas de juego)

El jugador interactuará con el videojuego controlando el avatar del personaje Jack durante la partida, o bien a través de teclado, mediante los cursores y la barra espaciadora, o bien a través de la palanca de juego y el botón principal de disparo. Con ellos se controlará a Jack de la siguiente forma:

- Presionar el **botón de disparo** o la **barra espaciadora** mientras Jack está sobre una plataforma o el suelo hará que salte. Si se presiona una vez mientras Jack está saltando, se detendrá el salto. Si, por contra, se presiona repetidamente, Jack descenderá lentamente al suelo.
- Presionar **izquierda** o **derecha** hará que Jack se mueva en esa dirección, tanto si está corriendo, cayendo o saltando.
- Presionar **abajo**, si Jack está saltando, le hará caer a la parte inferior de la pantalla más rápidamente.
- Presionar **arriba** junto con el **botón de disparo** o la **barra espaciadora** hará que Jack salte más alto de lo normal, hacia la parte superior de la pantalla.

Además, presionar la tecla *escape* permitirá abortar la partida en cualquier momento, volviendo al menú de inicio.

6.2.1.2. Acceso a disco

Con el fin de disponer de la mayor cantidad de memoria RAM posible, se hará uso del almacenamiento secundario de disco para guardar la información relativa al videojuego (gráficos, músicas, diseño de niveles...) y se cargará en memoria principal sólo aquellos datos necesarios para el inicio del videojuego o el nivel de la partida a jugar. Por tanto, se accederá a disco en estas situaciones:

- Cargar la *pantalla de carga* que haga transparente al jugador la inicialización de las estructuras de datos del videojuego y su posterior carga en memoria.
- Cargar los datos comunes en todo el videojuego en las estructuras de datos del videojuego (*sprites hardware*, tabla de puntuaciones y efectos sonoros).
- Cargar la pantalla del menú de inicio.
- Cargar en las estructuras de datos aquellos datos relativos al nivel a jugar (gráficos fondo y modelo de las plataformas, distribución de las plataformas y bombas y música).

6.2.1.3. Gráficos

La versión original tiene una distribución de pantalla vertical, repartiendo los elementos que componen el interfaz gráfico en la parte superior e inferior de la pantalla. La versión para MSX-1 mantiene la configuración original pero sobre una distribución de pantalla horizontal, acomodándola así, a la resolución gráfica del MSX. Por contra, la versión realizada para MSX-2 mantiene la distribución de pantalla vertical para mantener las proporciones de la versión original pero distribuye los elementos del interfaz gráfico de usuario en la parte derecha de la pantalla. La versión a desarrollar seguirá este mismo diseño en la distribución de los elementos de la interfaz gráfico por la pantalla de los gráficos.

PANTALLA DE CARGA, MENÚ Y FONDO DE LOS NIVELES

La *pantalla de carga* estará basada en la carátula de las versiones realizadas para ordenadores domésticos de 8 y 16 bits por la empresa desarrolladora Elite (véase la Figura 6.4)[55].

La *pantalla de menú de inicio* se basará en la versión realizada para MSX-2, que a su vez lo hace de la versión original, y que corresponde con el rótulo del nombre del videojuego, siendo la letra 'O' el dibujo de una bomba en color negro y el resto de letras de color rojo y naranja sobre un fondo con tonalidades de color azul que se reparten en forma de rayos de luz con el foco de inicio en el centro (véase la Figura 6.5).

Las *pantallas de fondo de los niveles* se corresponden con cinco escenarios que se repiten siempre en el mismo orden:

1. **Egipto**: la gran esfinge de Guiza y la pirámide de Kefrén.
2. **Grecia**: la Acrópolis de Atenas.
3. **Alemania**: el castillo de Neuschwastein.
4. **Nueva York**: vista del *skyline* de los rascacielos de la ciudad.

55 Fuente: https://www.mobygames.com/game/bomb-jack/cover-art/gameCoverId,40233/

5. **Los Ángeles**: vista nocturna desde un mirador. Tiene la peculiaridad de ser el único fondo de pantalla sobre el que no hay ninguna plataforma.

Las *pantallas de fondo de los niveles* estarán basadas en los gráficos de la versión original (véase la Figura 6.6)[56].

Figura 6.4: Carátula videojuego Bomb Jack versión Commodore 64

Figura 6.5: Pantalla menú inicio (MSX-2)

Figura 6.6: Escenarios (fondos de pantalla de los niveles) versión original

SPRITES (AVATAR, ENEMIGOS E ÍTEMS RECOLECTABLES)

Los *sprites* de las entidades del videojuego estarán basados en los gráficos de la versión original (véase la Figura 6.7)[57].

56 Fuente: https://www.spriters-resource.com/arcade/bombjack/sheet/86129/

57 Fuente: https://www.spriters-resource.com/arcade/bombjack/sheet/50270/

Figura 6.7: Tabla de sprites (versión original)

INTERFAZ GRÁFICO

El interfaz gráfico de usuario lo componen los elementos que proporcionan al jugador información relativa a la partida en curso. Dichos elementos son:

1. Puntuación actual del nivel: muestra la puntuación que el jugador lleva durante el nivel al que está jugando. Los puntos se obtienen según estos conceptos:
    - *Coger bomba apagada*: 100 puntos.
    - *Coger bomba encendida*: 200 puntos.
    - *Coger moneda 'B'*: 1.000 puntos e incrementa el multiplicador en 1.
    - *Coger moneda 'E'*: 3.000 puntos e incrementa el número de vidas en 1.
    - *Coger moneda 'S'*: 5.000 puntos y pasa al nivel siguiente.
    - *Coger moneda 'P'*: dependiendo del color que tenga al ser cogida proporciona distintos puntos (azul: 100 puntos, rojo: 200 puntos, rosa: 300 puntos, verde: 500 puntos, azul claro: 800 puntos, amarillo: 1.200 puntos, gris: 2.000 puntos). El color de la moneda varía progresivamente. Tras el color gris, el color de la moneda vuelve al azul inicial repitiendo el ciclo.
    - *Matar enemigos*: el jugador sólo puede matar durante un periodo de tiempo reducido que se inicia al coger una moneda 'P'. El primer enemigo en morir proporciona 100 puntos extras, el segundo 200 puntos, el tercero 300

puntos, el cuarto 500 puntos, el quinto 800 puntos, el sexto 1.200 puntos y los restantes 2.000 puntos por igual.

- *Bonus*: dependiendo de la cantidad de bombas encendidas recogidas se obtienen puntos extras (20 bombas: 10.000 puntos, 21 bombas: 20.000 puntos, 22 bombas: 30.000 puntos y 23 bombas: 50.000 puntos).

2. Multiplicador de puntuación (x): Al finalizar el nivel, serán multiplicados por su valor los puntos alcanzados en el nivel. Su rango de valores es de 1 a 5. Empieza con el valor '1' que se incrementa cada vez que el jugador coge una moneda 'B'. Cuando el multiplicador está en su valor máximo, las monedas 'B' dejan de aparecer. Si el jugador pierde una vida, el multiplicador vuelve a su valor mínimo.
3. Barra de energía: esta barra flanquea al multiplicador de puntuación y se incrementa cada vez que el jugador recolecta bombas. Se incrementa más rápidamente si se recogen bombas que están encendidas. Cuando alcanza su tamaño máximo aparece en pantalla la moneda 'P' y la barra de energía vuelve a su tamaño mínimo.
4. Número de vidas: muestra el número de vidas restantes.
5. Nivel actual: muestra el nivel del juego al que se está jugando.
6. Puntuación máxima de la partida: muestra la puntuación acumulada a lo largo de los niveles jugados durante la partida en curso. Cuando durante la partida el jugador supera la puntuación máxima, esta parpadea.

La Figura 6.8 muestra la distribución del interfaz gráfico de la versión original. La Figura 6.9 muestra la del interfaz gráfico de la versión realizada para MSX-2. La versión para MSX-1 a desarrollar seguirá la distribución del interfaz de la versión para MSX-2, dado que, a su vez, fue la distribución que siguieron las conversiones a ordenadores domésticos de la época.

Al finalizar cada nivel, se mostrará en pantalla un resumen de los elementos del interfaz gráfico donde se desglose la puntuación alcanzada.

Figura 6.8: Interfaz gráfico versión original

Figura 6.9: Interfaz gráfico versión MSX-2

6.2.1.4. Audio

El apartado sonoro del videojuego original lo componen una serie efectos sonoros y melodías que suenan como retroalimentación auditiva ante algunas acciones importantes del jugador o que acompañan cada nivel amenizando la partida.

EFECTOS DE SONIDO

Estos son los efectos sonoros que suenan tras alguna acción importante:

1. Disparo de inicio de la partida.
2. Sirena cuando aparece la moneda 'P'.
3. Zumbido al presionar el botón de salto.
4. Tintineo al obtener una vida extra.
5. Zumbido tras la pérdida de una vida.
6. Tintineo al aparecer las monedas "B", "E" y "S".

MÚSICA (BANDA SONORA)

Existen dos tipos de música en el videojuego que conforman su banda sonora: la que indica alguna situación importante en el videojuego y las melodías que acompañan los escenarios de los niveles. La banda sonora la compone un total de 8 músicas:

1. Título (menú inicio).
2. *Ringo no Mori no Koneko Tachi* (escenarios de Egipto y Nueva York).
3. *Lady Madonna* (escenarios de Grecia y Los Ángeles).
4. Desconocida (escenario de Alemania).
5. Cogida una moneda 'P': sirve de referencia para cronometrar la duración de los efectos de la moneda.
6. Nivel superado.
7. Puntuación total al finalizar el nivel.
8. Fin de la partida.

Dos de las melodías que acompañan a los escenarios del juego tienen licencia de fuentes populares. La melodía *Ringo no Mori no Koneko Tachi* es la canción final de la serie de dibujos animados japonesa *Spoon Obasan* y la melodía *Lady Madonna* es de los Beatles.

6.2.2. Videojuego

6.2.2.1. Entidades del videojuego

- Jack: es el avatar del jugador que representa a un superhéroe que puede saltar o dar grandes saltos y flotar en el aire. Tiene por misión recolectar todas las bombas del escenario. Jack pierde una vida cada vez que entre en contacto con algún enemigo. Puede matar a los enemigos tras coger una moneda 'P' ya que los convierte en monedas recolectables. Aparece, de inicio, siempre en el centro de la pantalla. Cambia de color a parpadeos entre azul y blanco al coger una moneda 'P'.

- Bombas: Al inicio de cada nivel hay repartidas un total de 24 bombas, todas ellas apagadas. Tras coger Jack la primera bomba o una bomba encendida, se enciende otra de las bombas restantes. Simultáneamente sólo hay una bomba encendida. El orden en el que se encienden las bombas está prefijado y es siempre el mismo. Para obtener mayores puntuaciones deben recogerse las bombas por el orden en el que se encienden. Las bombas encendidas nunca llegan a explotar.

- Enemigos: existen dos tipos de enemigos principales y 5 secundarios. Los enemigos secundarios aparecen después de que uno de los tipos de enemigos principales choque contra el suelo. Cada tipo de enemigo tiene su propia rutina de movimientos e inteligencia artificial.

    - **Pájaros**: enemigos principales de color blanco que aparecen al inicio del nivel en cualquiera de las cuatro esquinas de la pantalla. El número de enemigos pájaro que aparecen de inicio se calcula según la fórmula: (nº_nivel / 10) + 1. Se mueven horizontal o verticalmente siempre en línea recta hacia Jack. Son capaces de rodear las plataformas.

      Dificultad: su velocidad de movimiento es lenta y se incrementa a medida que se avanzan niveles.

    - **Momias**: enemigos principales de color gris que aparecen desde la parte alta de la pantalla. Se mueven a izquierda o derecha, dependiendo de donde se encuentre Jack, caminando sobre las plataformas. Si caen de ellas y llegan al suelo se transforman en uno de los 5 enemigos secundarios.

      Dificultad: en los primeros niveles caminan por las plataformas procurando no caer de ellas. A medida que se avanza en niveles, buscan caer directamente al suelo.

- **Esferas**: enemigos secundarios de color gris. Se mueven rebotando verticalmente sin control acercándose hacia la posición de Jack.

  

  Dificultad: su velocidad de movimiento es lenta y se incrementa a medida que se avanzan niveles.

- **Orbes**: enemigos secundarios de color negro. Se mueven rebotando por toda la pantalla buscando a Jack.

  

  Dificultad: su velocidad de movimiento es lenta y se incrementa a medida que se avanzan niveles.

- **Ovnis**: enemigos secundarios de color gris. Se desplazan sobre un mismo eje (horizontal o vertical) hasta rebotar con una pared o plataforma, entonces aceleran su velocidad buscando a Jack, siempre rectilíneamente. Si no lo alcanzan siguen su camino hasta rebotar nuevamente.

  

  Dificultad: su velocidad de movimiento es lenta mientras están lejos de Jack. Cuando están próximos a Jack aceleran vertiginosamente su velocidad.

- **Cuernos**: enemigos secundarios de color gris. Se desplazan por la pantalla buscando a Jack por el camino más corto. Tienen la capacidad de cambiar de dirección en cualquier momento.

  

  Dificultad: su velocidad de movimiento es lenta y se incrementa a medida que se avanzan niveles.

- **Satélites**: enemigos secundarios de color gris que van rebotando por toda la pantalla a velocidad constante. Su velocidad de movimiento es lenta y carece de inteligencia. Nunca persiguen a Jack.

  

  Dificultad: la falta de atención por carencia de inteligencia es una sorpresa a pesar de su previsibilidad.

