



Universidad
Zaragoza

TRABAJO FIN DE GRADO

Ingeniería Inversa sobre el Videojuego
MIDI Maze

AUTOR:

Jesús Ángel González Gorrado

DIRECTOR:

Eduardo Mena Nieto

TITULACIÓN:

Grado en Ingeniería Informática

Escuela de Ingeniería y Arquitectura, Zaragoza

Noviembre 2022

RESUMEN

El objetivo principal es intentar que el videojuego MIDI Maze (1987)¹, desarrollado para el ordenador Atari ST (1985–1993)², y que permite un modo multijugador a través de puertos *MIDI* (*Musical Instrument Digital Interface*)³, consiga comunicarse de la misma manera con un PC ejecutando un clon de MIDI Maze para Windows. Para ello habrá que hacer ingeniería inversa del protocolo de comunicación que emplea el juego sobre el MIDI.

El protocolo *MIDI* fue creado en 1983 con la intención de comunicar dispositivos musicales electrónicos. El ordenador Atari ST traía de fábrica dos puertos MIDI para poder conectarse a instrumentos musicales. La empresa Hybrid Arts aprovechó esta capacidad de comunicación para desarrollar el primer videojuego multijugador en red de la historia, permitiendo conectar hasta 16 ordenadores Atari ST.

En este proyecto se analiza la posibilidad de poder conectar a esa red MIDI del juego MIDI Maze tanto ordenadores Atari ST (ejecutando el juego original) como PCs (ejecutando un clon de dicho juego para Windows), todos ellos “hablando” MIDI. Para ello se darán los siguientes pasos:

- Familiarizarse con MIDI y su funcionamiento en Windows, ya que es una tecnología al margen de los contenidos de la carrera y era totalmente necesario saber su funcionamiento y características.
- Intentar comunicar MIDI Maze con un PC mediante el protocolo MIDI realizando ingeniería inversa sobre dicho juego. Primero lo comunicaremos con un PC mediante MIDI para monitorizar su funcionamiento, y después analizaremos el propio MIDI Maze para obtener más información de su uso de MIDI.
- Creación de un clon de MIDI Maze para Windows. Una vez realizado todo el estudio anterior se va a crear un clon de MIDI Maze para Windows que integre las funcionalidades necesarias para la realización del proyecto.

¹https://en.wikipedia.org/wiki/MIDI_Maze

²https://es.wikipedia.org/wiki/Atari_ST

³<https://es.wikipedia.org/wiki/MIDI>

AGRADECIMIENTOS

Dar gracias a Eduardo por las horas invertidas en este proyecto, la ayuda e ideas aportadas durante el mismo y sobre todo los ánimos aportados durante estos meses, ya que es en parte gracias a ellos que el proyecto ha salido adelante.

También dar las gracias a todos mis amigos cuyo apoyo ha resultado inestimable todos estos meses. Agradecer directamente a Carlos, David y Marina por dedicar parte de su tiempo libre en ayudarme a repasar y corregir gramaticalmente el proyecto.

Índice general

1. Introducción	4
1.1. El juego MIDI Maze de Atari ST	4
1.1.1. El protocolo MIDI	4
1.1.2. El juego MIDI Maze de Atari ST	7
1.2. Clon de MIDI Maze para PC desarrollado para la asignatura optativa Video- juegos	8
1.3. Objetivos del proyecto	9
1.4. Estructura de la memoria	10
2. Monitorización de las comunicaciones del juego MIDI Maze	11
2.1. <i>Software, hardware</i> y pruebas realizadas	11
2.1.1. Pruebas de MIDI en PC	12
2.2. Pruebas de red MIDI entre MIDI Maze en Atari ST y PC	13
2.2.1. Pruebas con MIDI <i>THRU</i>	14
2.2.2. Envío de mensajes MIDI <i>raw</i> desde PC a MIDI Maze en Atari ST . .	15
2.2.3. Resumen pruebas realizadas y los resultados obtenidos	16
2.3. Conclusión de las pruebas y medidas a tomar	17
2.4. Análisis del código del videojuego MIDI Maze	17
2.4.1. Protocolo MIDI usado por MIDI Maze	18
2.4.2. Estudio del código ensamblador de MIDI Maze	20
3. Desarrollo de un clon de MIDI Maze para PC	24
3.1. Diseño de la comunicación MIDI del clon de MIDI Maze para PC	25
3.2. Comunicaciones MIDI en el clon MIDI Maze para PC	26
4. Conclusiones	30
4.1. Cronograma	31
4.2. Posibles ampliaciones	31
4.3. Opinión personal	32
Bibliografía	33
A. Entrevista a Markus Fritze traducida	35
A.1. Preguntas y respuestas	35
A.2. Documentación del protocolo MIDI usado por MIDI Maze Atari ST	36
B. Análisis del código ensamblador M68000 del MIDI Maze de Atari ST	39
C. Materiales de MIDI Maze Atari ST	66
D. Hardware prestado por la <i>Asociación RetroAcción</i>	70

Capítulo 1

Introducción

En este capítulo se va a describir el objetivo del presente Trabajo de Fin de Grado así como el marco y contexto en el que se ha realizado, el juego MIDI Maze de Atari ST, el protocolo MIDI, el clon de MIDI Maze desarrollado en la asignatura optativa Videojuegos, así como la descripción de la estructura de la memoria.

1.1. El juego MIDI Maze de Atari ST

MIDI Maze es un videojuego en primera persona y multijugador para el ordenador Atari ST, el cual fue publicado en 1987 por *Hybrid Arts*. Fue el primer juego *FPS* (*First Person Shooter*) en red, destacando sobre todo la implementación de un modo multijugador y aumentaba los grados de libertad a la hora de moverse por el mapa 3D (simulado).

MIDI Maze fue el precursor de algunos conceptos que han llegado a nuestros días, como el de *deathmatch*¹ (modo de juego cuyo objetivo es el de eliminar a los contrincantes), o el de juego multijugador en red local, ya que permitía la conexión de hasta 16 jugadores simultáneamente haciendo uso de los puertos MIDI de la consola.

1.1.1. El protocolo MIDI

MIDI es un estándar tecnológico que describe un protocolo, una interfaz digital y conectores que permiten que varios instrumentos musicales electrónicos, ordenadores y otros dispositivos relacionados se conecten y comuniquen entre sí.

El protocolo MIDI transporta mensajes que identifican distintos eventos, dentro de un estándar que define que mensajes pertenecen al protocolo y su interpretación (ver Figura 1.1). Estos mensajes son eventos enviados por dispositivos mediante un cable MIDI a otros dispositivos, o instrumentos que controlan la generación de sonido, por ello MIDI no controla el envío de sonido si no que identifica eventos a ser reproducidos por otros dispositivos, es decir, es como enviar una partitura por una red, no defines como tiene que sonar, si no qué tiene que sonar, por ello un mensaje MIDI válido para el protocolo MIDI indica el tipo de acción a realizar (pulsar la nota, soltarla, etc...), el canal (los dispositivos MIDI disponen de hasta 16 canales), número de nota (codifica de forma numérica las notas de las que dispone

¹<https://es.wikipedia.org/wiki/Deathmatch>

un dispositivo musical), velocidad de la acción (valor desde 0 hasta 127), etc...

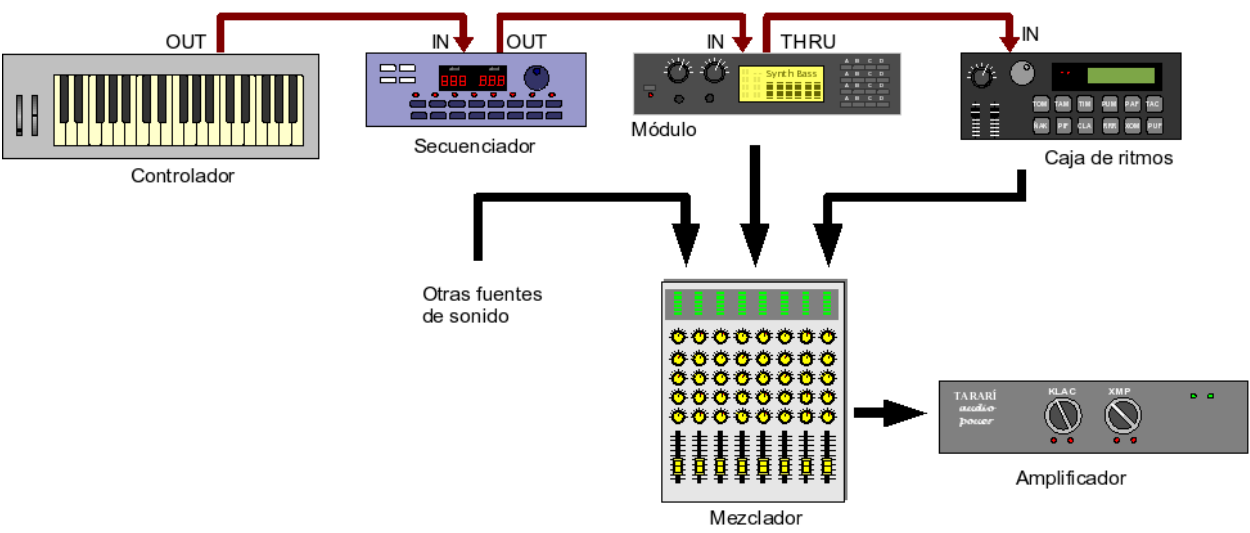


Figura 1.1: Dispositivos MIDI conectados en una red MIDI controlados mediante un teclado.

Los dispositivos MIDI se comunican a través de puertos MIDI con cables MIDI, dichos puertos pueden ser de tres tipos dependiendo de su función (Ver Figura 1.2):

- MIDI OUT: Puerto de salida de los mensajes MIDI producidos por el dispositivo.
- MIDI IN: Puerto de entrada de los mensajes MIDI que llegan al dispositivo, si no están dirigidos a ese dispositivo hay que redirigirlos al siguiente dispositivo de la red MIDI .
- MIDI THRU: La mayoría de los dispositivos MIDI no copian los mensajes de MIDI IN al MIDI OUT, por ello algunos dispositivos cuentan con el puerto MIDI THRU, el cual emite una copia de los mensajes recibidos por MIDI IN que van dirigidos a otro dispositivo de la red MIDI, esto permite crear redes MIDI en las que con un solo dispositivo emitiendo mensajes MIDI se pueden controlar múltiples dispositivos.