- Monedas: aparecen aleatoriamente en el tiempo o tras alcanzarse un hito importante. Existen 5 tipos de monedas recolectables:

  - **B** (Bonus): incrementa en 1 el *multiplicador de puntuación*. Cae desde la altura máxima de la pantalla y se desplaza en dirección horizontal al posicionarse sobre una plataforma o suelo. Es la moneda que aparece con mayor frecuencia. Es de color azul. Si el multiplicador está en su máximo valor, la moneda no aparece.

    

  - **P** (Power): aparece cuando la *barra de energía* llega a su máximo desde la ubicación de la bomba que la colma y va rebotando por toda la pantalla. Varía su color (azul, rojo, rosa, verde, azul claro, amarillo y gris), cada color determina una puntuación distinta. La melodía del escenario que suena es sustituida por un efecto sonoro a modo de alarma de alerta hasta ser cogida.

    

- **E** (Extra): proporciona una vida extra al jugador. Es de color verde. Aparece cuando Jack ha matado un número concreto de enemigos al recolectarlos como monedas tras coger Jack una moneda 'P'.

- **S** (Special): aparece de forma aleatoria en pocas ocasiones. Es de color rojo y hace que el jugador acceda al nivel siguiente.

- **Oro**: sustituyen a los enemigos que hay en pantalla en el momento que el jugador coge una moneda 'P'. Son de color amarillo y permanecen en la pantalla mientras suena la música de moneda 'P' recolectada. Tras finalizar la música, las monedas 'Oro' vuelven a convertirse en los enemigos.

6.2.2.2. Niveles

Existen un total de 63 niveles que han de ser superados para finalizar con éxito el videojuego. Cada nivel consta de dos tipos de elementos que lo definen:

1. Los *escenarios* representados en las *pantallas de fondo del nivel* —existen un total de 5 escenarios—. El escenario de Los Ángeles (L.A.) aparece en los niveles múltiplos de 5 durante los 30 primeros niveles —sólo aparece un total de 6 veces —. Una vez superado el nivel 30, dicho escenario no vuelve a aparecer a lo largo de la partida.
2. Los *arreglos escénicos*, es decir, la disposición y diseño de las plataformas, de las bombas y el orden en el que estas se encienden. Existe un total de 16 configuraciones distintas que se repiten alternando su escenario. En el escenario de Los Ángeles se mantiene el arreglo escénico de las bombas pero no así el de las plataformas ya que carece de ellas.

La Figura 6.10[58] muestra los 16 arreglos escénicos existentes numerados con las letras 'A' a 'P' y el orden en el que se encienden las bombas en dicho arreglo. El arreglo 'E' sólo ocurre en los niveles 5 y 15. La Tabla 6.1[59] muestra la combinación, por nivel, del escenario y el arreglo escénico.

Nivel	Arreglo escénico	Escenario	Nivel	Arreglo escénico	Escenario	Nivel	Arreglo escénico	Escenario	Nivel	Arreglo escénico	Escenario
1	A	Egipto	17	O	Grecia	33	M	Nueva York	49	B	Grecia
2	B	Grecia	18	P	Alemania	34	N	Egipto	50	I	Nueva York
3	C	Alemania	19	B	Nueva York	35	G	Grecia	51	J	Egipto
4	D	Nueva York	20	N	L.A.	36	I	Nueva York	52	N	Egipto
5	E	L.A.	21	K	Egipto	37	L	Grecia	53	K	Grecia
6	F	Egipto	22	G	Grecia	38	O	Alemania	54	M	Nueva York
7	G	Grecia	23	D	Alemania	39	F	Egipto	55	G	Grecia
8	H	Alemania	24	L	Nueva York	40	F	Egipto	56	A	Egipto
9	I	Nueva York	25	M	L.A.	41	H	Alemania	57	O	Grecia
10	A	L.A.	26	P	Alemania	42	P	Alemania	58	F	Egipto
11	J	Egipto	27	H	Alemania	43	G	Grecia	59	H	Alemania
12	K	Grecia	28	O	Grecia	44	D	Nueva York	60	C	Alemania
13	L	Alemania	29	J	Egipto	45	K	Grecia	61	P	Alemania
14	M	Nueva York	30	A	L.A.	46	C	Alemania	62	G	Grecia
15	E	L.A.	31	F	Egipto	47	L	Alemania	63	D	Nueva York
16	N	Egipto	32	C	Alemania	48	I	Nueva York			

Tabla 6.1: Niveles: combinación de arreglos escénicos y escenarios.

58 Fuente: https://strategywiki.org/wiki/Bomb_Jack/Walkthrough

59 Fuente: https://strategywiki.org/wiki/Bomb_Jack/Walkthrough

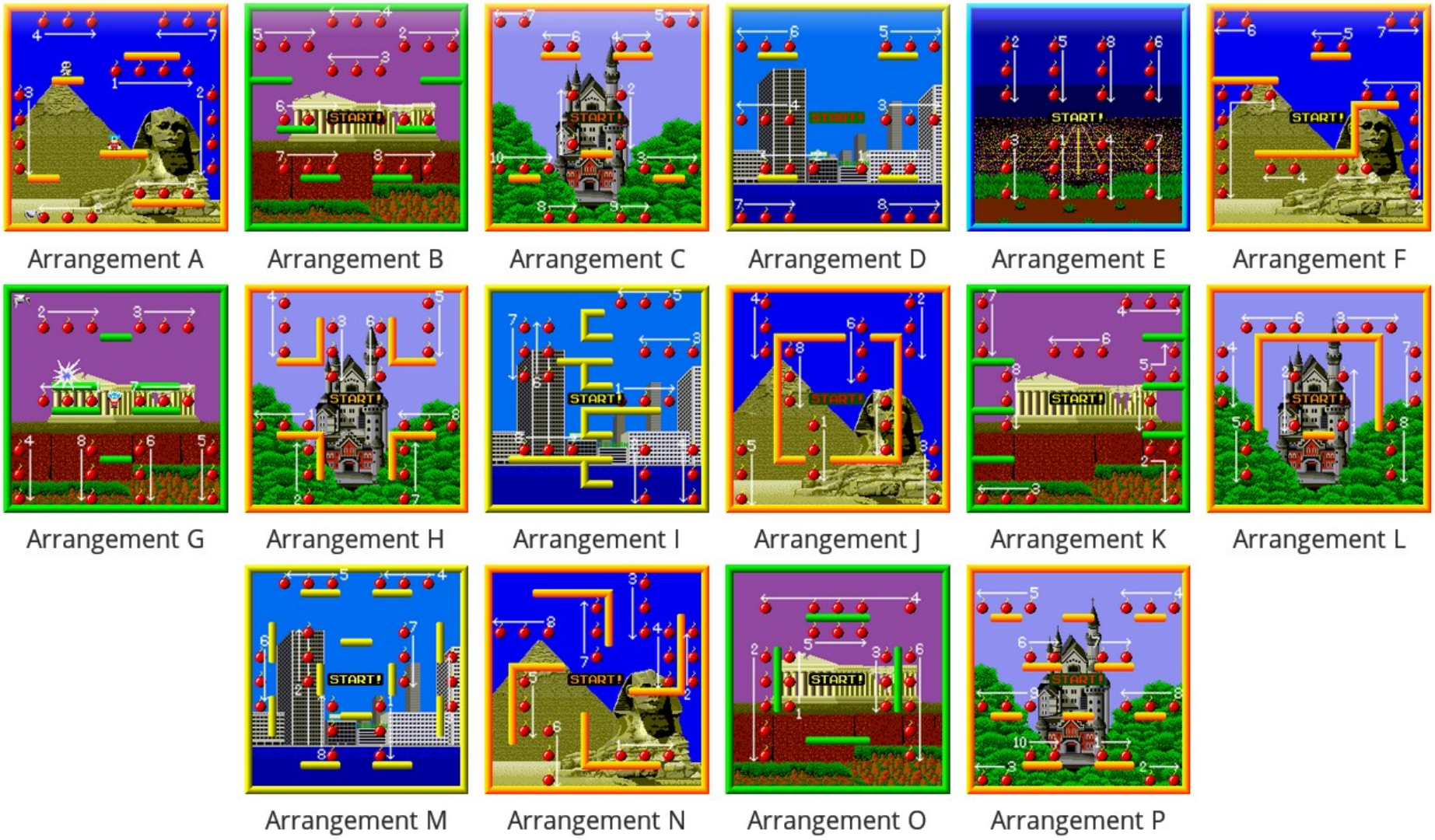

Figura 6.10: Niveles: arreglos escénicos (A..P) y orden encendido de las bombas

6.2.2.3. *Menú de inicio*

El menú de inicio constará de dos pantallas: la ya comentada *pantalla de menú de inicio* y la *pantalla del salón de la fama* (*ranking* de puntuaciones y nombres de jugadores) que se mostrarán en bucle —transición controlada por un temporizador— mientras no se inicie una partida nueva. La *pantalla del salón de la fama* mostrará las puntuaciones obtenidas a lo largo de las sucesivas partidas que se juegen mientras el videojuego esté en funcionamiento. De inicio, este salón de la fama estará prefijado y será siempre el mismo al inicio del videojuego, es decir, no serán persistentes las nuevas puntuaciones que entren en él.

La Figura 6.11 muestra el diagrama de flujo del algoritmo que rige el comportamiento del menú de inicio.

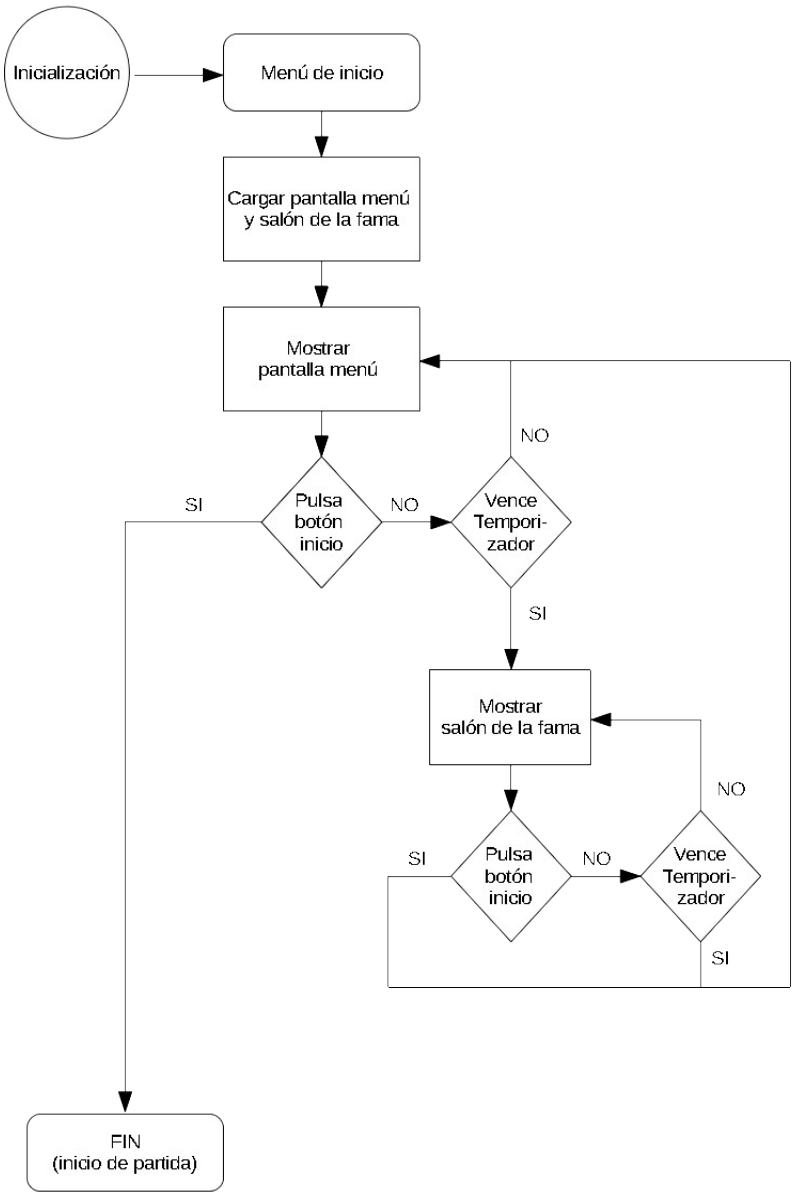

Figura 6.11: Diagrama de flujo del algoritmo del menú de inicio

6.2.2.4. Bucle principal

Constituye el esqueleto principal del videojuego, es decir, el motor del mismo. Combina, de forma repetitiva, las tareas invisibles al jugador: capturar la entrada, realizar los cálculos de simulación física, detectar las colisiones..., y la de mostrar en pantalla todos los cambios producidos tras el resto de tareas.

El Figura 6.12 muestra el algoritmo con la secuencia de tareas diseñadas que constituirán el bucle principal del videojuego.

```
Inicialización estructuras principales
Carga y visualización pantalla de carga
Menú Inicio (bucle)
Mientras (continuar y existan niveles)
        Iniciacialización del nivel
        Mientras no finalización del nivel
                Obtener entrada jugadores (teclado, palanca de juego)
                Actualizar avatar/Jack
                Actualizar tiempos (monedas 'B' y 'S')
                Inteligencia artificial enemigos/resto items recolectables
                Simulación física (cálculo de las nuevas posiciones enemigos)
                Actualizar estado de las entidades del videojuego
                Detección de colisiones
                Renderizar la pantalla
                Realimentación sensorial al jugador (efectos sonoros y música)
                Pausa hasta tiempo final de fotograma
        Cierre/finalización de nivel
        ¿Continuar al nivel siguiente?
Cierre/finalización partida
```

Figure 6.12: Algoritmo bucle principal del juego

6.2.2.5. Finalización

La partida finalizará por dos motivos:

1. La consecución exitosa de los 63 niveles que componen el videojuego. En este caso, aparecerá un mensaje de texto de felicitación indicando la consecución del mismo y, posteriormente, se procederá a la introducción del nombre del jugador en el salón de la fama.

2. La pérdida de todas las vidas que dispone el jugador. Si el jugador ha alcanzado una puntuación mínima, introducirá su nombre en el salón de la fama.

Tras la introducción del jugador en el salón de la fama, el videojuego volverá al menú de inicio.

La Figura 6.13 ilustra el diagrama de la máquina de estados finitos por la que transitará la ejecución del videojuego. La significación de las entradas es la siguiente:

- *salir*: abandonar el videojuego volviendo al MSX-DOS o abandonar la partida en curso volviendo al menú de inicio.
- *timeOut*: vencimiento del tiempo de espera para alternar la pantalla de menú de inicio y el Salón de la Fama.
- *botónDisparo*: pulsación de la barra espaciadora del teclado o del botón de disparo de la palanca de juegos.
- *gameOver*: fin de la partida al no disponer el jugador de más vidas.
- *finNivel*: consecución exitosa del nivel al que se está jugando.
- *entrarEnHOF*: indica que el jugador ha obtenido la puntuación necesaria para escribir su nombre en el *Salón de la Fama*.

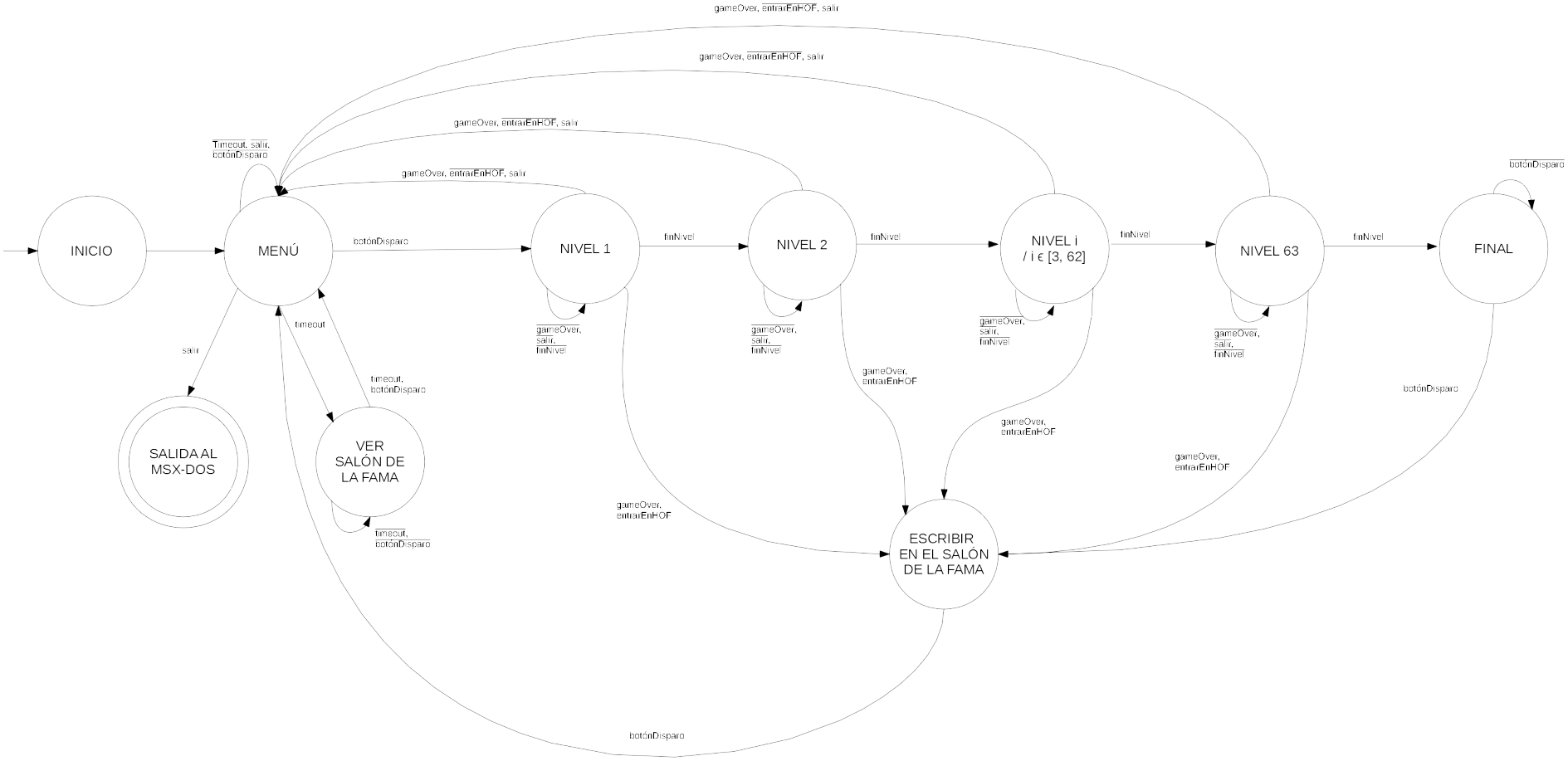

Figura 6.13: Máquina de estados del videojuego

6.3. Implementación

6.3.1. Documentación

En este apartado se va a documentar las decisiones adoptadas a la hora de implementar los distintos elementos del videojuego, describir las estructuras de datos escogidas para modelarlos y las funciones que operan con dichas estructuras de datos. También se nombrarán las funciones usadas que pertenecen a ejemplos demostrativos que acompañan a la librería Fusion-C.

Se ha escogido el modo gráfico 2, de alta resolución, para desarrollar el videojuego.

La información relativa al avatar y el interfaz gráfico se ha modelado a través de una variable de tipo registro (*player* y *gui*). El resto de entidades del videojuego: bomba, moneda y enemigo; y el Salón de la Fama, se han modelado a través de una lista de variables de tipo registro implementada sobre un vector (*lista_bombas, listaMonedas, listaEnemigos* y *HallOfFame*). En los siguientes apartados se describe con mayor detalle la implementación de estas variables de tipo registro.

Para la información a mostrar en pantalla se ha optado por contar con *buffers* de memoria específicos en memoria RAM de las tres tablas en las que se divide la memoria VRAM (*tablaPatrones, tablaColor, tablaNombres* y *buffer_tablaNombres* —este último se ha usado como un doble *buffer* de la Tabla de Nombres—), aplicando primero las modificaciones sobre la memoria *buffer* y volcando posteriormente su contenido sobre la tabla correspondiente de la memoria VRAM cuando ha sido necesario.

La información de la distribución de plataformas y bombas a lo largo del nivel en juego se ha modelado a través de una matriz (*Mundo*) que se ha usado principalmente para la detección de colisiones entre estos elementos y el avatar, enemigos o monedas.

Se ha hecho uso de algunas funciones implementadas por segundas personas y que pertenecen a varios ejemplos que acompañan a la librería Fusion-C. Estas funciones son:

- *char FT_wait (int cicles)*: pausa el microprocesador durante un determinado número de ciclos.
- *char FT_RandomNumber (char a, char b)*: proporciona un número aleatorio dentro de un intervalo dado.
- *void FT_errorHandler (char n, char *name)*: manejador de errores en la lectura en la unidad de disco.
- *void FT_SetName (FCB *p_fcb, const char *p_name)*: ayuda a establecer el nombre del fichero con el que operar en la unidad de disco.

- *int FT_LoadData (char *file_name, char *buffer, int size, int skip)*: lee el contenido de un fichero. Usado para cargar en memoria el banco de efectos sonoros.
- *void FT_CheckFX (void)*: comprueba si hay algún efecto sonoro reproduciéndose.

6.3.2. Inicialización y pantalla de carga

INICIALIZACIÓN

En esta parte se inicializan el dispositivo de sonido (PSG) para la reproducción de efectos de sonido, el dispositivo gráfico (VDP), fijando el modo gráfico, los colores de pantalla y el modo y tamaño de los *sprites* (16 x 16). También se inicializan las variables y estructuras de datos que dan soporte a los elementos comunes a todos los niveles: efectos de sonido, *sprites* y *Salón de la Fama*.

Se ha implementado una función auxiliar para la creación e inicialización del Salón de la Fama:

- void crearHOF (t_HallOfFame * HallOfFame).

PANTALLA DE CARGA

Como fuente originaria para la pantalla de carga se ha tomado la imagen de la carátula que incorporaba la versión para Commodore Amiga [42]. El proceso seguido ha sido el siguiente:

- Obtención de la imagen de la versión de Comodore Amiga.
- Manipulación de la imagen con la aplicación GIMP: recorte de la imagen para eliminar las bandas negras, escalado proporcionado al ancho de 256 *píxeles*, conversión a modo indexado con la paleta de colores de MSX-1, ampliación del tamaño del lienzo a una altura de 192 *píxeles* manteniendo el ancho a 256 *píxeles* añadiendo, nuevamente, las barras superior e inferior negras. Haciendo uso de la rejilla, se corrigen los colores para evitar los fallos por la característica de *attribute clash*. Exportación a formato de *bitmaps* PNG.
- Con la aplicación MSX Tiles Devtools se carga la imagen exportada con GIMP y se configuran las opciones de conversión a lenguaje de programación a 'C', tanto de las pestañas *tileset* y *screen map*, y se obtienen los valores de la imagen correspondientes a la tabla de patrones, de colores y de nombres.
- Se crea un pequeño programa en C (saveSC2.c) *ex profeso* que guarda en un fichero los valores en bruto obtenidos (loading.SC2), prescindiendo de las cabeceras y otros datos de los ficheros binarios de imagen SC2. Se reduce, así, el tamaño del fichero imagen resultante en 3 KB, pasando a ocupar 13 KB en lugar de los 16 KB que ocuparía el formato binario SC2. Se busca, con esto, reducir el tiempo de lectura de disco.

Las funciones implementadas para la carga y visualización de la pantalla de carga son:

- void loadSC2 (char *filename).
- void mostrarSC2 (void).

6.3.3. Bucle menú de inicio

GRÁFICOS

La *pantalla de menú de inicio* se basa en la versión realizada para MSX-2 (véase la Figura 6.5), que a su vez lo hace de la versión original y que corresponde con el rótulo del nombre del videojuego, siendo la letra 'O' el dibujo de una bomba en color negro y el resto de letras de color rojo y naranja sobre un fondo con tonalidades de color azul que se reparten en forma de rayos de luz con el foco de inicio en el centro. Bajo el rótulo del nombre del videojuego se halla la frase indicativa de inicio de partida. Contiene, además, la referencia a la autoría del juego original y la de la versión para MSX-2.

La *pantalla del salón de la fama* contiene, en la parte superior, los letreros indicativos de la puntuación máxima existente y de la puntuación obtenida por el jugador. En la parte central aparece el esqueleto de la lista con las 8 posiciones que conforman el *Salón de la Fama* y el letrero del nivel máximo alcanzado por esa posición. En la parte inferior aparece la autoría de esta versión flanqueada a ambos lados por dos dibujos del personaje Jack.

Ambas pantallas comparten tabla de patrones y de colores, siendo diferentes la tabla de nombres.

El proceso seguido para la realización de ambas pantallas ha sido el siguiente:

Pantalla de menú de inicio

- Manipulación de la imagen con la aplicación GIMP: recorte del rótulo del videojuego de la imagen de la tabla de *sprites* original (véase la Figura 6.7). Creación de un proyecto de imagen para MSX-1. Pegado del rótulo en el proyecto y escalado del mismo para adaptarlo a las dimensiones del proyecto. Conversión de la imagen a modo indexado y con la herramienta de la rejilla se alinea el rótulo al inicio de una de las rejas. Se retoca el rótulo para minimizar el efecto *attribute clash*. Exportación a formato de *bitmaps* PNG.
- Con la aplicación nMSXtiles se crea el juego de 96 caracteres ASCII que corresponden con los valores comprendidos entre el 32 y el 127. Cada caracter se ubica en la posición del banco de patrones que corresponde con su valor ASCII (en los tres bancos que conforman la tabla de patrones y de colores) para, posteriormente, poder referenciarlos como texto. Además, se crea el resto de elementos que componen ambas imágenes. Se guarda el proyecto en el formato propio de la aplicación.
- Con la aplicación MSX Tiles Devtools se carga el proyecto guardado con nMSXtiles y se configuran las opciones de conversión a lenguaje de

programación C, tanto de las pestañas *tileset* y *screen map*, y se obtienen los valores de la imagen correspondientes a la tabla de patrones, de colores y de nombres.

- Con el programa creado en C (saveSC2.c) se guarda en un fichero los valores en bruto obtenidos (menu.SC2).

Pantalla del Salón de la Fama

- Con la aplicación nMSXtiles, partiendo del proyecto creado para la pantalla de menú, se crea una nueva pantalla (tabla de nombres —*screen*—) y se dibuja, con los patrones de cada banco, la pantalla del Salón de la Fama. Se guarda en un fichero (hof.scr) solamente la pantalla creada, es decir, los valores de la tabla de nombres puesto que comparte la tabla de patrones y de colores con la pantalla de menú de inicio (opción del menú *screen→save as*...).

Las Figura 6.14 muestra, por este orden, la imagen correspondiente a la tabla de patrones y colores compartida por ambas pantallas y el resultado final de ambas.