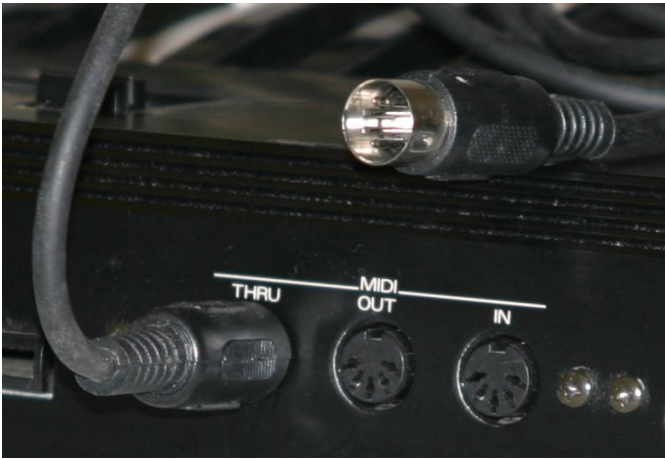


Figura 1.2: Cable y puertos MIDI.

Los cables MIDI terminan en un conector DIN5 de 180º ², que cuenta con 5 pines, cuya funcionalidad es distinta dependiendo de donde se aplica, en el caso de as aplicaciones MIDI estándar solo se usan 3 de los 5 conductores del cable: tierra (pin 2) y un par de cables balanceados (pines 1 y 3) que llevan un señal de 5V (ver Figura 1.3). Esta configuración del conector solo puede llevar mensajes en una dirección, así que es necesario un cable para una comunicación de dos vías, lo cual justifica que existan un puerto de entrada y otro de salida, y no uno solo con ambas funciones.

²https://es.wikipedia.org/wiki/Conector_DIN

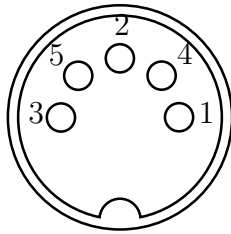


Figura 1.3: Pines numerados en un conector MIDI.

MIDI se estandariza en 1983, por *MIDI Manufacturers Association (MMA)* [11]. Al mismo tiempo, el mercado de los ordenadores personales se estabilizó, haciendo posible que los ordenadores fueran una opción viable para la producción musical. Con ello aparecieron distintos ordenadores que ya integraban la tecnología MIDI en sus diseños, como el Yamaha CX5M³ y el Atari ST..

Atari ST (ver Figura 1.4) es una línea de ordenadores domésticos presentadas por la empresa Atari⁴, la cual se lanza en el año 1985. Contaba con un microprocesador de 16 bits Motorola 68000⁵, y respecto a su predecesor una interfaz gráfica de mapa de bits en color.



Figura 1.4: Ordenador Atari ST (1985).

Atari ST fue diseñado como competidor del Macintosh⁶ presentado un año antes, el cual permitía instalar una interfaz MIDI. Siendo una de las principales características de Atari ST la introducción del estándar MIDI, lo cual integraba la producción musical en un ordenador personal, algo que previamente no era posible. Por ello, Atari ST también implementa el estándar MIDI en su diseño, incluyendo dos puertos MIDI, un puerto MIDI IN por el cual entran los mensajes MIDI dirigidos al Atari ST, y un puerto MIDI OUT por el cual se emiten los mensajes MIDI producidos por el Atari ST, así como una interfaz MIDI de serie.

La combinación de la potencia del estándar MIDI y la aparición de ordenadores personales que permitían su uso abrió muchas puertas, como es lógico en el campo de la producción musical, pero también en otros muchos campos, ya que como se ha definido el estándar, no comunica sonido, sino mensajes bajo un protocolo que son interpretados por los dispositivos adecuados. Por ello, fue muy usado como protocolo de control en aplicaciones no musicales. Esto dio lugar a numerosas innovaciones, una de ellas en el campo de los videojuegos.

³https://en.wikipedia.org/wiki/Yamaha_CX5M

⁴<https://es.wikipedia.org/wiki/Atari>

⁵https://es.wikipedia.org/wiki/Motorola_68000

⁶<https://es.wikipedia.org/wiki/Macintosh>

1.1.2. El juego MIDI Maze de Atari ST

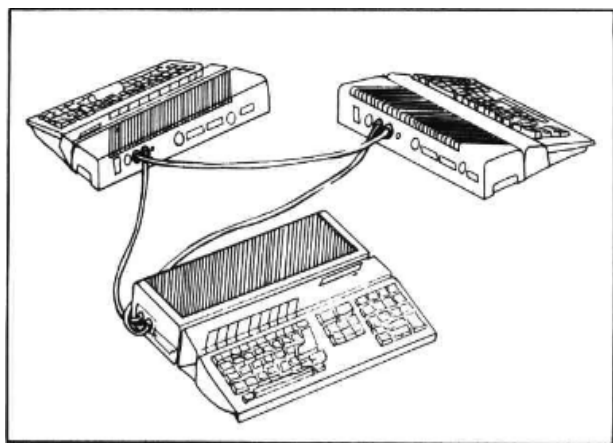
MIDI Maze es un videojuego para Atari ST publicado en 1987. Se trata de un FPS que tiene lugar en un laberinto. Los personajes están representados por un avatar *smiley*⁷ de diferentes colores y las balas que disparan como pequeñas esferas. El control del juego es simple, moverse adelante y atrás y rotar a la derecha o izquierda, con la particularidad de permitir una rotación completa del personaje, en contraposición a solo permitir cambios en la dirección de 90º grados, algo muy común en la época.

Una de las revoluciones que trae es el concepto de un modo multijugador LAN de hasta 16 jugadores simultáneos, haciendo uso de MIDI siendo así el primer videojuego en hacer uso de MIDI para conectar varias computadoras simultáneamente. Esto es posible formando un anillo⁸ entre los ordenadores con el juego a conectar, conocido en el marco del juego como *MIDI Ring*, mediante los puertos MIDI de Atari ST, de esta forma se puede conseguir comunicar distintos ordenadores haciendo uso del estándar MIDI como un protocolo de control (ver Figura 1.5).

En una partida multijugador se designan dos tipos de jugadores, el primero es el MASTER, el cual selecciona las reglas de la partida (ver Figura 1.6a):



(a) MIDI Maze en Atari ST.



(b) Representación del funcionamiento de un anillo MIDI entre tres Atari ST.

Figura 1.5: MIDI Maze para Atari ST.

- El tiempo de recarga de la bala, entre rápido o lento.
- El tiempo que se tarda en regenerar una vida perdida, entre rápido y lento.
- El tiempo que se tarda en revivir después de morir, entre rápido y lento.
- El número de vidas con las que se revive, indica el número de impactos que debe recibir un jugador antes de morir, pueden ser 1, 2 o 3 vidas.
- El número de drones que va a haber en la partida, teniendo en cuenta que el número de drones *Very Dumb*, *Plain Dumb*, *Not so Dumb* más el de jugadores en la red MIDI no debe sobrepasar 16. Los drones se categorizan según el nivel de “inteligencia”, siendo de menor a mayor *Very Dumb*, *Plain Dumb* y *Not So Dumb*. En el caso de *Very Dumb* se trata de drones que huyen de los jugadores y no les atacan, *Not So Dumb* ya no huyen y además atacan al jugador si se cruzan con el, aunque no tienen muy buena puntería, y en el caso de *Not so Dumb* son los drones más peligrosos, persiguen a los

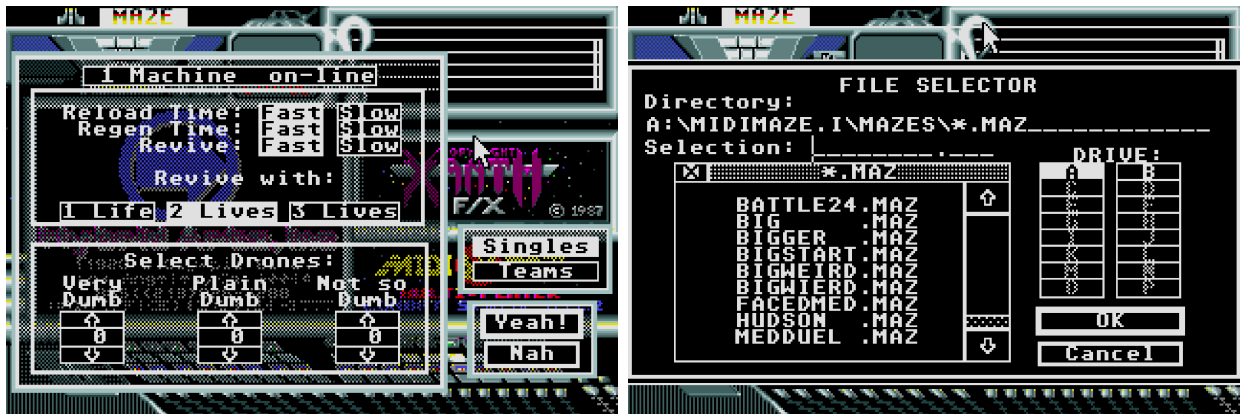
⁷<https://en.wikipedia.org/wiki/Smiley>

⁸https://en.wikipedia.org/wiki/Network_topology

jugadores y les disparan con casi una puntería perfecta.

- Selecciona si la partida va a ser todos contra todos o si se van a formar equipos entre los jugadores y los drones.

El MASTER selecciona el laberinto en el que se va a jugar, el propio juego incluye una serie de laberintos, pero también existe la opción de crear los tuyos propios ya que se trata de un archivo de texto donde la primera línea indica el tamaño del laberinto seguido del laberinto representando en el mismo los muros como x y los espacios vacíos como $.$, el laberinto debe ser cuadrado (ver Figura 1.6b). El segundo tipo de jugador es el SLAVE, el cual se limita a recibir la información del MASTER para la partida.



(a) Menú de opciones del MASTER en MIDI Maze.

(b) Selección de laberintos del MASTER en MIDI Maze.

Figura 1.6: Control del MASTER sobre la partida en MIDI Maze.

Al comienzo de la partida se coloca a todos los jugadores en posiciones aleatorias del laberinto, durante la partida el objetivo de los jugadores es eliminar al resto, obteniendo la victoria el primero en abatir a un total de 10 jugadores. Para eliminar a un jugador hay que acertarle con los disparos el número de veces que se haya configurado en la partida, 1, 2 o 3 veces, una vez conseguido el jugador muere y reaparece en otra posición aleatoria del laberinto.

En el Anexo C se pueden encontrar materiales interesantes de MIDI Maze, como su manual o la caratula del disquete con el que se comercializo.

1.2. Clon de MIDI Maze para PC desarrollado para la asignatura optativa Videojuegos

Se trata de un clon de MIDI Maze realizado en Windows para la asignatura optativa Videojuegos en el curso 2021/22 [9], el cual es casi idéntico al original, a falta de algunas características que no se implementaron, cabe destacar en este contexto el multijugador, el cual no esta en el clon realizado (ver Figura 1.7).

Este clon cuenta con ampliaciones respecto al juego MIDI Maze original, tiene un menú para controlarlas, y son, modificar el volumen del juego, modificar el control del juego y como se ve a unos gráficos y un movimiento más moderno, para ser más exactos se modifica el movimiento simulando el que hay en los FPS modernos, moverse con las teclas *WASD*, apuntar con el ratón y disparar con el botón izquierdo del mismo. Por su parte, los gráficos actualizados presentan el juego en pantalla completa, en contra parte a la pantalla reducida

del original, y modifican la *GUI*(*Graphical User Interface*)⁹ para adaptarse a la nueva pantalla sin perder ninguna información esencial para el juego. Por último se añade una nueva *IA* (*Inteligencia Artificial*)¹⁰ más refinada que la original y que presenta un desafío adicional para el jugador.



(a) Clon MIDI Maze PC.

(b) MIDI Maze Atari ST.

Figura 1.7: Comparativa MIDI Maze Atari ST con clon MIDI Maze PC.

1.3. Objetivos del proyecto

El objetivo principal del proyecto es intentar conseguir la comunicación entre el videojuego MIDI Maze (1987) ejecutándose en un Atari ST, y un clon del mismo videojuego ejecutándose en Windows, en una red MIDI (ver Figura 1.8). Para ello es necesario realizar ingeniería inversa sobre el videojuego original MIDI Maze para Atari ST. Para conseguir este propósito se definen los siguientes objetivos:

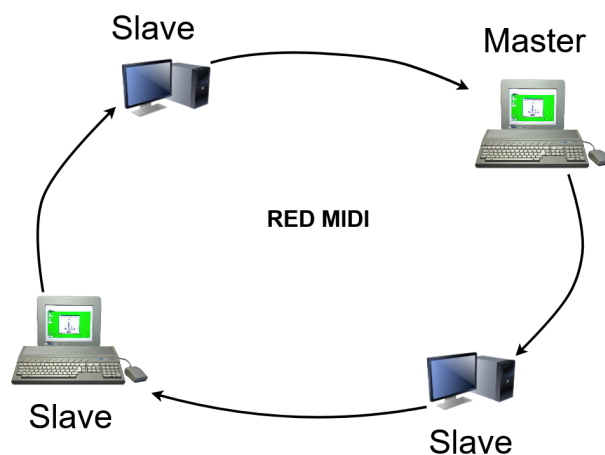


Figura 1.8: Funcionamiento de la red MIDI que se intentara diseñar entre MIDI Maze Atari ST y clon de MIDI Maze en PC.

- Familiarizarse con MIDI y su funcionamiento en Windows, ya que es una tecnología al margen de los contenidos de la carrera y era totalmente necesario saber su funcionamiento y características.
- Intentar comunicar MIDI Maze con un PC mediante el protocolo MIDI realizando ingeniería inversa sobre dicho juego. Primero lo comunicaremos con un PC mediante MIDI para monitorizar su funcionamiento, y después analizaremos el propio MIDI Maze para obtener más información de su uso de MIDI.

⁹https://en.wikipedia.org/wiki/Graphical_user_interface

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

- Creación de un clon de MIDI Maze para Windows. Una vez realizado todo el estudio anterior se va a crear un clon de MIDI Maze para Windows para integrar las funcionalidades necesarias para la realización del proyecto.

1.4. Estructura de la memoria

El resto del documento se organiza de la siguiente forma. El capítulo 2 trata principalmente de las pruebas realizadas sobre la red MIDI, para leer e interpretar los mensajes MIDI de Atari ST, los problemas encontrados y las soluciones propuestas, así como las conclusiones e información relevante obtenida del análisis del código en ensamblador del juego MIDI Maze original.

El capítulo 3 trata de las decisiones tomadas a raíz de los resultados del capítulo 2, la creación de una comunicación MIDI para jugar en multijugador, los cambios realizados en el clon para hacer esta comunicación posible y añadidos sobre el mismo respecto al entregado en la asignatura optativa Videojuegos para hacerlo más fiel al original.

Finalmente, el capítulo 4 contiene las conclusiones del trabajo realizado, un cronograma del mismo y posibles ampliaciones sobre el resultado final.

Ademas, en el Anexo A se adjunta traducida al Español una entrevista a Markus Fritze. En el Anexo B un análisis del código ensamblador de MIDI Maze. En el Anexo C un conjunto de materiales referentes a MIDI Maze. Y en el Anexo D un listado de los materiales prestados por la Asociación RetroAcción para la realización del proyecto.

Capítulo 2

Monitorización de las comunicaciones del juego MIDI Maze

Este capítulo presenta y describe las pruebas realizadas sobre la red MIDI para ser capaces de recibir los mensajes MIDI usados por MIDI Maze en su comunicación multijugador, a un software de visualización en PC. También describiremos el software y hardware usado en dichas pruebas y su función, así como un análisis en alto nivel del código en ensamblador Motorola 68000 de MIDI Maze.

Antes de comenzar con las pruebas, es necesario entender el estándar tecnológico MIDI, ya que todas las pruebas posteriores van a girar alrededor del mismo. Para ello se hace uso de libros [7], y sobre todo, documentación *online*, siendo la web de *MIDI Association* [11] una muy buena guía donde se puede encontrar resúmenes y especificaciones sobre el protocolo.

En el Anexo D se puede encontrar un listado del hardware prestado por la *Asociación RetroAcción*¹ para la realización de las pruebas.

2.1. *Software, hardware y pruebas realizadas*

Como prueba inicial se intenta conectar dos emuladores *Steem SEE*² mediante MIDI para poder comenzar una partida multijugador entre ambos emuladores, para así no recurrir a dos Atari ST físicas, ya que en ese momento no se disponía de ellos y realizar esta prueba de forma local es mucho más sencillo.

No se consigue conectar ambos emuladores, así que tras una breve búsqueda sobre el porque no era posible se descubre que el problema viene de que el emulador no ofrece soporte de las conexiones MIDI, por lo que no se puede crear un anillo entre emuladores. Se realiza la prueba con otro emulador de Atari ST, *Hatari*³ obteniendo el mismo resultado anterior.

Después se ha propuesto ver como funciona una partida multijugador entre MIDI Maze, el objetivo de este prueba es simular un ciclo de uso en el que dos jugadores quieren iniciar, jugar y finalizar una partida multijugador de MIDI Maze. Para ello con la ayuda de dos Atari ST se forma un anillo MIDI entre ellos y se comienza una partida multijugador, con esta

¹<https://www.retroaccion.org/>

²<https://sourceforge.net/projects/steemsse/>

³<https://hatari.tuxfamily.org/>

prueba se observa las etapas que tiene una partida multijugador como lo haría un usuario normal que quisiera jugar, con ello distinguimos las siguientes etapas:

- Una primera etapa en la cual se designa quien es el MASTER en el anillo y quienes los SLAVES.
- La selección de las opciones de la partida y el mapa por parte del MASTER, y comienzo de la misma.
- Condiciones de finalización de la partida:
 - Gana uno de los jugadores.
 - El MASTER finaliza la partida manualmente, lo que provoca que la partida finalice para todos los jugadores en el anillo.
 - Se rompe el anillo, esto es producido cuando uno de los cables se desconecta, rompiendo el anillo ya que la información no puede navegar por el.

Esta prueba también es útil para observar como se realiza la conexión de dispositivos mediante MIDI. En este caso como queremos conectar dos dispositivos, con el uso de cables MIDI, se conecta el puerto MIDI IN del primer Atari, con el MIDI Out del segundo, y viceversa, el MIDI In del segundo al MIDI Out del primero, formando así un anillo entre ellos.

2.1.1. Pruebas de MIDI en PC

El objetivo es comprobar la conexión MIDI a PC, es decir enviar a nuestro PC mensajes MIDI para tocar un teclado virtual. Para ello usamos un teclado físico (ver Figura 2.1b) que ofrece puertos MIDI IN y MIDI OUT, y lo conectamos mediante un adaptador USB-MIDI(ver Figura 2.1a) el cual cuenta con un puerto MIDI IN, un puerto MIDI OUT, y un puerto *USB* que se comporta tanto como MIDI IN como MIDI OUT y se conecta al PC, donde tenemos un teclado MIDI virtual con la aplicación *VMPK*⁴, y un monitor MIDI con el que capturar los mensajes y mostrarlos.



(a) Adaptador USB-MIDI para PC.



(b) Teclado MIDI.

Figura 2.1: Materiales MIDI.

Vamos a usar *MIDI View*⁵, por lo que conectamos al MIDI OUT del teclado físico el MIDI IN del adaptador y el USB al PC, y al tocar el teclado físico podemos observar que el teclado virtual también esta siendo tocado y que en el monitor podemos observar los mensajes MIDI

⁴<https://vmpk.sourceforge.io/>

⁵<https://hautetechnique.com/midi/midiview/>

del teclado. Ahora, para ver que la conexión funciona en ambas direcciones conectamos el MIDI IN del teclado físico al MIDI OUT del adaptador y ahora podemos observar como al pulsar una tecla en el teclado virtual también suena el teclado físico.