Figura 6.14: Pantallas bucle menú inicio

MÚSICA / FX

Con la aplicación AYFX Editor se ha creado un banco de efectos sonoros con algunos de los que vienen en la biblioteca que acompaña a la aplicación. El primer efecto del banco corresponde con el efecto que sonará al presionar la barra espaciadora o el botón de disparo. El escogido es *noname017.afx* del videojuego *Streets of Rage 2*.

LÓGICA

Se carga en memoria RAM la pantalla de menú de inicio (con las tres tablas) —fichero *menu.SC2*— y se envía a la VRAM. Se guarda una copia de la tabla de nombres de la VRAM en una memoria *buffer* de la RAM (buffer_menu) para alternarla con la tabla de nombres del Salón de la Fama (HOF). Se cargan en la memoria *buffer* de la tabla de nombres los valores correspondientes a la tabla de nombres del Salón de la Fama.

La información que se guarda sobre una entrada en el *Salón de la Fama* se modela a través de una variable de tipo registro que contabiliza la puntuación obtenida, el nombre del jugador (3 letras) y el nivel máximo alcanzado durante la partida.

La intermitencia del texto que indica la acción de inicio de la partida se realiza accediendo directamente a las direcciones de la VRAM donde se ubica la tabla de

colores y se modifican, alternando cada ciertos ciclos, los valores con aquellos relativos a los colores rojo y negro.

Un temporizador regula la alternancia entre ambas pantallas volcando en la VRAM la que toca visualizar.

El bucle del menú de inicio finaliza tras la pulsación del botón/tecla de disparo.

Las funciones relacionadas con las pantallas de *Menú de inicio* y *Salón de la Fama* son:

- void crearHOF (t_HallOfFame *HallOfFame).
- void loadSC2 (char *filename).
- void mostrarSC2 (void).
- void menu (t_HallOfFame * HallOfFame, unsigned int MAX_score, unsigned int PL_score).
- void loadHOF (char *filename).
- void mostrarHOF (t_HallOfFame * HallOfFame, unsigned int MAX_score, unsigned int PL_score).

6.3.4. Niveles

GRÁFICOS

Existen 5 fondos de escenario que van alternándose a lo largo de los niveles. El proceso seguido para la realización cada uno de los fondos ha sido el siguiente:

- Manipulación de la imagen con la aplicación GIMP: recorte del escenario correspondiente de la imagen original (véase la Figura 6.6). Creación de un proyecto de imagen para MSX-1. Pegado del escenario en el proyecto y escalado del mismo para adaptarlo a la dimensión vertical del proyecto, justificándolo al margen izquierdo y dejando la parte derecha excedente para almacenar elementos posteriores a pintar sobre el escenario (interfaz de usuario). Conversión de la imagen a modo indexado. Se retoca la imagen para minimizar el efecto *attribute clash*. Exportación a formato de *bitmaps* PNG.
- Con la aplicación nMSXtiles se crean los patrones y colores correspondiente a las bombas y plataformas que serán pintadas sobre el fondo, los elementos que conformaran la interfaz de usuario e inicio y fin de partida. Dichos patrones se almacenan en el espacio excedente de la derecha de la imagen resultante de la acción anterior. Tanto las bombas, en sus estado de apagado como de encendido, y las plataformas se repiten en los tres bancos de las tablas de patrones y colores. Se deja un desplazamiento de 160 patrones entre el patrón que indica el estado de bomba apagada y el que indica el estado de bomba encendida.
- Con la aplicación MSX Tiles Devtools se carga el proyecto guardado con nMSXtiles y se configuran las opciones de conversión a lenguaje de

programación C, tanto de las pestañas *tileset* y *screen map*, y se obtienen los valores de la imagen correspondientes a la tabla de patrones, de colores y de nombres.

- Con el programa creado en C (saveSC2.c) se guarda en un fichero los valores en bruto obtenidos (Egipto.SC2, Grecia.SC2, Alemania.SC2, NY.SC2 y LA.SC2).

La Figura 6.15 muestra el final del proceso para el escenario de Egipto, siendo la parte izquierda de la imagen la correspondiente a los patrones organizados en la tabla de patrones y de colores, y la parte derecha la imagen final a mostrar según la tabla de nombres.

Figura 6.15: Escenario de Egipto: tablas de patrones y colores, e imagen final según la tabla de nombres

LÓGICA

La información relativa al fondo del escenario y de la distribución de los arreglos (bombas y plataformas) del nivel a jugar —véase la Tabla 6.1— se almacena en el vector bidimensional *niveles*. La primera dimensión almacena el escenario a cargar —codificados como de 1 a 5— y la segunda dimensión la configuración de los arreglos —codificados como de 'A' a 'P'—. La posición que ocupan en el vector corresponde con el número del nivel.

La información de cada uno de los arreglos se halla almacenada en un fichero de texto con extensión 'map'. El nombre del fichero indica el tipo de arreglo. El contenido del fichero se estructura en 24 filas y 24 columnas que corresponden con las dimensiones de la matriz *Mundo* que modeliza el nivel y almacena la posición de las plataformas y las bombas. Las plataformas están compuestas por 6 patrones y se han codificado con letras mayúsculas de la 'A' a la 'G'. Las bombas están compuestas por 4 patrones (2 para la parte superior y 2 para la parte inferior). Además de estos 4 patrones, existen otros 4 patrones para la parte superior de la bomba pero con distintos colores de fondo que faciliten su integración con el fondo del escenario. Los 8 patrones se han codificado con letras minúsculas de la 'a' a la 'h'. La relación entre el valor codificado (valor ASCII) y la posición del patrón que ocupa en el banco tiene un desplazamiento de '-9' para las plataformas y de '-73' para las bombas.

La información relativa al orden de encendido de las bombas se halla, también, almacenada en un fichero de texto con extensión 'bmb'. El nombre del fichero indica el arreglo al que pertenece. El contenido del fichero se estructura en 24 filas de 2

columnas —a razón de una fila por bomba—. Cada fila almacena la posición (x,y) que ocupa el primer patrón de la bomba en la matriz *Mundo*. Las posiciones se han codificado con las primeras 24 letras del alfabeto en minúscula. Los patrones que pintan el encendido de la bomba tienen un desplazamiento de '160' patrones con respecto al patrón origen que pinta el apagado de la bomba.

La Figura 6.16 muestra la carga de un nivel, mostrando su escenario y la distribución de los arreglos (plataformas y bombas).

Figura 6.16: Muestra de carga de nivel (escenario y arreglos) y esqueleto de GUI

Las funciones implementadas para la carga y visualización del nivel son:

- void loadSC2 (char *filename).
- void mostrarSC2 (void).
- void loadNivel (byte nivel).
- void loadMap (char *filename, bool pintar_plataformas).
- char loadTracaBombas (byte nivel).

6.3.5. Interfaz de usuario

Siguiendo la distribución del la interfaz de la versión de MSX-2, la parte derecha de la pantalla muestra la información relativa a la puntuación máxima existente, el multiplicador de puntuación, la barra de energía, la puntuación actual del jugador durante la partida, el número de vidas restantes y el nivel al que se está jugando. La barra de energía tiene 4 estados. El estado inicial no muestra ningún dibujo de la barra y, a medida que está avanza de estado, se muestra la barra hasta rodear el multiplicador de puntuación.

La información que muestra el interfaz se modela a través de una variable de tipo registro que contabiliza la puntuación máxima, la puntuación alcanzada durante el nivel de juego, el valor del multiplicador, de la barra de energía y el nivel actual. Tanto la puntuación actual del jugador como el número de vidas restantes se contabilizan en la variable-registro del jugador.

La Figura 6.16 muestra el esqueleto del interfaz de usuario con el estado de la barra de energía al máximo.

La función implementada para la visualización del interfaz gráfico es:

- void mostrarGUI (unsigned int PlScore, byte vidas).

6.3.6. Sprites

El tamaño de *sprites* seleccionado hace que sólo pueda haber en memoria un total de 64 *sprites* para representar los distintos cuadros de la animación del avatar, de los enemigos, de las monedas y de las bombas. Si se tiene en cuenta que sólo puede haber simultáneamente en pantalla 32 *sprites* y que existe un total de 24 bombas que recoger, harían falta 25 planos sólo para las bombas —24 para dibujar las bombas y uno extra para dibujar el encendido de la bomba—. Esto dejaría muy pocos planos para el resto de elementos móviles y obligaría a que todos los *sprites* fueran monocromo, además de que fácilmente se incumpliría la regla de 5º *sprite* en las filas de bombas que hay en los niveles. Por ello, se trata a las bombas como *sprite* por *software* y no por *hardware,* siendo consideradas como un elemento del fondo y excluyendo su definición de la TABLA GENERADORA DE SPRITES para incluirla en la TABLA GENERADORA DE PATRONES.

Al disponer de los 64 *sprites* para dibujar el resto de ítems móviles, se decide que el avatar sea un *sprite* multicolor, compuesto por 3 *sprites* —uno para cabeza, manos y pies; otro para ojos y capa; y otro para tronco, brazos y piernas—. Tanto los enemigos como las monedas serán representados con *sprites* monocromo para no caer rápidamente en el incumplimiento de la regla del 5º *sprite*.

Tanto el avatar como los enemigos tendrán un máximo de dos cuadros (*sprites*) para su animación, por contra, la animación de las monedas será de tres cuadros.

Se van a representar 12 estados para el avatar: reposo, salto, salto-izquierda, salto-derecha, caída, caída-izquierda, caída-derecha, andar-izquierda, andar-derecha y muerte. Sólo los estados de andar estarán animados por más de 1 cuadro (*sprite*). Esto hace que se necesiten en total 36 *sprites* para representar al avatar del jugador, lo que deja 28 *sprites* para enemigos y monedas.

Respecto a los enemigos, los dos principales tendrán 2 estados a animar según sea su orientación (a izquierda o a derecha), siendo, por tanto, necesario 8 *sprites* para ellos. Los cinco enemigos secundarios sólo tendrán un único estado a animar, por lo que se necesitarán 10 *sprites* para ellos. Esto hace un total de 18 *sprites* para los enemigos.

Quedan, pues, 10 *sprites* para las monedas. De las cinco clases que hay, la moneda Oro no precisa de animación pues es estática. Las cuatro monedas restantes dispondrán de 2 *sprites* propios para su animación: orientación a izquierda y orientación a derecha, además de un tercer *sprite* compartido por todas ellas que representará el canto de la moneda.

Con todo esto, se tienen cubierto los 64 *sprites hardware* disponibles.

GRÁFICOS

De la tabla de *sprites* de la versión original (véase la Fig. 6.7) se ha seleccionado un subconjunto que cubra las necesidades descritas previamente. El proceso seguido para la realización los *sprites* ha sido el siguiente:

- Manipulación de la imagen con la aplicación GIMP: sobre la imagen con la tabla de *sprites* de la versión original se ha aplicado una rejilla de tamaño 1 x 1, enmarcando cada *sprite* a reproducir en un marco de 16 x 16 *píxeles* de tamaño para que sirva de modelo a la hora de definir los *sprites*.
- Con la aplicación *spriteSX devtool* se ha creado un proyecto nuevo para *sprites* de tamaño 16 x 16 monocromo y se ha creado un banco con los 64 *sprites* donde se ha definido cada uno de ellos en base al modelo original y la especificación sobre los *sprites* dada anteriormente (véanse las Figuras 6.17 y 6.18).
- Con la misma aplicación, a través de la opción de menú *Get Data* y activando sólo la casilla de *Sprite Data*, se han obtenido los valores en hexadecimal de los *sprites* para el lenguaje de programación 'C' que se han almacenado en un fichero (véase la Figura 6.17).
- Con el programa creado *ex profeso* en C (saveSPR.c) se guarda en un fichero los valores en bruto obtenidos (sprites.dat).