2.2. Pruebas de red MIDI entre MIDI Maze en Atari ST y PC

Una vez que hemos comprobado que la conexión funciona procedemos a conectar los Atari al PC, para ello se hace uso del adaptador USB-MIDI. Después con dos Atari ST, y dos cables MIDI se forma un anillo MIDI entre ambas, y se sustituye uno de los cables del anillo por el adaptador MIDI-PC, de tal forma que los mensajes enviados por el anillo también vayan al PC, y puedan ser visualizados mediante *MIDIView* (ver Figura 2.2). Como resultado de esta prueba podemos ver que al realizar cualquier acción en el MASTER la cual debería enviar un mensaje MIDI al anillo y por lo tanto pasar por nuestro adaptador el monitor MIDI no captura ningún mensaje. Como primera idea para solucionarlo se prueba con distintos monitores, sin resultado alguno en ninguno de ellos.

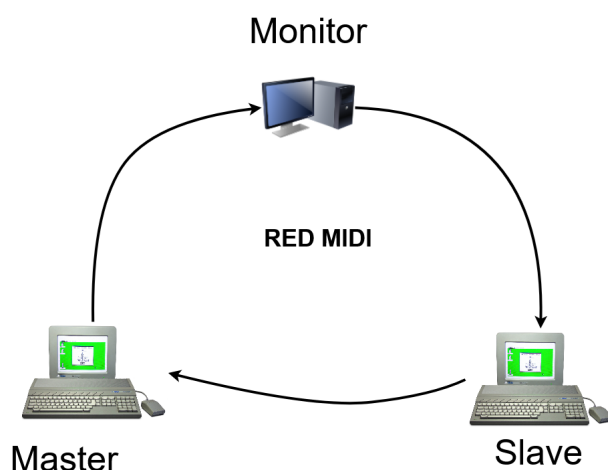


Figura 2.2: Anillo MIDI entre dos Atari ST con el PC mediante el adaptador USB-MIDI en medio para analizar los mensajes MIDI.

Tras investigar este problema, y a partir de una pregunta realizada en el foro *Atari-Forum* [1] se encontró información de que MIDI Maze envía los mensajes MIDI en formato *raw* y en *stream* de datos de 8 bits, lo que quiere decir que esta usando el puerto MIDI como un puerto serie, enviando bytes que no se ajustan a los mensajes del protocolo MIDI. Sobre todo destacar en las respuestas al usuario "sarnau", ya que se trata de *Markus Fritze*⁶ uno de los desarrolladores de MIDI Maze II el cual tiene un diseño muy similar a MIDI Maze.

Dada la información encontrada como resultado de la anterior prueba, se procede a buscar un monitor MIDI que permita capturar y visualizar mensajes MIDI *raw*, encontrando así el monitor *MIDI-OX*⁷ el cual cuenta con la opción de enviar y recibir MIDI *raw*. Por lo que volvemos a intentar capturar los mensajes de MIDI Maze, sin ningún resultado, el monitor no captura ningún mensaje. Según la información resultante de la consulta en el blog *Atari-Forum* si se usa un monitor MIDI dicho monitor solo detectaría mensajes MIDI completos, por lo que si MIDI Maze esta enviando mensajes *raw*, la recepción en el monitor puede fallar. En consecuencia se opta por usar un *sniffer* de puertos directamente, en este caso *Wireshark* (ver Figura 2.3).

⁶<http://www.sarnau.com/>

⁷<http://www.midiox.com/>

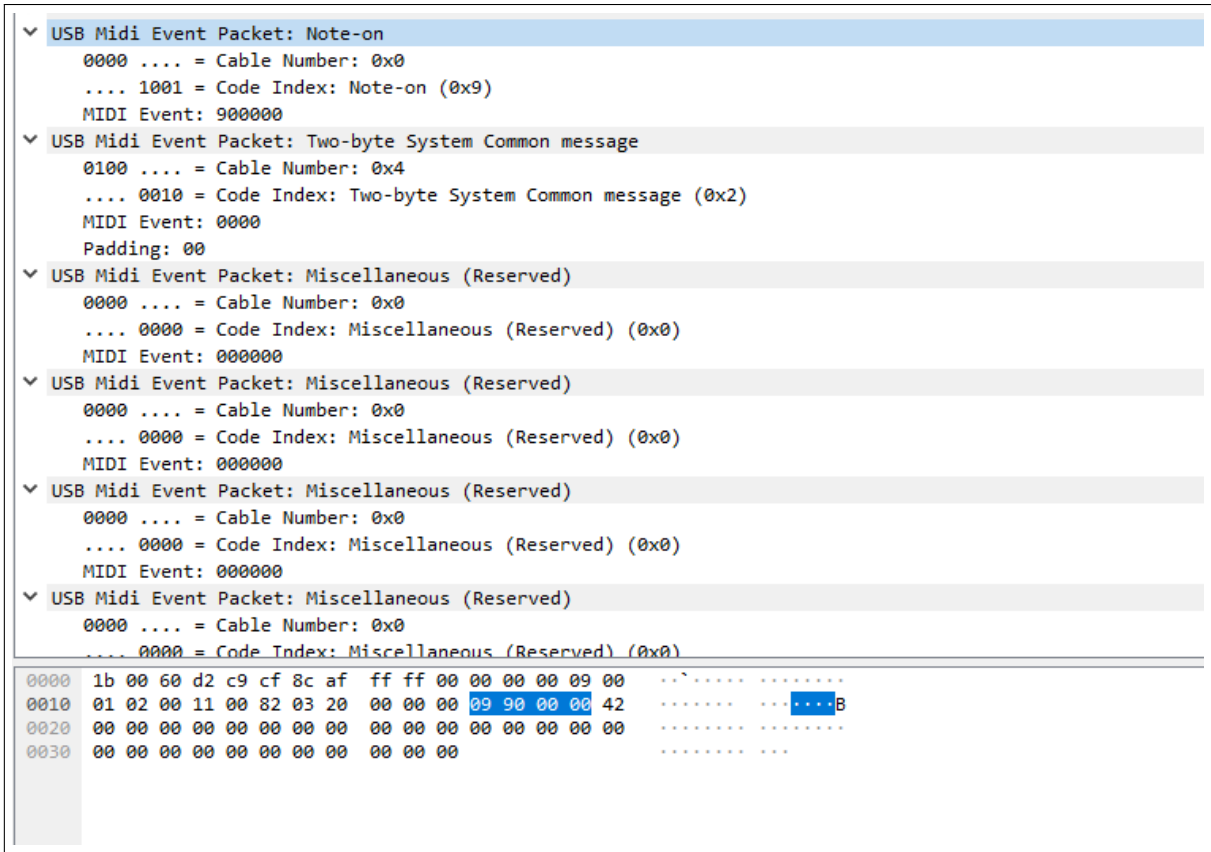


Figura 2.3: Evento MIDI Note-ON capturado con Wireshark.

El resultado cambia y empezamos a recibir el mensaje *0x90 00 00* el cual es un mensaje dentro del estándar MIDI, lo que entra en conflicto con la información encontrada de que MIDI Maze envía sus mensajes MIDI en MIDI *raw* de 8 bit.

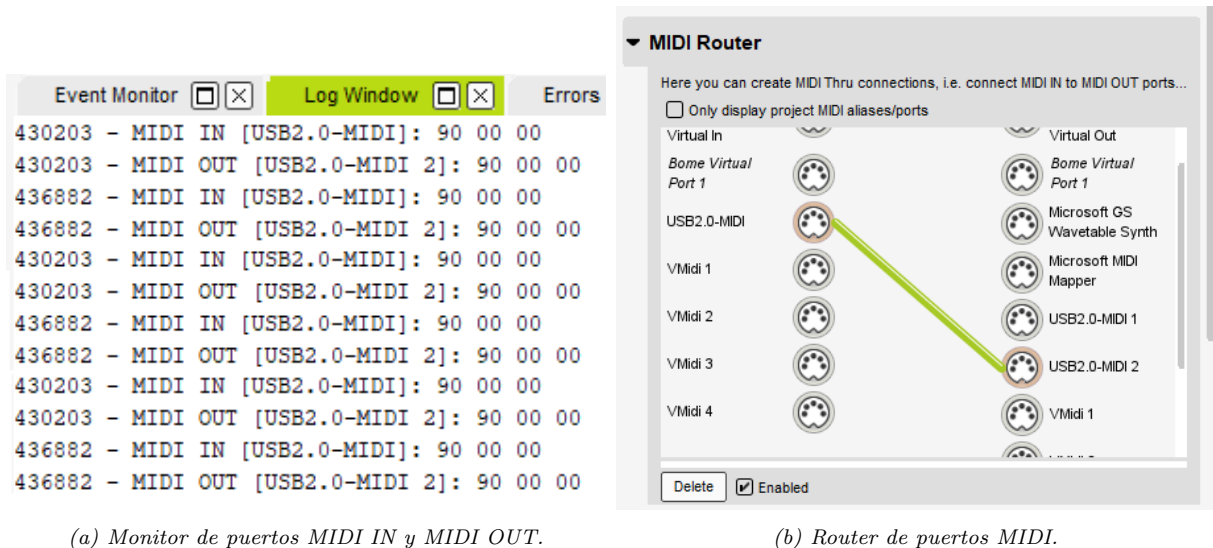
2.2.1. Pruebas con MIDI *THRU*

Se vuelve a investigar el problema en el recibimiento de mensajes, encontrando así una posible solución, el MIDI *THRU*, se trata de una conexión lógica la cual duplica los datos provenientes de MIDI IN y lo reenvía por MIDI OUT, por lo que aunque esto no solucione el problema en la captura de los datos, puede ser el primer paso para ello, ya que podríamos tener un anillo funcionando con el adaptador USB-MIDI en medio. Así que investigando posibles *softwares* para realizar un MIDI *THRU*, se encuentra *Bome's Midi Translator*⁸ un monitor con una interfaz mediante la cual se puede realizar MIDI *THRU* de un puerto a otro.

Al realizar esta prueba nos encontramos que, teniendo el MIDI *THRU* configurado y el propio programa funcionando como monitor a la vez ocurre lo siguiente, al recibir un mensaje MIDI de MIDI Maze no ocurre nada y el monitor no captura dicho mensaje, si volvemos a enviar un mensaje al anillo, el monitor captura el mensaje y lo muestra como *90 00 00* y lo reenvía mediante MIDI *THRU* a la salida, por lo que lo reenvía al anillo, pero el Atari que debe recibirlo solo parpadea durante un momento (ver Figura 2.4). A raíz de esto se decide investigar directamente que mensajes concretamente envía MIDI Maze, ya que como conclusión a las pruebas hasta ahora se piensa que el problema puede estar en los propios mensajes de MIDI.

Tras investigar, se llega a una *wiki* en *Github* de una implementación de *MIDI Maze II*, en dicha *wiki* se describe como se ha implementado el modo multijugador por MIDI en el

⁸<https://www.bome.com/products/miditranslator>



(a) Monitor de puertos MIDI IN y MIDI OUT.

(b) Router de puertos MIDI.

Figura 2.4: Realización y monitorización de MIDI THRU mediante el router de puertos MIDI de Bome's MIDI Translator.

propio *MIDI Maze II*, con el resumen de una entrevista a *Markus Fritze*, el cual asegura que el protocolo usado es casi idéntico al de *MIDI Maze*. Al fijarnos en el protocolo podemos ver que cuando un jugador quiere ser MASTER, al inicial el juego o al pulsar *Alt-R* se envía por el puerto MIDI de Atari *0x00* al anillo, en hexadecimal o lo que es lo mismo *00000000* en binario, y espera recibir el mismo mensaje. Se puede observar que en las pruebas anteriores siempre se ha intentado enviar mensajes MIDI respecto al protocolo [4], y el mensaje *0x00* no existe en dicho protocolo.

Como consecuencia, se genera la hipótesis de que el problema puede estar en intentar realizar un MIDI THRU de forma virtual, ya que THRU es una conexión lógica, así que se procede a intentar hacerlo físicamente, con un *patchbay* el cual ya viene provisto de un MIDI THRU. Para ello se elimina el adaptador del anillo y se recolocan los cables MIDI, de tal forma que el OUT de un Atari ST vaya al IN del *patchbay* y del OUT del *patchbay* al IN del Atari ST. El resultado obtenido es que el *patchbay* ni siquiera consume el mensaje del IN del Atari ST, dado que tiene unos LEDs que informan de ello. El funcionamiento del *patchbay* a sido comprobado anteriormente, con la misma prueba inicial del teclado, por lo que se descarta un posible fallo del *patchbay*.

2.2.2. Envío de mensajes MIDI raw desde PC a MIDI Maze en Atari ST

Como siguiente prueba se busca enviar datos MIDI totalmente *raw* a mano al anillo, mediante el adaptador MIDI-PC, por ello se llega a un proyecto llamado *sendmidi*⁹, desarrollado en C++, que ofrece una interfaz para una librería llama *JUCE* [13] la cual se trata de un *framework* de código abierto para desarrollar aplicaciones de sonido, por lo que permite la creación, envío y recibimiento de MIDI, siendo una de sus aplicaciones enviar mensajes MIDI *raw* al puerto MIDI seleccionado.

Hacemos uso de *sendmidi* para enviar al anillo el mensaje *0x00* que espera el MIDI Maze, pero sin reacción alguna del Atari ST al que se le envía el mensaje. Por ello, se decide realizar una prueba intermedia más, mediante el uso de puertos virtuales de MIDI, usando

⁹<https://github.com/gbevin/SendMIDI>

el software *loopMIDI*¹⁰ creamos un puerto MIDI virtual, y mediante *sendmidi* enviamos al puerto virtual *0x00* con un monitor escuchando al mismo puerto, como resultado el monitor no captura dicho mensaje, además que *loopMIDI* muestra para un puerto virtual el número de bytes que han sido enviados por el, y si nos fijamos en el puerto indica 0 bytes, por lo que todo indica a que el problema no esta en el monitor, si no en una etapa anterior, el propio envío del mensaje.

Teniendo esto en cuenta, el adaptador USB-MIDI dispone de dos *LEDs*, uno que indica cuando un mensaje entra por el MIDI IN y otro que indica cuando sale por el MIDI OUT, por ello volvemos a conectar el anillo de Atari, con el adaptador en medio conectado a PC e intentamos capturar el mensaje enviado por el Atari ST, con lo que el *LED* correspondiente a un mensaje dirigido del Atari al PC se enciende, lo cual indica que MIDI Maze esta enviando un mensaje que llega al adaptador. Ahora intentamos enviar *0x00* mediante un *raw* bit con *sendmidi* con lo que vemos que el *LED* de MIDI IN del adaptador no se enciende, mientras que si cambiamos el mensaje a *0x90 00 00*, un mensaje MIDI que sigue el protocolo, el *LED* de MIDI IN si que se enciende. En este momento, para comprobar si el fallo era del adaptador se hace la misma prueba con otro distinto, obteniendo exactamente los mismos resultados.

2.2.3. Resumen pruebas realizadas y los resultados obtenidos

Por lo que, en este punto y dado que al enviar a un puerto MIDI virtual *0x00* no llegaba al puerto, que al adaptador USB-MIDI en medio del anillo llegan los mensajes procedente de MIDI Maze, pero desde PC no llega *0x00* pero si *0x90 00 00*, se forma la hipótesis de que el bloqueo se encuentra en PC, ya que si además nos fijamos en como funciona el adaptador USB-MIDI se trata de un hardware muy simple, sin margen para programar algo en el.

Partiendo de esta hipótesis se comienza a investigar y se llega a una pregunta en el foro de usuario de *MIDI-OX* [5] en la cual un usuario pregunta como reconocer *data streams raw* de 1 byte, exactamente nuestra situación respecto a los mensajes de MIDI Maze. A lo que la respuesta es que no es posible, ya que los drivers MIDI que usa Windows bloquean todo mensaje MIDI que no sea valido.

A partir de Windows XP, el sistema operativo se comunica con los controladores MIDI *input* y *output* mediante mensajes de dispositivo de audio. Esto esta controlado de *WINMM* [2] que es una biblioteca de vínculos dinámicos (DDL)¹¹ de Windows, cada controlador define una función de entrada y salida, a la cual se le asocia un función del *DDL* llamada *modMessage* [6] la cual recibe un puntero al mensaje de dispositivo de audio a ejecutar, dicho mensaje de dispositivo de audio es determinado por *WINMM* y son constantes definidas por el sistema, por lo que cuando *modMessage* recibe un mensaje de dispositivo de audio usa un *switch*¹² para determinar la acción que se va a realizar, en función del valor del mensaje.

¹⁰<https://www.tobias-erichsen.de/software/loopmidi.html>

¹¹https://en.wikipedia.org/wiki/Dynamic-link_library

¹²https://es.wikipedia.org/wiki/Switch_case

2.3. Conclusión de las pruebas y medidas a tomar

Dados los resultados obtenidos en las pruebas, y la información conseguida fruto de ellas, llegamos a varias conclusiones. La primera, respecto a *WINMM.dll* de Windows, al investigar los mensajes de dispositivo de audio que define, ninguno contempla la posibilidad de recibir como parámetro un solo byte de datos, lo cual, según la información proporcionada por *Markus Fritze* que define el protocolo MIDI usado por MIDI Maze, es el causante de que la comunicación entre PC y MIDI Maze en Atari ST no sea posible de conseguir, al menos con los medios de los que disponemos, ya que el propio sistema operativo bloquea los mensajes pertenecientes al protocolo de MIDI Maze ya que no los concibe como mensajes validos, esto se debe a que los drivers de MIDI de Windows definen un punto de entrada a todas las conexiones MIDI que tenga disponible, tanto de entrada, como de salida, como *THRU*, dicha entrada recibe el dispositivo que envía la petición, a que dispositivo la dirija, y los datos de la petición, con esos datos asigna la petición a uno de los mensajes de dispositivo de audio definidos en *WINMM.dll* y si no coincide con la descripción de ninguno de ellos desecha la petición recibida. De ello podemos justificar algunos de los resultados obtenidos en las pruebas:

- En el caso de el envío de mensajes MIDI desde Atari ST a PC, el adaptador indicaba recibir el mensaje de Atari ST pero dicho mensaje nunca llegaba al PC, esto se debía a que Windows recibía la petición pero la desechaba, dado que no pertenecía a ningún mensajes de dispositivo de audio. En el caso inverso, PC enviando datos a Atari ST, el adaptador no indicaba recibir el mensaje del PC, de nuevo, se debe a que al realizar la petición del mensaje, Windows no lo asociaba a un mensajes de dispositivo de audio valido.
- En el caso de las pruebas internas con puertos virtuales, el resultado se justifica como en el punto anterior, ya que es la misma situación, solo que internamente en PC, pero esto justifica porque los puertos virtuales no detectaban paso de datos.

Por ello se decide analizar directamente el código ensamblador de MIDI Maze, ya que en el se puede ver directamente que mensajes MIDI se envían y reciben en la comunicación, para así asegurar que la hipótesis y las conclusiones a las que se ha llegado son correctas.

2.4. Análisis del código del videojuego MIDI Maze

En este capítulo se va a presentar la información obtenida sobre el protocolo MIDI usado por MIDI Maze, así como el análisis realizado sobre el código ensamblador del juego para Atari ST.

Se ha realizado ingeniería inversa sobre el código ensamblador para identificar directamente que mensajes MIDI se envían en la comunicación de MIDI Maze. Para ello se ha usado el emulador *Steem SSE*, más concretamente la versión 3.3.0 la cual incluye la opción de *debug*, con la que podemos manipular el código ensamblador que se este ejecutando, por lo que con la imagen de MIDI Maze podemos ejecutarla en el emulador y descargar todo el código ensamblador del juego para así poder analizarlo (ver Figura 2.5). Los mensajes MIDI usados por MIDI Maze son mensajes *raw* los cuales no siguen el estándar de comunicación MIDI, y

dicha comunicación se realiza en varios pasos. Se va a realizar un resumen en alto nivel de los aspectos más importantes del protocolo usado por MIDI Maze.