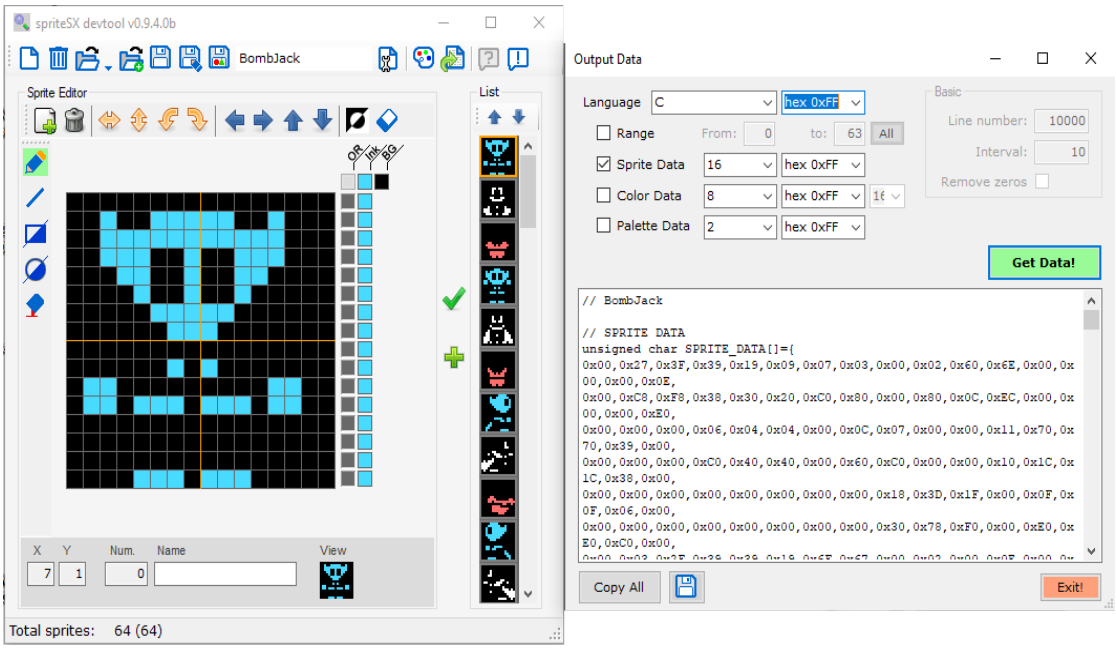

Figura 6.17: Definiendo los sprites

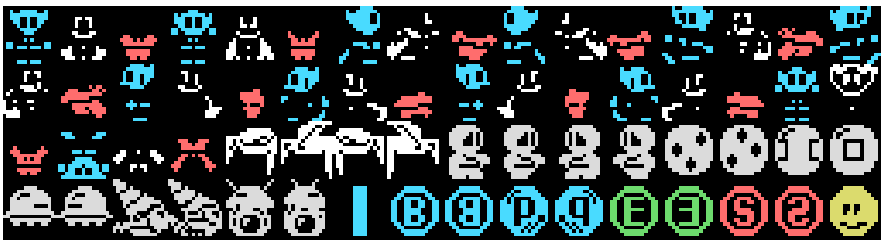

Figura 6.18: Banco de 64 sprites definidos

LÓGICA

Tanto el avatar del jugador (Jack) como los enemigos, las monedas y las bombas se han modelado como estructuras de registros que almacenan información referente a ellos. Para cada caso ha sido:

- **Bomba**: la posición origen de la esquina superior izquierda (coordenadas x, y) donde empieza a dibujarse la bomba y los estados de encendida y recogida.
- **Avatar (player)**: los *sprites* de inicio y actual, el plano del VDP en el que se ubica, las coordenadas (x,y) de la pantalla que ocupa, la fila y la columna que ocupa en la matriz Mundo, las coordenadas máximas y mínimas de su caja de inclusión AABB (x1, x2, y1, y2) para la matriz Mundo, la velocidad a la que se desplaza por los ejes (dx, dy), si limita con alguna plataforma en alguno de los puntos cardinales (*grounded*, *ceiling*, *leftBounded* y *rightBounded*), un temporizador para la acción de salto, el número de bombas que lleva recolectadas, la puntuación que lleva conseguida, el estado actual en el que se encuentra de su máquina de estados y si dispone de inmunidad. Esta última para la fase de pruebas.
- **Enemigos**: el identificador del tipo de enemigo, los *sprites* de inicio y actual, el plano del VDP en el que se ubica, las coordenadas (x,y) de la pantalla que ocupa, la fila y la columna que ocupa en la matriz Mundo, las coordenadas máximas y mínimas de su caja de inclusión AABB (x1, x2, y1, y2) para la matriz Mundo, la velocidad a la que se desplaza por los ejes (dx, dy), si limita con alguna plataforma en alguno de los puntos cardinales (*grounded, ceiling, leftBounded* y *rightBounded*), la orientación (si mira a derechas o a izquierdas) y el estado actual en el que se encuentra de su máquina de estados. Una variable (eje) que sirve para implementar la inteligencia artificial de algunos enemigos y un temporizador para medir el tiempo que permanece en estado de Moneda (también se utiliza para complementar la inteligencia artificial de algunos enemigos).
- **Moneda**: el identificador de moneda, el subtipo de moneda, los *sprites* de inicio y actual, la fila y la columna que ocupa en la matriz Mundo, el color de la moneda, el plano del VDP en el que se ubica, las coordenadas (x, y) de la pantalla que ocupa, la velocidad a la que se desplaza por los ejes (dx, dy), el tiempo de vida que lleva la moneda en la pantalla, si la moneda está activa (y

por tanto, aparece en la pantalla) y si limita con alguna plataforma en alguno de los puntos cardinales (*grounded, ceiling, leftBounded* y *rightBounded*).

Las funciones implementadas para la carga y visualización de los *sprites* son:

- void loadSprites (char *filename).
- void enciendeBomba (t_bomba *bomba).
- void repintaFondo (byte fila, byte columna).
- void dibujaEnemigo (byte enemigo, byte nEnemigos).
- void dibujaAvatar(byte plano, byte patron, byte x, byte y).
- void dibujaMoneda (byte moneda, byte nMonedas).

6.3.7. Bucle principal

Se ha implementado un contador de iteraciones del bucle principal (*frame*) con el proposito de repartir alguna carga de cómputo dependiendo de la iteración en curso. Por ejemplo, la actualización de los enemigos se realiza en iteraciones pares y la actualización de las monedas se realiza en las iteraciones impares.

6.3.7.1. Obtención de la entrada de usuario

La entrada del usuario se ha modelado con una estructura de registro que almacena la información referente a la dirección del movimiento que realiza el avatar y la acción de salto.

Por defecto, se considera como dispositivo de entrada principal el teclado (cursores y la barra espaciadora) pero se comprueba si existe alguna palanca de juego conectada y de ser el caso, esta se asigna como dispositivo de entrada. En el caso de haber dos palancas de juego conectadas al ordenador, tiene preferencia la que está conectada en el primer puerto de juego del ordenador.

También se analiza el mapa de teclado para comprobar si ciertas teclas son pulsadas, concretamente la tecla de escape ‘ESC’ que aborta la partida en curso volviendo a la pantalla de menú o regresa al MSX-DOS si ya se está en el menú.

La función implementada para la obtención de la entrada de usuario es:

- char procesarEntrada (t_Entrada *entrada).

6.3.7.2. Actualización del avatar

La actualización del avatar se modela a través de una máquina de estados finitos. La definición de estados y entradas, así como su representación en diagramas de estado figura en el *anexo IV Máquina de estados del avatar*.

La primera comprobación que se realiza para el avatar es si existe colisión entre este y las plataformas o los límites de la pantalla. Esta comprobación genera el valor de algunas entradas que disparan la transición entre estados.

La animación gráfica y la simulación física del avatar forman parte de las salidas de los estados.

La actualización del avatar para el caso de pérdida de una vida se modela fuera de esta máquina de estados. Se produce durante la detección de colisiones con los enemigos. Esto es así ya que la detección de colisiones se produce con posterioridad a la actualización del avatar según la entrada del jugador.

Las funciones implementadas para la actualización e inicialización del avatar son:

- void actualizaAvatar (t_Entrada *entrada).
- void inicializaAvatar (void).

6.3.7.3. Inteligencia artificial

Bajo este concepto se desarrolla la curva de dificultad que ha de experimentar el jugador a lo largo de la partida así como el comportamiento de las entidades *Moneda* y *Enemigos*.

La curva de dificultad va asociada al nivel, es decir, a medida que se progresa y se aumenta el número de niveles superados, se incrementa el grado de dificultad. Esta se ha implementado a través de tres variables que controlan el número máximo de enemigos a mostrar en el nivel (*maxEnemigos*), cada cuanto tiempo ha de aparecer un nuevo enemigo (*timerEnemigos*) y la velocidad a la que se mueven (*velocidad*). Estas tres variables se actualizan en el momento en el que se inicializa cada nivel.

Respecto al comportamiento de las monedas, este se ha implementado siguiendo la especificación del diseño y a la vez que se actualiza el estado de las monedas. El comportamiento de los enemigos, al igual que con las monedas, se ha hecho junto con la actualización del estado de los enemigos. Los enemigos principales de tipo *momia*, al alcanzar la parte inferior de la pantalla, se convierten al azar en algún tipo de enemigo secundario. La probabilidad que tienen los enemigos tipo *momia* de caer de las plataformas para convertirse en enemigos secundarios se ha regulado midiendo el tiempo que han de permanecer sobre la plataforma. A medida que se superan niveles, disminuye el tiempo de permanencia de estos enemigos sobre las plataformas.

Las inteligencia artificial se ha implementado a través de las funciones :

- void main (void).

- void actualizaMoneda (t_Moneda *moneda, unsigned int frame).
- void actualizaEnemigos (t_Enemigo *enemigo, unsigned int frame, byte i, boolean *enemigo_monedaOro).

6.3.7.4. Simulación física

Los elementos que requieren de simulación física son el avatar, los enemigos y las monedas pues son los elementos que se mueven por la pantalla. Este movimiento se basa en incrementar o decrementar su posición una cantidad concreta de *píxeles* a lo largo de los ejes de coordenadas: abscisas y ordenadas.

La cantidad de *píxeles* que ha de desplazarse viene marcada por la variable *velocidad*. Esta variable se divide en dos partes: el *nibble* alto contiene la información de la velocidad a la que se desplaza el avatar y el *nibble* bajo contiene la información relativa a la velocidad a la que se mueven los enemigos y las monedas. Esto permite incrementar sólo la velocidad de los enemigos, incrementando la dificultad del juego, sin variar la velocidad del avatar.

La actualización de las nuevas posiciones del avatar, de los enemigos y de las monedas se realiza a la vez que se actualiza la inteligencia artificial de los mismos, en el caso del avatar depende de la entrada que proporcione el jugador.

El salto normal del avatar se ha implementado a través de un contador (*cuentaSalto*). Este contador decrementa la posición del avatar en el eje de ordenadas tantos *píxeles* como indique la variable *velocidad* hasta alcanzar su máximo prefijado por la constante *Timer_Salto_MAX,* que corresponde con la mitad de la altura del eje de ordenadas (SCREEN_HEIGHT). En el caso del supersalto del avatar, este finaliza al alcanzarse el punto origen del eje de ordenadas.

La simulación física se ha implementado a través de las funciones :

- void actualizaAvatar (t_Entrada *entrada).
- void actualizaMoneda (t_Moneda *moneda, unsigned int frame).
- void actualizaEnemigos (t_Enemigo *enemigo, unsigned int frame, byte i, boolean *enemigo_monedaOro).

6.3.7.5. Detección de colisiones

La detección de colisión entre el avatar y las plataformas o entre el avatar y los límites de la pantalla se realiza a la hora de actualizar el avatar. Para ello se hace uso de las cuatro variables (*grounded*, *ceiling*, *leftBounded* y *rightBounded*) del avatar y de la matriz Mundo, que guarda la información de la ubicación de las plataformas. La detección de colisión con las plataformas consiste en comprobar si la celda de la matriz Mundo a ubicar al avatar tras su actualización ya está ocupada por una plataforma.

La detección de colisión entre el resto de entidades del juego y las plataformas o los límites de la pantalla se realiza de forma análoga al avatar y durante el proceso de actualización de dicha entidad.

La detección de colisión entre el avatar y el resto de entidades del juego (bombas, monedas y enemigos) se realiza recorriendo las listas de cada entidad comprobando si ocupan las mismas coordenadas (fila y columna) que el avatar en la modelización del nivel.

Durante este proceso, además, se actualiza el estado de pérdida de vida del avatar en caso de colisión con un enemigo.

La función implementada para la detección de colisiones con el resto de entidades del juego es:

- bool actualizaColisiones (byte nEnemigos, byte nMonedas).

6.3.7.6. Actualización de las entidades del juego

ENEMIGOS

Al iniciar el nivel sólo existe un enemigo de tipo principal. La cantidad de enemigos se incrementa cada lapso de tiempo definido en la variable *timerEnemigos* hasta alcanzar el número máximo de enemigos fijado para dicho nivel de juego. Dependiendo del nivel, existe un máximo de enemigos principales de tipo *pájaro* que se rige por la fórmula descrita en el apartado de diseño. El resto de enemigos principales son de tipo *momia*.

Los enemigos principales tipo *momia* se convierten en enemigos secundarios al alcanzar, su posición, el límite inferior de la pantalla. El tipo de enemigo secundario al que se convierte es elegido al azar.

Se han implementado dos máquinas de estados que rigen dos aspectos de los enemigos. La primera máquina de estados controla el tiempo de vida de los enemigos y la segunda la orientación del *sprite* del enemigo, en caso de tener más de una orientación.

En la primera máquina de estados, los enemigos transitan entre tres estados: ALIVE (vivo), MONEDA (convertido en *moneda oro*) y DEAD (muerto). Parten, inicialmente, del estado ALIVE, si el avatar coge una moneda tipo *Power* (P), estos se convierten en *monedas oro* (estado MONEDA) quedándose inmóviles durante un periodo de tiempo delimitado por la constante *Timer_monedaActiva*. Se ha implementado otra variable (*nMonedasORO*) a modo de bandera-contador que indica si los enemigos (y cuántos de ellos, están en estado MONEDA restringiendo, así, la creación de nuevos enemigos principales mientras esté la bandera activa (*nMonedasORO* > 0). Si durante este periodo de tiempo, el avatar coge alguna de las *monedas oro,* el enemigo muere y desaparece (estado DEAD). Al vencer el periodo de tiempo, los enemigos en estado MONEDA recuperan su estado inicial ALIVE.

La creación de un nuevo enemigo principal implica la inserción de este en la lista de enemigos. La inserción se realiza al final de la lista (última posición del vector). Al recoger el avatar una moneda oro (acción de matar a un enemigo), la lista de enemigos se reordena para dejar la lista sin huecos intermedios.

La segunda máquina sólo tiene dos estados: LOOK_RIGHT, que indica si el enemigo mira a la derecha y LOOK_LEFT, que indica si mira a la izquierda. Estos estados marcan la orientación del *sprite* actual a dibujar para dicho enemigo. Las transiciones entre estos dos estados dependen de si colisionan con los límites laterales de la pantalla o de la ubicación del avatar.