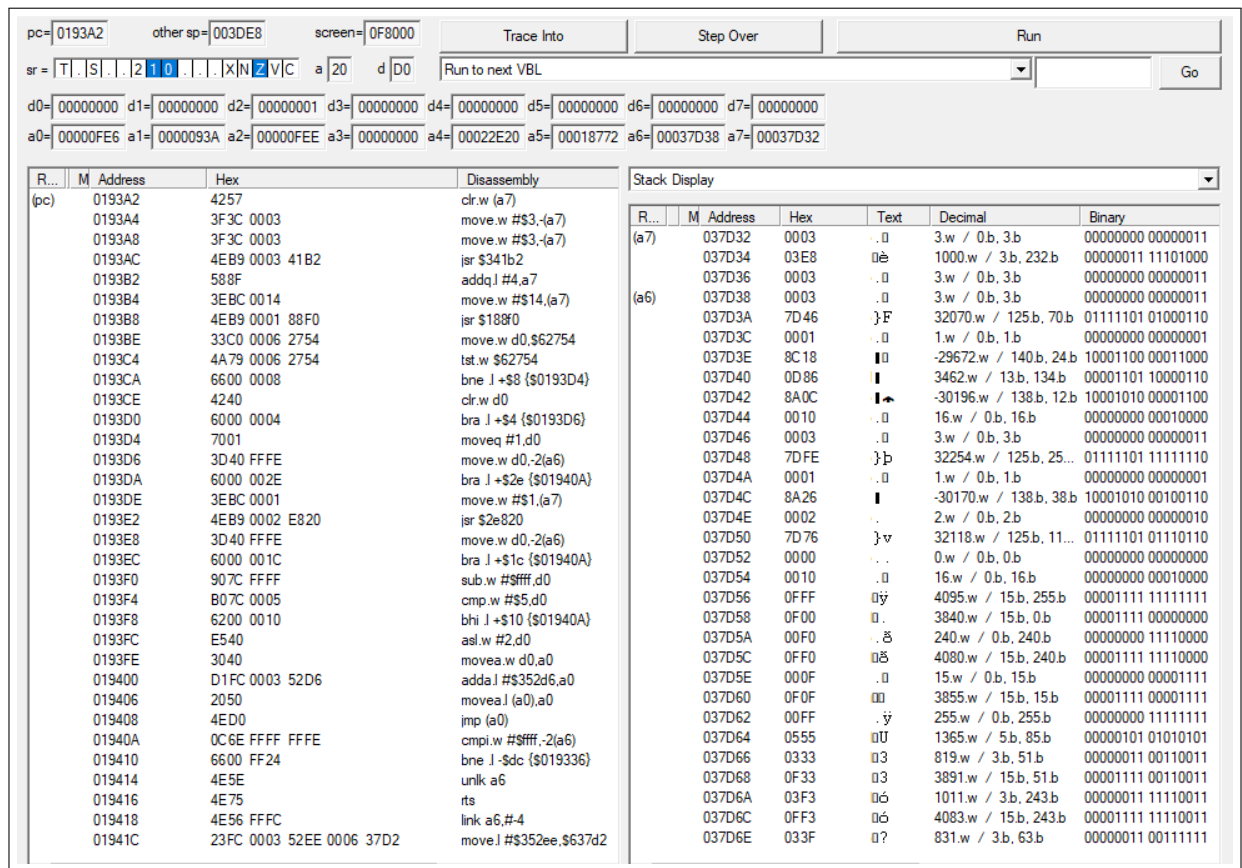


Figura 2.5: Interfaz del debugger de Steem SEE con MIDI Maze

2.4.1. Protocolo MIDI usado por MIDI Maze

Primero se establece la conexión entre los distintos clientes en el anillo, designando así a un MASTER, el cual se encarga de seleccionar un mapa y las opciones de la partida, y comenzar la partida con los parámetros seleccionados. Una vez dentro de la partida cada cliente envía un byte de datos en el *bitmask* estándar de joystick, el cual es un byte donde los 5 primeros bits representan *left, right, down, up, button*, en dicho orden, donde si el bit es 1 significa que se ha pulsado dicho botón, y si es 0 que no. A cada cliente le llega el byte de joystick de todo el resto de clientes, por lo que la velocidad del juego se reduce a la velocidad del anillo y lo que tarde cada cliente en enviar su byte, recibir el del resto y realizar los cálculos necesarios para el siguiente bucle. Durante la partida el MASTER puede decidir finalizarla, es el único comando que se reconoce durante la partida.

Ahora, se van a describir los mensajes que envía MIDI Maze en su comunicación, descritos los más importantes por unas etiquetas que los definen. La descripción del protocolo viene dada por *Markus Fritze* uno de los desarrolladores de *MIDI Maze II*:

- Una propiedad importante del protocolo, es que cuando un SLAVE recibe un mensaje, lo reenvía por su MIDI OUT, lo cual también implica que un MASTER siempre espere recibir su mensaje de vuelta al enviar un mensaje al anillo.
- Cuando se lanza un cliente se envía *0x00* al anillo, si se recibe una respuesta el cliente se convierte en MASTER, si no, en SLAVE. Al pulsar *Alt-R* se envía *0x00* al anillo, con la misma lógica anterior.
- MIDI-COUNT-PLAYERS *0x80*: Lo envía el MASTER, seguido de *0x00*. Cada SLAVE

que lo recibe incrementa el número y lo reenvía, de esta forma cuando el MASTER recibe el mensaje de vuelta el que originalmente envió como un byte *0x00* ahora contiene el número de clientes en el anillo. Una vez lo recibe de vuelta lo reenvía, de esta forma los SLAVES también reciben el número total de clientes en el anillo.

- MIDI-RESET-SCORE 0x81: Es un mensaje de un solo byte enviado por el MASTER que reinicia el marcador para todos los jugadores del anillo.
- MIDI-TERMINATE-GAME 0x82: Se trata de una alerta en el MASTER activada al pulsar *Escape*, el MASTER puede elegir entre continuar la partida (MIDI-START-GAME) o acabarla (MIDI-TERMINATE-GAME).
- MIDI-SEND-DATA 0x83: Se envía después de MIDI-START-GAME y contiene todos los datos de la partida:
 - El nombre del jugador, terminado en *0*. Cada jugador lo envía y espera al del resto.
 - Un byte con el tamaño del laberinto.
 - Un byte con la velocidad de recarga.
 - Un byte con la velocidad de regeneración.
 - Un byte con la velocidad de reaparición.
 - Un byte con el número de vidas al revivir.
 - Tres bytes con el número de los distintos drones.
 - 4096 (64x64) bytes con los datos del laberinto.
 - Un byte para indicar si se juega o no en equipos.
 - 16 bytes para indicar el equipo de cada jugador.
 - Un byte para indicar si el fuego amigo esta o no activado.
 - Dos bytes (alto/bajo) para crear la semilla de inicialización del generador aleatorio.
- MIDI-START-GAME 0x84: La envía el MASTER para indicar el comienzo de la partida, seguido de MIDI-SEND-DATA con las opciones seleccionadas para la misma, también se envía para continuar la partida al solicitar un MIDI-TERMINATE-GAME.
- MIDI-ABOUT 0x85: El MASTER envía un byte indicando que se muestre la caja de *ABOUT* del juego a todos los jugadores. El *OK* para cerrar la caja solo se muestra para el MASTER.
- MIDI-NAME-DIALOG 0x86: Lo envía el MASTER para indicar a los jugadores que pueden seleccionar su nombre para la partida, se realiza en cuatro fases:
 - Primero se envía un MIDI-COUNT-PLAYERS.
 - Todos los jugadores en el anillo pueden cambiar su nombre mediante un cuadro

de diálogo.

- Cada jugador espera un byte *0x00* por cada uno del resto de jugadores. El byte *0x00* se manda cuando un jugador pulsa *OK* en el diálogo, por lo que se espera a que todos los jugadores lo hagan para poder continuar.
- Cada jugador envía su nombre con un byte *0x00* al final del mismo, y espera a recibir el nombre del resto de jugadores.
- JOYSTICK-DATA: Una vez dentro de la partida cada jugador envía un byte de datos de su joystick y espera recibir el del resto de jugadores a cada loop del juego, en el bitmask estándar de joystick, el cual es un byte donde los 5 primeros bits representan *left, right, down, up, button*, en dicho orden, donde si el bit es 1 significa que se ha pulsado dicho botón, y si es 0 que no. Durante el bucle del juego solo se reconocen dos tipos de mensaje, JOYSTICK-DATA y MIDI-TERMINATE-GAME.

En el Anexo A se puede encontrar la entrevista realizada a Markus Fritze en su totalidad y traducida al Español.

2.4.2. Estudio del código ensamblador de MIDI Maze

Para realizar el análisis primero se ha estudiado el funcionamiento del procesador Motorola 68000 [8] (ver Figura 2.6). Siendo de especial interes todo aquello relacionado con MIDI y como se comunica, de este estudio se pueden sacar varios puntos que van a ser de gran utilidad en el análisis:

- El microprocesador define 5 dispositivos, los llama internamente *dev*, principales, para los que ofrece puertos de entrada y salida de serie:
 - Dispositivo 0: El puerto paralelo¹³.
 - Dispositivo 1: Puerto *RS-232* (*Recommended Standard 232; Estándar Recomendado 232*)¹⁴.
 - Dispositivo 2: Puerto que sirve simultáneamente como la entrada del teclado, y la salida por pantalla.
 - Dispositivo 3: Puertos MIDI. Controla ambos puertos, tanto MIDI IN como MIDI OUT.
 - Dispositivo 4: puerto *PS/2*¹⁵.

Para cada uno de estos dispositivos se define una tabla de accesos, mediante la cual el microprocesador interactuar con ellos, definiendo de nuevo una lista de operaciones:

- Operación 1: Comprobar si existe un carácter para leer en el dispositivo seleccionado. La operación devuelve -1 cuando hay al menos un carácter listo para ser leído, y un 0 si no.

¹³https://es.wikipedia.org/wiki/Puerto_paralelo

¹⁴<https://es.wikipedia.org/wiki/RS-232>

¹⁵<https://es.wikipedia.org/wiki/PS/2>

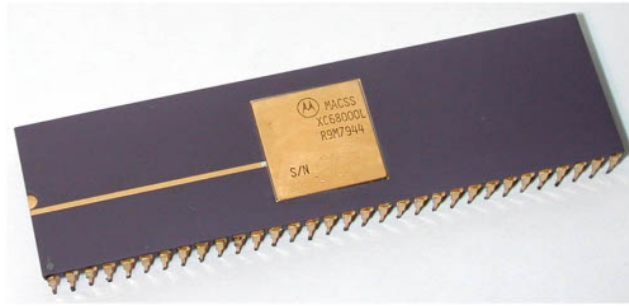


Figura 2.6: Microprocesador Motorola 68000.

- Operación 2: Lee un carácter del dispositivo seleccionado. Devuelve como resultado el carácter leído.
- Operación 3: Escribe un carácter por el dispositivo seleccionado.

Para hacer uso de estas operaciones sobre los dispositivos de Atari ST en código ensamblador hay que seguir unas pautas. En el siguiente ejemplo se va a describir como realizar una comprobación de lectura sobre un dispositivo MIDI, pero el proceso es análogo para el resto de dispositivos y operaciones (ver Figura 2.7).

```

move.w #'A',-(sp)
move.w #0,-(sp)      PRT:
move.w #3,-(sp)      bconout
trap    #13
addq.l  #6,sp

```

Figura 2.7: Ejemplo de como realizar una operación sobre la BIOS(trap 13). En este caso escritura (move.w '3', -(sp)) del carácter 'A' (move.w 'A', -(sp)) por el puerto paralelo (move.w '0', -(sp)). Se suma 6 a sp para devolver la pila a su estado inicial.

En el caso de una comprobación de lectura se mueve el entero que define al dispositivo a la *Stack Pile(SP)* del microprocesador, en el caso del Motorola 68000 el *SP* se encuentra en el registro A7, al querer acceder a MIDI se mueve el entero 3 a *SP*, los enteros en el M68000 se definen como *#\$entero*, después se mueve a *SP* el entero que define la operación a realizar en el caso de la comprobación de lectura un 1, después, el Motorola 68000 ofrece una función para acceder a las funciones de *GEMDOS*¹⁶ llamada *trap*.

Trap define distintos números para acceder a distintas funciones de GEMDOS, en el ejemplo de comprobación de lectura que nos encontramos se hace uso de *trap #13* el cual define el acceso a las funciones de la *BIOS (Basic Input Output System)*, al realizar la llamada a la *BIOS*, se realiza la función definida anteriormente en el *SP*, y se almacena el resultado de la misma en el registro *D0*. En el caso de lectura y escritura el proceso es el mismo, con la diferencia en la escritura que primero se almacena en *SP* el carácter a escribir por el dispositivo, en formato hexadecimal.

Después se ha realizado el análisis del código ensamblador de dos formas paralelamente, primero directamente sobre el código, para esta parte se usa como apoyo el resumen del ensamblador Motorola 68000 de la Universidad de Nápoles Federico II [12] y de la web propiedad de Markey Jester [10], y segundo con la ayuda del *debugger* de *Steem SSE* se ha simulado el funcionamiento de ciertas tramas de ejecución del programa, para ver su comportamiento respecto a lo analizado en el código ensamblador. Como ejemplo vamos a analizar uno de los mensajes usados en el protocolo.

¹⁶*GEMDOS* es el nombre que recibe el sistema operativo usado por los computadores con Motorola 68000,

En el caso del análisis del mensaje *0x00*, el cual hemos definido anteriormente como el mensaje que se envía al inicio y al pulsar *Alt-R* y define el MASTER del anillo, podemos ver como la trama de ejecución en la que se usa tiene las 3 operaciones, comprobación, lectura y escritura, este fragmente se ejecuta al iniciar el juego (ver Figura 2.8).

```

move.w #$3, (a7)      ; 019378: 3EBC 0003 //Lectura por MIDI
move.w #$2, -(a7)     ; 01937C: 3F3C 0002
jsr $341b2            ; 019380: 4EB9 0003 41B2
addq.l #2, a7         ; 019386: 548F
and.l #$ff, d0        ; 019388: C0BC 0000 00FF
move.w #$3, (a7)      ; 01938E: 3EBC 0003 //Comprobación para leer MIDI
move.w #$1, -(a7)     ; 019392: 3F3C 0001
jsr $341b2            ; 019396: 4EB9 0003 41B2
addq.l #2, a7         ; 01939C: 548F
tst.l d0              ; 01939E: 4A80
bne.s -42 {$019378}   ; 0193A0: 66D6
clr.w (a7)            ; 0193A2: 4257
move.w #$3, -(a7)     ; 0193A4: 3F3C 0003 //Escritura por MIDI "0x00"
move.w #$3, -(a7)     ; 0193A8: 3F3C 0003
jsr $341b2            ; 0193AC: 4EB9 0003 41B2
addq.l #4, a7         ; 0193B2: 588F
move.w #$14, (a7)     ; 0193B4: 3EBC 0014
jsr $188f0            ; 0193B8: 4EB9 0001 88F0
move.w d0, $62754     ; 0193BE: 33C0 0006 2754
tst.w $62754          ; 0193C4: 4A79 0006 2754
bne .l +$8 {$0193D4} ; 0193CA: 6600 0008

```

Figura 2.8: Fragmento de ensamblado de MIDI Maze. Una lectura por MIDI, comprobación de lectura por MIDI y escritura por MIDI de “0x00”.

Al analizar el código se puede ver en los tres casos que funciona como se ha definido anteriormente. La función *jsr* recibe una dirección de memoria y mueve el programa a la subrutina indicada en el, en el caso de *jsr \$341b2* se mueve a una subrutina que realiza una llamada a *trap #13*, es decir a la *BIOS*¹⁷.

Podemos observar ya algo que se repetirá en todas las escrituras de MIDI y es que después de realizar la escritura realiza una llamada a la subrutina *\$188f0*, dicha subrutina realiza unas operaciones para reservar la memoria en caso de ser modificada en las operaciones posteriores y acaba llamando a la subrutina *\$341a2*, que realiza una llamada a la *XBIOS* (*extended BIOS*)¹⁸ mediante *trap #14*, con esta llamada conseguimos que en D0 se almacene el MIDI IN del dispositivo MIDI, con lo que se confirma la característica del protocolo definido anteriormente que definía que todo mensaje enviado por el MASTER debe acabar volviendo a él.

Posteriormente se realiza un *tst* para comparar el carácter que se le da con *#\$0*, en caso de ser iguales se convierte en MASTER. Para analizar mejor el comportamiento del programa ejecutamos esta traza en el *debugger*, ya que aunque no podemos crear una conexión MIDI entre emuladores, podemos modificar los registros con lo que queremos que llegue, de esta forma repetimos la traza y vemos que todo se confirma, además que si al acceder a la *XBIOS* en *D0* hay un 0 se vuelve MASTER y si hay algo distinto se vuelve un SLAVE. Esto también confirma el uso de *0x00* del protocolo descrito anteriormente.

Como conclusión de este análisis se ve que el protocolo descrito al principio de la sección es correcto, por lo que lo podemos usar de guía para el proyecto. Además de que, al ser correcto y estar comprobado, podemos concluir que efectivamente MIDI Maze usa el dispositivo MIDI para enviar mensajes *raw* no validos para el protocolo, y dado que esto esta permitido en la

¹⁷<https://es.wikipedia.org/wiki/BIOS>

¹⁸La *XBIOS* ofrece funciones de soporte para hardware específico de Atari ST y funciona exactamente igual que la *BIOS*

arquitectura de Motorola 68000 es posible crear una comunicación como la descrita, algo que como se ha indicado en la Sección 2.3 no está permitido en Windows, lo cual hace imposible usar el protocolo de MIDI Maze para la comunicación. Además, analizar el código nos ha ayudado a comprender mejor el flujo de ejecución de MIDI Maze, lo que será útil en la creación del clon para PC.

En el Anexo B se encuentra un análisis en profundidad de los fragmentos en ensamblador más importantes.

Capítulo 3

Desarrollo de un clon de MIDI Maze para PC

Se va a usar como punto de partida del clon de MIDI Maze para PC el videojuego obtenido como resultado del trabajo realizado en la asignatura optativa Videojuegos, descrito en la Sección 1.2.

En este capítulo se van a presentar los cambios realizados al clon de Windows de MIDI Maze para poder comunicarse en una red MIDI y jugar multijugador. Primero se expondrán las decisiones tomadas respecto a los resultados obtenidos hasta el momento, y como se ha adaptado la idea final respecto a ellos. Después se desarrollaran de forma resumida lo añadido al programa para hacer posible el objetivo propuesto, los problemas encontrados y las soluciones adoptadas para alcanzarlos.

Dados los resultados obtenidos en las pruebas (ver Capítulo 2), se ha podido demostrar que no va a ser posible realizar una conexión entre el clon en Windows de MIDI Maze y el MIDI Maze de Atari ST, por ello se toma la decisión de mantener la idea de jugar en multijugador por MIDI, pero entre varios ordenadores ejecutando el clon de MIDI Maze para Windows usando la red MIDI (ver Figura 3.1).

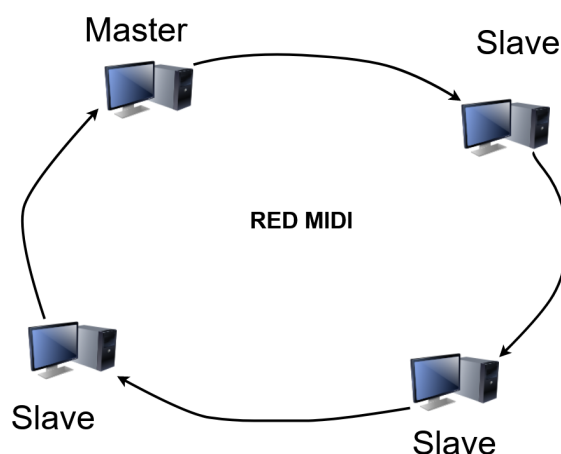


Figura 3.1: Funcionamiento de la red MIDI a diseñar entre clones de MIDI Maze en PC.