MONEDAS

Se ha implementado un máximo de dos monedas simultáneas desplazándose por la pantalla a través de la constante MAX_MONEDAS. La creación de una moneda de tipo *Bonus* (B) se realiza mediante el vencimiento de un temporizador (*Timer_monedaNueva*) de valor constante, si se da el prerrequisito descrito en el apartado de diseño. La creación de una moneda de tipo *Special* (S) tiene añadido un factor multiplicador de carácter aleatorio dentro de un intervalo dado [5, 9]. La creación del resto de tipos de monedas sigue lo fijado en el diseño.

Las monedas tienen dos estados, el de activa (verdadero) y el de desactivada (falso). Sólo las monedas en estado activo son las que se actualizan y se dibujan en pantalla. La moneda, al ser creada, adquieren el estado de activa. Transita al estado de desactivada en el caso de ser recogida por el avatar o que venza su tiempo de vida.

La moneda creada se inserta en la primera posición libre —porque no exista moneda alguna en dicha posición o porque esté desactivada— de lista de monedas (vector).

Todas las monedas cuentan con tres *sprites* para su animación, siendo el *sprite* de canto de la moneda compartido por todas ellas.

El tiempo de vida una moneda viene marcado por el mismo temporizador que controla el estado de MONEDA de los enemigos (*Timer_monedaActiva*).

Las funciones implementadas para la creación y actualización del resto de entidades del juego son:

- void creaEnemigoPrincipal (void).
- void creaEnemigoSecundario (byte enemigo).
- boolean ordenaListaEnemigos (void).
- void actualizaEnemigos (t_Enemigo *enemigo, unsigned int frame, byte i, boolean *enemigo_monedaOro).
- void crearMoneda (byte type).
- void actualizaMoneda (t_Moneda *moneda, unsigned int frame).

6.3.7.7. Realimentación sensorial al jugador

EFECTOS SONOROS

Con la herramienta AYF Editor se ha creado un banco de efectos sonoros (FX.afb) con diez elementos cogidos de entre la colección que la acompaña. La tabla 6.2 muestra los efectos escogidos para cada evento y que son reproducidos al dispararse dicho evento.

La actualización del efecto sonoro se produce al final de la iteración del bucle principal.

Pos.	Evento	Fichero origen	Librería (colección)
1	Disparo de inicio de la partida en el menú	*nomame017.afx*	*Streets of rage 2*
2	Sirena cuando aparece la moneda 'P'	*megatetris2k_4.afx*	*ZX games*
3	Presionar el botón de salto	*fireandice_5.afx*	*ZX games*
4	Tintineo al aparecer la moneda 'E' (vida extra)	*nemesis_15.afx*	*ZX MSX demo*
5	Pérdida de una vida	*girlyblock_15.afx*	*ZX MSX demo*
6	Tintineo al aparecer las monedas "B", "E" y "S"	*vampirekiller_1.afx*	*ZX MSX demo*
7	Fin de la partida	*nomame012.afx*	*Streets of rage 2*
8	Bomba encendida	*nomame021.afx*	*Streets of rage 2*
9	Buscar letra en el Salón de la Fama	*cybernoid_8.afx*	*ZX games*
10	Seleccionar letra en el Salón de la Fama	*nomame009.afx*	*Streets of rage 2*

Tabla 6.2: Banco de efectos sonoros

MÚSICA (BANDA SONORA)

Pese a que el juego original dispone de música que suena a lo largo del nivel y en otros eventos, no se ha implementado música para esta versión por no disponer de memoria suficiente para incorporarla y reproducirla.

6.3.8. Finalización

Dos variables, *gameOver* y *endGame*, controlan el modo de finalización de la partida. El valor verdadero en la variable *gameOver* indica que la partida ha finalizado sin éxito y aparece sobreimpresa en la pantalla el texto de fin de juego "GAME OVER" junto con el efecto de sonido de fin de la partida. El valor verdadero en la variable *endGame* indica, por contra, que la partida ha finalizado con la consecución exitosa de todos los niveles de juego, dando paso a la visualización en la pantalla de la imagen final con el texto de felicitación por dicho hito. Ambas variables no pueden tomar el mismo valor a la vez.

Posteriormente, se comprueba si el jugador ha obtenido la puntuación mínima para grabar sus iniciales en el Salón de la Fama.

Las funciones implementadas para esta fase son:

- void mostrarLetreroGameOver (void).

- char entrarEnHOF (t_HallOfFame * HallOfFame).

6.4. Pruebas

En este apartado se describen las pruebas de *software* diseñadas para verificar y validar que se ha desarrollado con corrección el videojuego, es decir, hace lo que debería hacer de acuerdo con sus especificaciones de diseño y detectar algún defecto o fallo que haya podido producirse durante la fase de implementación.

Para ello, del proceso global de pruebas, se han diseñado los siguientes tipos de pruebas:

- **Integración o funcional**: con el juego implementado se ha generado una imagen en disco del videojuego con el fichero compilado y los ficheros de datos asociados al mismo así como un fichero imagen en formato ROM y se han probado sobre diversas máquinas MSX de primera generación emuladas a través del emulador OpenMSX.
- **Del sistema**: se ha probado el videojuego en ordenadores MSX-1 reales haciendo uso de las imágenes de disco y ROM a través de dos cartuchos, un emulador de unidad de disco virtual y otro que permite el almacenamiento de imágenes ROM y de disco, y su ejecución como si se tratase de un cartucho ROM o disquete original.

De los restantes tipos de pruebas que conforman el proceso de prueba, el de pruebas **unitarias**, que validan el código implementado, se ha omitido para ajustarse a los tiempos marcados en la planificación inicial. Respecto a las pruebas de **aceptación**, que permite la prueba del videojuego por el usuario final, se desarrollará *a posteriori* a este trabajo, compartiendo la imagen de disco y la imagen ROM del videojuego entre la comunidad de usuarios de MSX para que sea probada y ver si tiene aceptación.

6.4.1. Integración

Para ensamblar todo el videojuego, se ha hecho uso de la herramienta *MSX Disk Image*. Con ella se ha creado una imagen de disco (*BombJack.dsk*) de baja densidad (720 KB DD) que contiene los 45 ficheros que componen el videojuego y son:

- 1 fichero con el videojuego compilado y ejecutable (extensión .COM).
- 9 ficheros de imágenes que corresponden con la pantalla de carga, la pantalla del menú inicial, los 5 escenarios, la pantalla de bonificación de puntos y la pantalla final tras la consecución de todos los niveles (extensión .SC2).
- 1 fichero de imagen con la distribución de la Tabla de nombres del *Salón de la Fama* (extensión .SCR).

- 1 fichero con la definición de los *sprites* (extensión .DAT).
- 1 fichero con el banco de efectos de sonido (extensión .AFB).
- 16 ficheros con la información de los arreglos de nivel —distribución de las bombas y plataformas— (extensión .MAP).
- 16 ficheros con la información relativa al orden de encendido de las bombas según el arreglo de nivel (extensión .BMB).

Así mismo, se han incluido en la imagen de disco los ficheros relativos al MSX-DOS1 versión 1.03 (*command.com* y *msxdos.sys*) que vienen con la librería FUSION-C y cuya ruta se encuentra en el directorio de trabajo, subdirectorio "Working Folder\Tools\MSX-DOS\MSXDOS-103".

También se ha incluido un fichero de autoarranque (*autoexec.bat*) con el único contenido del nombre, sin su extensión, del fichero ejecutable: *bombjack* para que inicie la carga del videojuego al arrancar el ordenador con el disco insertado en la unidad de disco.

Haciendo uso de la herramienta *Dsk2rom* se ha convertido la imagen de disco en una imagen de memoria ROM. Se han habilitado las siguientes opciones:

- (f): rellenar el fichero hasta completar el tamaño estándar de un MegaRom.
- (a): uso de un gestor de memoria (*mapper*) de tipo *ascci8*.
- (c 2) para comprimir al máximo el fichero resultante y hacer que la imagen ROM sea del menor tamaño estándar posible.

Por tanto, el comando ejecutado con la configuración final de la herramienta es:

```
dsk2rom -fac 2 bombjack.dsk bombjack.rom
```

Para comprobar el correcto funcionamiento de todo el videojuego a lo largo de una partida, se ha implementado la posibilidad de que el avatar sea inmune frente a las colisiones con los distintos enemigos. Para habilitarla, basta con teclear las letras que componen la palabra 'fede' mientas se muestra el *Salón de la Fama* en el menú de inicio. Si esta palabra es escrita, la pantalla lo corroborará parpadeando en blanco por un instante.

Finalmente, se ha probado el videojuego en su formato de imagen de disco y formato ROM en una batería de ordenadores MSX-1 de distintos fabricantes a través del emulador OpenMSX (todos ellos con 64 KB de memoria RAM) (véanse las figuras 6.19 y 6.20). Para la verificación de la versión de disco se ha configurado el emulador con una unidad de disco —se han probado dos modelos de unidades de disco distintas—. La Tabla 6.3 muestra el resultado obtenido en la relación de modelos de ordenadores emulados en los que ha sido probado el videojuego e indica, también, el modelo de unidad de disco emulada y el formato (DSK, ROM) probado.

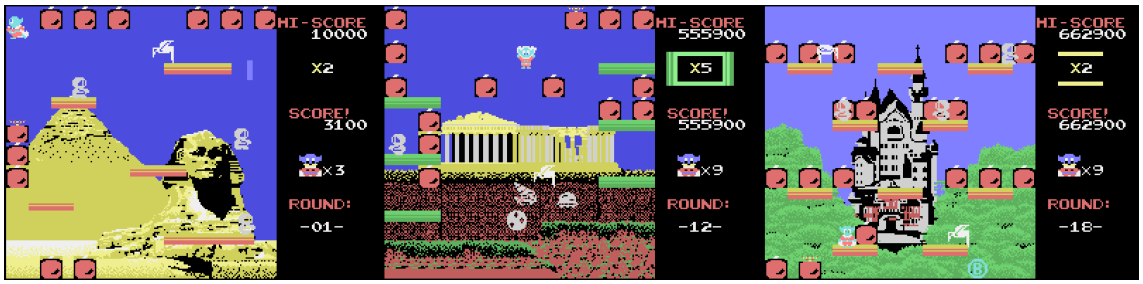

Figura 6.19: Pruebas de integración #1

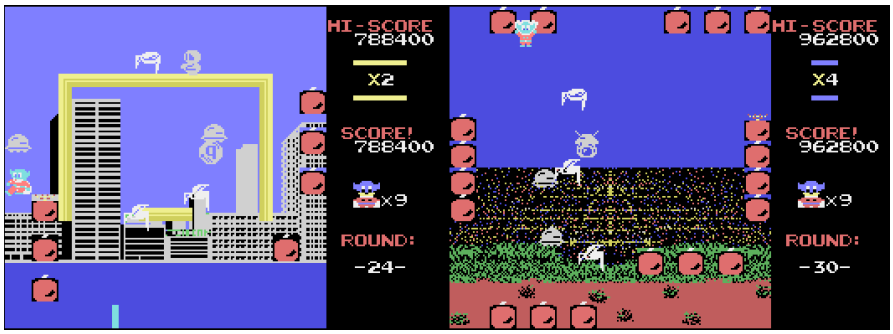

Figura 6.20: Pruebas de integración #2

			RESULTADO	
N.º prueba	Modelo ordenador MSX-1	Unidad de disco	Versión DSK	Versión ROM
1	Canon V-20	Sony HBK-30	OK	OK
2	JVC HC-7	Panasonic FS-FD1A	OK	OK
3	Mitsubishi ML-FX1	Sony HBK-30	OK	OK
4	Panasonic CF-2700	Panasonic FS-FD1A	OK	OK
5	Philips VG-8020/20	Panasonic FS-FD1A	OK	OK
6	Sanyo MPC-100	Sony HBK-30	OK	OK
7	Sony HB-10P	Panasonic FS-FD1A	OK	OK
8	Sony HB-75P	Sony HBK-30	OK	OK
9	Spectravideo SVI-728	Sony HBK-30	OK	OK
10	Talent DPC-200	Panasonic FS-FD1A	OK	OK
11	Toshiba HX-10D	Panasonic FS-FD1A	OK	OK
12	Yamaha CX5MII	Sony HBK-30	OK	OK

Tabla 6.3: Pruebas de integración sobre distintos ordenadores MSX-1 (emulación)

6.4.2. De sistema

En este punto del proceso de pruebas, se ha probado el videojuego en dos ordenadores MSX-1 reales (no emulados) propiedad de quien suscribe este trabajo: Sony HB-10P (véase la Figura 6.21) y Philips VG-8020 (véase la Figura 6.22). Ambos ordenadores no disponen de unidad de disco, por lo que se ha hecho uso de *hardware* adicional: el cartucho Rookie Drive que implementa una unidad de disco virtual sobre

USB para MSX y el cartucho MegaFlashROM SCC+ SD[60] que incorpora una unidad de almacenamiento masivo microSD y una memoria FlashROM que permite cargar en ella, desde la microSD, imágenes ROM o DSK de videojuegos sobre cualquier ordenador MSX sin unidad de disco.

Adicionalmente, en las pruebas realizadas sobre estos dos ordenadores, se ha contado con una palanca de juegos.

La Tabla 6.4 muestra el resultado obtenido en las pruebas y las distintas configuraciones realizadas.

			RESULTADO	
N.º prueba	Modelo ordenador MSX-1	Unidad de disco	Versión DSK	Versión ROM
1	Philips VG-8020/20	Rookie Drive	OK	-
2	Philips VG-8020/20	MegaFlashROM SCC+ SD	OK	OK
3	Sony HB-10P	Rookie Drive	OK	-
4	Sony HB-10P	MegaFlashROM SCC+ SD	OK	OK

Tabla 6.4: Pruebas de sistema sobre ordenadores MSX-1 (reales)

Figure 6.21: Prueba de sistema (SONY HB-10P)

Figure 6.22: Prueba de sistema (Philips VG-8020/20)

60 En el proceso de grabado de la imágen ROM en la memoria FlashROM es necesario desactivar los subslots y dejar sólo el MegaFlashROM (opción /U) para que el videojuego funcione.

7 . Conclusiones

En este último capítulo, como informe de terminación y cierre del trabajo, se analiza el cumplimiento de los objetivos marcados al inicio del mismo, se describen los problemas hallados así como la solución propuesta y los riesgos asumidos asociados. Se evalúa, también, la planificación planteada y el control de los tiempos fijados así como las lecciones aprendidas a lo largo de todo el desarrollo y, para finalizar, se realiza una propuesta de posibles trabajos futuros y ampliaciones.

7.1. Revisión de los objetivos

En este trabajo se pretendía realizar una guía o manual básico que permita abordar el desarrollo *amateur* de videojuegos para ordenadores MSX de primera generación, haciendo uso de herramientas modernas en ordenadores actuales mediante la compilación cruzada de proyectos. En este punto, se considera que se ha alcanzado dicho objetivo general y prueba de ello es la realización de los entregables propuestos.

Los capítulos comprendidos entre el tercero y el quinto, ambos inclusive, representan una guía completa, sin llegar a ser excesivamente exhaustiva que desmotive la lectura de la misma, de cómo iniciarse el desarrollo de videojuegos para ordenadores MSX-1. El capítulo tercero estudia la arquitectura del sistema MSX de primera generación y describe con detalle los componentes *hardware* que lo componen pues es importante conocer la organización de la máquina para la que se pretende desarrollar. El capítulo cuarto propone un amplio conjunto de herramientas *software* gratuitas o libres con las que construir un entorno de desarrollo completo sobre ordenadores actuales y abarcando los tres perfiles básicos de desarrolladores, permitiendo el paradigma de desarrollador en solitario o de grupos pequeños tal y como se hacía en la “Edad de oro del software español”. El capítulo quinto enumera y describe los distintos módulos y funciones que implementa la librería FUSION-C relacionándolos con los distintos componentes *hardware* del MSX-1 descritos en el capítulo tercero.

El capítulo sexto se centra en el desarrollo de un proyecto de videojuego concreto a través de compilación cruzada. En él se abordan todas las etapas del modelo clásico del ciclo de vida del *software* para asegurar que el *software* producido es de calidad. Así mismo, se tratan los aspectos característicos al diseño y desarrollo de un videojuego. Como consecuencia de este capítulo se han producido varios entregables que se anexan: el código fuente del videojuego y programas generados para integración del arte, y la versión del videojuego *Bomb Jack* para ordenadores MSX-1 en dos formatos distintos, una imagen de disco y una imagen en memoria ROM.

7.2. Problemas y soluciones

El primer problema a afrontar fue la elección de aplicaciones y herramientas *software* que complementaran a las que vienen junto con la librería FUSION-C para conformar un entorno de desarrollo lo más completo posible y que cubrieran los tres perfiles de desarrollador. Afortunadamente, existe una amplia comunidad de usuarios de MSX que permanece activa y que ha desarrollado aplicaciones gratuitas para cubrir todos los aspectos. Comunidad que también ha creado y mantiene diversas páginas web y foros específicos que orientan a como iniciar el proceso de desarrollo. Tras analizar las múltiples herramientas disponibles, se valoró incluir aquellas que tuvieran una curva de aprendizaje menor, ya fuera por su simplicidad o por venir acompañadas de manual, tutoriales o ejemplos.

El resto de problemas han surgido durante la fase de implementación del proyecto-aprendizaje. El primer problema en esta fase se debió a un fallo que viene de fábrica con la librería FUSION-C y es que no dibuja correctamente los *sprites* sobre el VDP del MSX-1. Tras consultar los foros específicos de la librería en la web *MSX Resource Center* [23], se observó que este fallo ya estaba documentado y que el autor de la librería había facilitado un parche para solventarlo. Se instaló el parche en el entorno de desarrollo y se documentó en la memoria (véase ANEXO I - Instalación del SDK).

El segundo problema surgido en esta fase estaba relacionado con la *regla del 5º sprite*. El *sprite* del avatar se compone de 3 *sprites* y teniendo en cuenta la cantidad de enemigos y monedas que pueden haber simultáneamente en pantalla, la probabilidad de que esta regla se incumpla es alta. La solución adoptada ha sido doble, primero priorizar el dibujado de *sprites*, de forma que se han asignado planos más prioritarios a los *sprites* del avatar –por lo que siempre es visualizado–, después a los *sprites* de los enemigos y, por último, a los *sprites* de las monedas. Como segunda solución, en caso de que algún *sprite* enemigo o moneda incumpla la regla del 5º *sprite*, se rota el plano de prioridad entre las mismas entidades. Es decir, si un *sprite* enemigo incumple la regla, todos los *sprites* enemigos decrementan la prioridad de su plano –siempre dentro de los planos asignados para ellos– haciendo que el *sprite* que incumplía la regla ocupe el plano de mayor prioridad. Ídem para las monedas. Esta solución hace que en algún momento no se dibuje en pantalla alguna parte de un *sprite* pero se considera un mal menor que no resta demasiada jugabilidad.

El tercer problema surgido estaba relacionado con el recurso de memoria principal y la gestión dinámica de ella. En un principio, las listas de enemigos y monedas junto con los búferes temporales para la carga de datos se realizaban a través de memoria dinámica, creando y liberando memoria RAM según la necesidad. A medida que el programa compilado crecía en tamaño, dejando menor espacio de memoria para la pila de ejecución y el montículo de memoria dinámica, se provocaba la congelación y bloqueo del ordenador en tiempo de ejecución. Tras un primer análisis aproximativo del anidamiento de las funciones y el número de parámetros en ellas, el consumo aproximado de memoria de la pila que realizan y las llamadas de creación de memoria dinámica en alguna de estas funciones, y observar que podría producirse desbordamiento de pila, se optó por hacer sólo uso de memoria estática. Esto implicó reescribir parte del código y añadir nuevas funciones pero resolvió el problema.

El cuarto problema surgido también estaba relacionado con la memoria principal y el sistema operativo de disco. De los 64 KB de memoria RAM iniciales, el MSX-DOS se guarda, aproximadamente, 10 KB de memoria para el sistema y la vuelta al DOS tras la finalización del programa en ejecución. Esto deja, de inicio, disponible para el programa 54 KB de memoria libre. Al hacer uso exclusivo de memoria estática, el programa compilado, tras añadir los efectos de sonido, ocupa un tamaño de 49 KB. Esto dejaba 5 KB libres de memoria para la pila de llamadas y los módulos de música a añadir. Se trató de seleccionar varios módulos de música de entre los que vienen como ejemplo con la aplicación Vortex Tracker II –aquellos que ocuparan el menor tamaño (1 ó 2 KB)– para realizar pruebas antes de componer las melodías originales del videojuego *Bomb Jack* pero incluso haciendo uso de un único módulo de música con el mínimo tamaño, el videojuego se congelaba en tiempo de ejecución. Esto puede deberse a que la reproducción de música necesite más memoria libre disponible. Esto motiva que la versión realizada del *Bomb Jack* no incluya melodías de música.

Existe un problema añadido debido al requisito de tamaño de 64 KB memoria RAM para el funcionamiento del sistema operativo MSX-DOS1. Y es que, a pesar que se realice el videojuego en formato ROM —posibilitando que pueda ser ejecutado en ordenadores sin unidad de disco—, el ordenador debe seguir teniendo un tamaño de 64 KB de memoria RAM ya que la conversión en este formato sólo permite la carga de sectores de disco desde memoria ROM en lugar de desde un disquete. Es decir, sigue ejecutándose el MSX-DOS1. Esto impide que el videojuego pueda ser ejecutado en ordenadores MSX-1 con tamaños de memoria menor a 64 KB aun en su formato ROM.

Puede existir un problema potencial debido a un riesgo asumido de inicio y que está relacionado con la consideración de que este trabajo esté, en parte, destinado a alumnos de, al menos, segundo curso de grado. Esta consideración implica que el lector pueda no tener conocimientos sobre la tipología de pruebas definida en la Ingeniería de Software, especialmente de las pruebas unitarias que validan el código de las funciones. Si bien se barajó, durante la fase de pruebas, la realización de la prueba unitaria para la función principal, el alto coste temporal de la realización de la prueba y el no demorar en exceso los plazos de entrega hicieron declinar negativamente esta opción. Durante las pruebas de integración se ha probado exhaustivamente el videojuego, finalizando con éxito todos los niveles de juego sin que se produzca ninguna anomalía en su ejecución en distintos modelos de MSX-1. Esto hace pensar que la probabilidad de que alguna anomalía se produzca es mínima.

7.3. Evaluación de la planificación

La planificación inicial fijaba un total de 360 h de esfuerzo destinadas a la elaboración de este trabajo. Este total de horas se balanceaba entorno al 50 % para cada una de las dos propuestas planificadas. Respecto a este punto, comentar que se ha respetado la distribución de horas en un 90%. Dentro de las tareas de implementación, ha habido algunas de ellas que ha requerido menos horas de las previstas y otras que han requerido de un número mayor pero unas han compensado las otras (p.e. la implementación de los *sprites* o de la simulación física).

Respecto a los plazos de tiempo planificados, se preveía finalizar el trabajo el día 4 de marzo de 2020. Los problemas surgidos durante la fase de implementación han hecho que la fecha de finalización se retrasara en 15 días. Teniendo en cuenta el coste económico del proyecto, que es nulo, y que la fecha final de entrega no se ha desviado en exceso de la fecha inicial, se considera que se ha cumplido de forma aceptable con los plazos de entrega.

7.4. Lecciones aprendidas

Con la realización de este trabajo ha podido ponerse en práctica conceptos aprendidos a lo largo de la carrera. Los primeros conceptos usados han sido los vistos en la asignatura Gestión de proyectos (tercer curso) para abordar la elaboración de este TFG. Para la realización de la primera propuesta han sido necesarios conceptos vistos en las asignaturas de Fundamentos de computadores (primer curso) y Estructura de computadores (segundo curso). Para la realización de la segunda propuesta lo han sido de las asignaturas Programación (primer curso), Teoría de autómatas y lenguajes formales, Estructura de datos y algoritmos (segundo curso), Ingeniería del software (tercer curso), Introducción a la programación de videojuegos y Arquitectura y entornos de desarrollo para videoconsolas (optativas).

Así mismo, la realización de este trabajo ha permitido el ampliar conceptos sobre la organización de computadores a través del estudio de un caso concreto como ha sido el sistema MSX. También ha permitido el profundizar los conceptos relativos al lenguaje de programación C, principalmente al uso de la pila de llamadas para el pase de parámetros. Y, finalmente, ha permitido el conocer las herramientas disponibles para la creación del arte en un videojuego.

7.5. Trabajos futuros y ampliaciones

Como trabajo futuro que amplíe este trabajo puede plantearse la resolución de los flecos pendientes que deja por falta de tiempo al querer ceñirse a los plazos de entrega. Puede estudiarse una solución más eficiente en la gestión de memoria RAM que permita la inclusión de melodías de música en el videojuego y, además, plantee una mejor resolución al incumplimiento de la *regla del 5º sprite*.

También puede extenderse haciendo un estudio sobre la arquitectura del microprocesador Z80 y su lenguaje de ensamblador, así como complementarlo desarrollando proyectos que estudien el desarrollo de videojuegos para la segunda o cuarta generación del sistema MSX.

Otro trabajo futuro sería la realización de una guía de desarrollo de videojuegos haciendo uso del entorno de trabajo CPCtelera[61] en lugar de la librería Fusion-C, dado que CPCtelera también supone una API de desarrollo para microprocesadores Z80 en lenguaje C. Si bien, el entorno de trabajo de CPCtelera se desarrolló para ordenadores

61 CPCtelera: https://github.com/lronaldo/cpctelera

Amstrad CPC de 8-bits por Francisco Gallego, el código binario que genera es ejecutable por microprocesadores Z80 –compartido tanto por los ordenadores MSX como por los ordenadores Amstrad CPC–. Esto hace que, en teoría, las modificaciones a realizar sobre el entorno de CPCtelera para adaptar el juego final a ordenadores MSX no sean complicadas en exceso. Ya existe algún apunte[62] realizado a este respecto por Retrobytes Productions[63] pero, como se comenta, no hay documentación acerca de ello. Así mismo, podría incluirse un estudio comparativo sobre el rendimiento del código generado por ambos.

62 https://twitter.com/amstradgamer/status/991239885716586496

63 Retrobytes Productions: https://retrobytesproductions.blogspot.com/

Bibliografía

[1]: Avalon Software (1985). *The MSX Red Book*. : Kuma Computers

[2]: Sony Corporation (1984). *MSX Technical Data Book*. : Sony Corporation

[3]: Adán Estrada (). *Un procesador de 8 bits. Z80-CPU*.

[4]: Gómez Andrián, J. A. (2016). *Evolución histórica de los lenguajes de programación. Un viaje a la historia de la informática*, 27-39. Valencia : UPV

[5]: Martí Campoy, A. (2016). Microcontroladores. *Un viaje a la historia de la informática*, 113-126. Valencia : UPV

[6]: Oniric Factory, Bit Knights. *Retro Sevilla: Programación en MSX y juegos Homebrew*. https://www.youtube.com/watch?v=ylZAN8WPpLU. [Consulta: octubre 2019]

[7]: Pritchard, Joe (1985). *Descubre tu MSX. Programación y aplicaciones*. Madrid: Anaya Multimedia

[8]: Pritchard, Joe (1988). *Lenguaje Máquina para MSX. Introducción y Conceptos avanzados*. Madrid: Anaya Multimedia

[9]: Sato Toshiyuki - Mapstone Paul - Muriel Isabella (1985). *MSX Guía del programador y manual de referencia*. Madrid: Anaya Multimedia

[10]: Burkinshaw, Chris - Goodley, Ross (1985). *Guía del Programador MSX*. Madrid: Ra-ma

[11]: Asín Gascón, Jesús - Asín Gascón, José L. (1986). *MSX - Más allá del Basic*. Madrid: Ra-ma

[12]: Edison A. Pires de Morales (1996). *MSX Top Secret*.

[13]: *Memòria MSX*. La Gaseta MSX (2016), nº 3, págs. 20-23.

[14]: *Mode de gràfics 2*. La Gaseta MSX (2016), nº 3, págs. 4-7.

[15]: *Sprites MSX*. La Gaseta MSX (2016), nº 3, págs. 8-13.

[16]: *MSX: hacia la compatibilidad*. Input MSX (1986), nº 1, págs. 52-54.

[17]: *Los sonidos del silicio*. Input MSX (1986), nº 1, págs. 7-12.

[18]: *El mapa de memoria de tu MSX*. Input MSX (1986), nº 3, págs. 50-56.

[19]: *El manejo de la VRAM en SCREEN 2*. Input MSX (1986), nº 4, págs. 10-13.

[20]: *La memoria de vídeo del MSX: Figuras móviles*. Input MSX (1986), nº 5, págs. 10-15.

[21]: *MSX-DOS, potente y versátil*. Input MSX (1986), nº 5, págs. 46-51.

[22]: *La memoria de vídeo de MSX, el VDP*. Input MSX (1986), nº 8, págs. 40-45.

[23]: MSX Resource Center. *MSX Wiki*. https://www.msx.org/wiki. [Consulta: agosto 2019]

[24]: Z80ST-Software. *MSX Inside*. http://www.z80st.es/cursos/msx-inside. [Consulta: agosto 2019]

[25]: Z80ST-Software. *A vueltas con las páginas de los cartuchos*. http://www.z80st.es/blog/2011/12/24-a-vueltas-con-las-paginas-de-los-cartuchos. [Consulta: agosto 2019]

[26]: FAQ MSX net. *The ultimate MSX FAQ*. https://www.faq.msxnet.org/ultmsxfaq.html#index. [Consulta: agosto 2019]

[27]: MSX Assemble Page. *MSX BIOS*. http://map.grauw.nl/resources/msxbios.php. [Consulta: agosto 2019]

[28]: Boez, E. - García, F. (2018). *FUSION-C. MSX C Library, complete journey*. : EBSoft Edition

[29]: GIMP. *GIMP, un potente editor de imágenes gratuito*. https://gimp.es/. [Consulta: agosto 2019]

[30]: Heras, R. - Herce, Gerardo. *NMSX tiles*. https://code.google.com/archive/p/nmsxtiles/. [Consulta: agosto 2019]

[31]: Orante, A. *MSX tiles devtool*. https://sites.google.com/site/multivac7/home/msx-tiles-devtool. [Consulta: agosto 2019]

[32]: Orante, A. *SpriteSX devtool*. https://sites.google.com/site/multivac7/home/spritesx. [Consulta: agosto 2019]

[33]: Bulba, S. *Vortex Track 2*. https://bulba.untergrund.net/vortex_e.htm. [Consulta: septiembre 2019]

[34]: ANT1's. *Tutorial Vortex Track 2*. http://www.culturachip.org/doku.php?id=tuto_vortex. [Consulta: septiembre 2019]

[35]: *Bomb Jack, la guía definitiva*. Retro Gamer (2013), nº 8, págs. 80-85.

[36]: Wikipedia. *Bomb Jack*. https://es.wikipedia.org/wiki/Bomb_Jack. [Consulta: septiembre 2019]

[37]: Vandal Retro. *Bomb Jack: el superhéroe de Tehkan*. https://vandal.elespanol.com/retro/bomb-jack-el-superheroe-de-tehkan. [Consulta: septiembre 2019]

[38]: StrategyWiki. *Bomb Jack*. https://strategywiki.org/wiki/Bomb_Jack. [Consulta: septiembre 2019]

[39]: The Brain Dump. *Bomb Jack Dissected*. https://floooh.github.io/2018/10/06/bombjack.html. [Consulta: septiembre 2019]

[40]: Games Database. *Bomb Jack: artwork-map*. https://www.gamesdatabase.org/media/sega-sg-1000/artwork-map/bomb-jack. [Consulta: septiembre 2019]

[41]: The Spriters Resource. *Bomb Jack*. https://www.spriters-resource.com/arcade/bombjack/sheet/86129/. [Consulta: septiembre 2019]

[42]: Lemon Amiga. *Bomb Jack*. https://www.lemonamiga.com/games/details.php?id=185. [Consulta: septiembre 2019]

[43]: VGMRips. *Bomb Jack*. https://vgmrips.net/packs/pack/bomb-jack-arcade. [Consulta: septiembre 2019]

[44]: Merino, Atila - Sánchez, Iván - Prini, Ignacio (2016). *Enciclopedia Homebrew. Vol. 1.* : Dolmen Editorial

Glosario

- **API:** acrónimo de las palabras inglesas *application programming interface* y en castellano Interfaz de programación de aplicaciones. Es un conjunto de subrutinas, funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro s*oftware* como una capa de abstracción.
- **Archivo de procesamiento por lotes (*batch)*:** archivo de texto sin formato, guardados con la extensión .BAT que contienen un conjunto de instrucciones DOS. Cuando se ejecuta este archivo, las órdenes contenidas son ejecutadas en grupo, de forma secuencial, permitiendo automatizar diversas tareas. Cualquier orden reconocible por el DOS puede ser utilizado en un archivo *batch*.
- **Bit**: unidad mínima de información que puede tener dos valores: cero y uno.
- **Bitmap**: acrónimo inglés de mapa de bits. Es una estructura o fichero de datos que representa una rejilla rectangular de *píxeles* o puntos de color.
- **Buffer (búfer)**: espacio de memoria en el que se almacenan datos de manera temporal.
- **BUS**: (o canal) conjunto de conductores eléctricos que transportan señales eléctricas representando dígitos binarios 0 ó 1 e interconectan los componentes del ordenador. Está formado por cables o pistas en un circuito impreso.
- **Byte:** conjunto de 8 bits que recibe el tratamiento de una unidad y que constituye el mínimo elemento de memoria direccionable de un ordenador.
- **Capa de abstracción de *hardware***: (en inglés, Hardware Abstraction Layer o HAL) es un elemento del sistema operativo que funciona como una interfaz entre el *software* y el *hardware* del sistema.
- **Chip**: circuito (integrado) electrónico de material semiconductor, especialmente silicio, en forma de cubo minúsculo que, combinado con otros componentes, forma un sistema integrado más complejo y realiza una función electrónica específica.
- **CISC**: acrónimo de las palabras inglesas *Complex Instrucction Set Computer*. Modelo de arquitectura de computadores donde su conjunto de instrucciones se caracteriza por ser muy amplio y permitir operaciones complejas entre operandos situados en la memoria o en los registros internos.
- **Clúster**: conjunto contiguo de sectores que componen la unidad más pequeña de almacenamiento de un disco.
- **Compilación cruzada**: proceso por el cual se desarrolla *software* ejecutable en plataformas u ordenadores distintos de aquellos que son los destinatarios del *software* desarrollado. En este proceso se genera código ejecutable para otra plataforma distinta a la que compila. El código ejecutable generado no es

interpretable por la plataforma que lo ha compilado sino por la plataforma destinataria.

- **CPU**: Unidad Central de Proceso, del acrónimo de las palabras inglesas *Central Process Unit*. Es el microprocesador principal, el circuito integrado cuya tarea es la de tomar ordenadamente las instrucciones del programa almacenado en la memoria de programa e ir interpretando su significado.
- **Dirección física**: dirección que corresponde con una ubicación en una unidad de memoria.
- **Dirección lógica**: dirección virtual generada por la CPU durante la ejecución de un programa.
- **Direccionamiento**: capacidad de acceder a una palabra o parte determinada de la memoria.
- **Espacio de direcciones**: conjunto de todas las posiciones de memoria o registros que son direccionadas por la CPU.
- **FAT**: tabla de asignación de archivos, comúnmente conocido como FAT (del inglés *File Allocation Table*). Es un sistema de archivos popular para disquetes admitido prácticamente por todos los sistemas operativos existentes para computadora personal.
- **Firmware**: programa que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.
- **HALT**: instrucción muy útil que detiene el proceso de ejecución. Al llamarla, la CPU comienza a ejecutar continuamente instrucciones vacías (NOPs) sin incrementar el contador de programa (CP) hasta que se vea interrumpido por una interrupción, en cuyo momento se incrementa CP y se procesa la interrupción.
- **Hardware**: partes físicas, tangibles, de un sistema informático; sus componentes eléctricos, electrónicos, electromecánicos y mecánicos. Los cables, así como los gabinetes o cajas, los periféricos de todo tipo y cualquier otro elemento físico involucrado, componen el *hardware* o soporte físico.
- **Homebrew**: palabra de origen inglés que suele denominar a las aplicaciones y juegos caseros no oficiales. Programas creados por programadores aficionados o expertos para cualquier plataforma, generalmente videoconsolas oficiales.
- **Interfaz**: elemento del sistema que interconexiona la CPU o el ordenador con otros componentes auxiliares del sistema; a estos componentes se les denomina periféricos.
- **Joystick**: palanca o mando de juegos.
- **Little endian**: formato de almacenamiento de bytes en la memoria donde el byte menos significativo se almacena en la posición más baja de la memoria mientras que el más significativo se guarda en la posición alta consecutiva.
- **MegaRAM**: cartucho que contiene chips de memoria RAM. La RAM se asigna como un MegaROM.

- **MegaROM**: cartucho con una ROM de al menos 1 mega-bit (128kB o más). Estos usan un gestor de memoria llamado *mapeador* destinado a extender la memoria a más de 64kB. Sin embargo, puede haber cartuchos que usen dicho *mapeador* mientras su tamaño de ROM sea de 64kB o menos.
- **Memoria expandida**: memoria adicional que sirve de extensión de la memoria principal o puede ser usada como memoria secundaria muy rápida.
- **Mhz**: acrónimo de Megahercio. El hercio es la unidad de frecuencia que equivale a la frecuencia de un fenómeno periódico cuyo periodo es 1 segundo.
- **MIPS**: acrónimo de *millones de instrucciones por segundos*. Es una forma de medir la potencia de los microprocesadores.
- **Palabra de datos**: Palabra de ordenador que es parte de los datos que procesa el ordenador, en oposición a palabra de instrucción y que también es conocida como palabra de información.
- **Peek**: instrucción del lenguaje de programación BASIC —también implementada para el lenguaje de programación C en la librería Fusion-C— que lee el valor de una determinada dirección de memoria. Ambos datos dados como argumento.
- **Píxel**: acrónimo del inglés *picture element*. Es la menor unidad homogénea en color que forma parte de una imagen digital.
- **Poke**: instrucción del lenguaje de programación BASIC —también implementada para el lenguaje de programación C en la librería Fusion-C— que almacena en una dirección de memoria determinado valor. Ambos datos dados como argumento.
- **PPI**: acrónimo de las palabras inglesas *Programmable Peripheral Input/Output*. Es el circuito integrado que controla y programa la entrada y salida a los periféricos (teclado, casete, impresoras...).
- **PSG**: acrónimo de las palabras inglesas *Programmable Sound Generator*. Chip encargado de producir sonidos.
- **RAM**: acrónimo de las palabras inglesas *Random Access Memory* que significa memoria de acceso aleatorio. Tipo de memoria que permite las operaciones de lectura y escritura sobre ella.
- **ROM**: acrónimo de las palabras inglesas *Read Only Memory* que significa memoria de sólo lectura. Sólo permite sobre ella realizar operaciones de lectura.
- **Script**: véase Archivo de procesamiento por lotes (*batch)*.
- **SDK**: acrónimo de las palabras inglesas *Software Development Kit* (Kit de desarrollo de *software*). Conjunto de herramientas de desarrollo de *software* que permite a un desarrollador de *software* crear una aplicación informática para un sistema concreto, por ejemplo ciertos paquetes de *software*, entornos de trabajo, plataformas de *hardware*, computadoras, videoconsolas, sistemas operativos, etcétera.
- **Sistema embebido**: *Una máquina o sistema que utiliza un ordenador para su gobierno recibe el nombre el nombre de sistema empotrado o embebido,*

independientemente de si el ordenador es un microcontrolador, microordenador u otra variante de ordenador. [Martí Campoy, A] MICROCONTROLADORES.

- **SLOT**: ranura de ampliación en la que pueden conectarse nuevos periféricos o incrementar la memoria RAM del sistema.
- **Software**: soporte lógico de un sistema informático. Comprende el conjunto de programas necesarios que hacen posible la realización de tareas específicas.
- **Sprite**: grupo de *píxeles* cuya forma y color pueden ser definidas por el usuario y que pueden desplazarse por la pantalla como un solo bloque.
- **URL**: siglas en inglés de *Uniform Resource Locator (*Localizador uniforme de recursos). Es un identificador, una dirección específica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que estos puedan ser localizados o identificados.
- **VDP:** acrónimo de las palabras inglesas *Video Display Processor*. El chip dedicado a gráficos.
- **VRAM**: memoria RAM dedicada exclusivamente para mantener la infomación usada por el chip de video. Sólo el chip de video tiene acceso a ella.