3.1. Diseño de la comunicación MIDI del clon de MIDI Maze para PC

Como punto de partida es necesario poder enviar y recibir mensajes MIDI en el clon, desarrollado en C++. Para ello se hará uso de la biblioteca *JUCE* para C++, la cual permite la transmisión e interpretación de mensajes MIDI, al más bajo nivel posible. Al comenzar las pruebas con la biblioteca, surge un problema, los driver de Windows de MIDI solo permiten que los puertos MIDI IN reciban mensajes cuando están abiertos, esto pasa dado que, a diferencia de Atari ST, el cual solo tiene un dispositivo MIDI IN y MIDI OUT de serie, los drivers de Windows están pensados para servir a varios dispositivos a la vez, por lo que mientras que en Atari ST teníamos la opción de comprobar si había un carácter y entonces leerlo, lo que permitía enviar un mensaje y que otro Atari ST lo leyera cuando quisiera, esta opción no es posible en Windows por lo que habrá que introducir sistemas de sincronización de los mensajes.

También es necesario modificar los mensajes MIDI usados en el protocolo, ya que no se pueden enviar ni recibir en Windows, así que se modifican de la siguiente forma, teniendo en cuenta que su uso es el mismo que en el original, para así respetar todo lo posible el protocolo original descrito por MIDI Maze:

- Como adaptación del byte *0x00* usado por MIDI Maze, en el clon se usa el mensajes *0x80 00 00*.
- MIDI-COUNT-PLAYERS *0x80*, se adapta usando *0x90 00 00*. Dado que necesitamos enviar un número entero que va a ir incrementando hasta un máximo de 16 por cada jugador no podemos hacer uso de un mensaje MIDI estructurado, ya que, por ejemplo, al enviarlos como *raw bytes 0x90 00 00* sera enviado como *10010000 00000000 00000000*, por lo que parecería lógico ir aumentando uno de los tres bytes, pero esto no es posible, ya que MIDI adapta lo mensajes a su protocolo lo cual hace que, por ejemplo *00000010* no sea *0x2*, si no *0xA*. Así que necesitamos una alternativa, por lo que optamos por el uso de los mensajes *MIDI SysEx(System Exclusive)* los cuales, dentro de una especificación, permiten definir tus propios mensajes dentro del protocolo. Para formar un mensaje *SysEx* hay que seguir la siguiente definición *0xF0 data F7*, dentro de *data* se puede enviar el número de bytes que quieras. Por lo que haciendo uso de *SysEx*, donde en el protocolo original se enviaba un *0x00* en nuestro protocolo se envía *0xF0 00 F7*, donde a cada jugador que lo recibe incrementa el byte de data y lo envía al siguiente, hasta que vuelve al MASTER, el cual envía un nuevo *SysEx* con el número total de jugadores a todos los jugadores del anillo.
- MIDI-TERMINATE-GAME *0x82*, se adapta enviando un mensajes *0x91 00 00*.
- MIDI-SEND-DATA *0x83*, en este caso se ha tenido en cuenta que *0x83* en el protocolo original siempre va después de MIDI-START-GAME, y que contiene los datos de la partida. Así que teniendo esto en cuenta MIDI-SEND-DATA y todos los datos de la partida se estructuran dentro de un *SysEx*, ya que ciertos datos son 1 o 0, así que se podrían codificar fácilmente en un mensajes MIDI del protocolo, mientras que otros, como el número de drones, de vidas o el tamaño del laberinto, son números enteros, así que se toma la decisión de formar un mensaje *SysEx* con todos los datos de la partida, excluyendo los datos del laberinto. Para enviar el laberinto se toma la decisión

de coger como base el mensaje MIDI valido *0x80 00 00* y modificarlo de la siguiente forma, codificar al final de cada byte si en el laberinto hay o no un muro, enviando así en cada mensaje la información de 3 posiciones del laberinto, esto es, por ejemplo, *0x81 00 01* esto significa que en la primera posición hay un muro, en la segunda no, y en la tercera si, y así sucesivamente hasta enviar todo el laberinto.

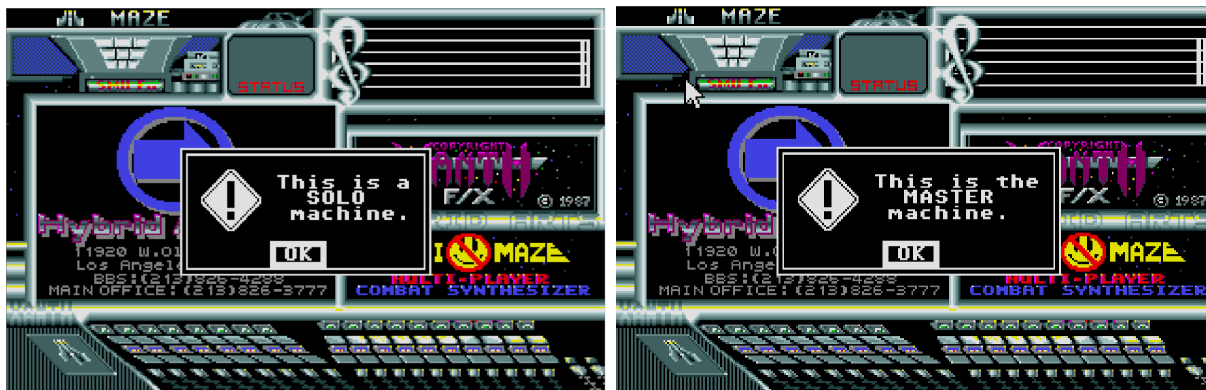
- MIDI-START-GAME *0x84*, se adapta con *0x80 00 01*. Una vez un jugador recibe este mensaje espera recibir MIDI-SEND-DATA.
- JOYSTICK-DATA, para adaptar este mensaje se mantiene la idea del uso de un *bitmask* como en el protocolo de MIDI Maze, así que se parte del mensaje *0x80 00 00*, en binario *10000000 00000000 00000000*, de esta forma es mucho más sencillo ver la adaptación, ya que la idea es basarse en el concepto de *bitmask* y usar el mensaje en binario para codificar los movimientos de la siguiente forma. Si se pulsa *up* el bit 16 se pone a 1, si se pulsa *down* el bit 20 se pone a 1, si se pulsa *fire* el bit 24 se pone a 1, si se pulsa *right* el bit 12 se pone a 1, y si se pulsa *left* el bit 8 se pone a 1.
- Como última adaptación, teniendo en cuenta la limitación de como funciona en Windows los mensajes MIDI que hemos comentado al principio del capítulo, hay que incorporar algún sistema de sincronización, para ello se hace uso de distintos *flags* a lo largo del código, que controlan que mensajes se pueden recibir en cada momento y cuales no, así como cuando es necesario bloquear un jugador hasta que reciba cierto mensaje. Como parte de la sincronización también se aplica la idea del protocolo original, de cuando se envía un mensaje se espera que el mensaje vuelva a llegar al MASTER. Combinando estos dos factores conseguimos solucionar el problema de sincronización que había surgido.
- También se ha realizado otro ajuste respecto a MIDI Maze en Atari ST, y es que en MIDI-SEND-DATA MIDI Maze envía en el mensaje dos bytes que inicializan una semilla para la generación aleatoria, y puesto que no se he podido averiguar como Atari ST genera números aleatorios, por lo que se toma la decisión de que cada vez que un jugador necesite seleccionar una posición aleatoria en el laberinto que se comunicara al resto de jugadores, además dicha posición va a estar limitada a números enteros, ya que no es posible enviar números reales mediante mensajes MIDI. Dicha posición se envía en un mensaje *SysEx*.
- Como último ajuste, si nos fijamos en el protocolo usado en MIDI Maze durante la partida solo se envía el joystick data, por lo que los drones son controlador individualmente por cada instancia de MIDI Maze. Esto generaba muchos problemas de sincronización en el clon de Windows, por lo que se delega el control de todos los drones, o lo que es lo mismo toda la inteligencia artificial, al MASTER el cual enviara las acciones realizadas por los drones en mensajes JOYSTICK-DATA al resto de jugadores.

3.2. Comunicaciones MIDI en el clon MIDI Maze para PC

Al clon de MIDI Maze realizado como parte de las práctica de la asignatura optativa Videojuegos se le han realizado añadidos, mayormente características que van a ser necesarias

para el multijugador MIDI:

- Pantallas de *SOLO MODE*, *MASTER MODE* y *SLAVE MODE* las cuales muestran el estado del jugador respecto a la red MIDI (ver Figura 3.2).
- La opción de mostrar el *About* del juego.



(a) Estado *SOLO MACHINE*.

(b) Estado *MASTER MACHINE*.



(c) Estado *SLAVE MACHINE*.

Figura 3.2: Pantallas de estado del jugador.

A continuación se describen los pasos realizados realizados en esta implementación:

1. Incluir la librería *JUCE* en nuestro proyecto. Al usarla para enviar mensajes MIDI es muy simple, basta con formar un mensajes válido y enviarlo al dispositivo MIDI seleccionado. Pero a la hora de recibir se debe definir una clase que se ocupe de abrir el puerto MIDI seleccionado y la lógica de recepción de los mensajes, esto se debe a que como se ha dicho ya al abrir un puerto este funciona como un monitor de los mensajes hasta que se cierra, por ello se debe implementar una función a la cual lleguen los mensajes MIDI que entran por el puerto abierto anteriormente y la cual contiene la lógica con la que se van a tratar dichos mensajes.
2. Se empieza a crear la lógica de envío de los mensajes de el protocolo que hemos definido para nuestro clon. Al inicio del programa se le muestra al usuario los puertos MIDI disponibles y se le pide que seleccione uno de entrada y otro de salida, haciendo que formen un anillo. Al inicio del programa este empieza por defecto en *SOLO MODE* esperando que se pulse *OK* para continuar, ahora al pulsar *Alt-R* se envía al MIDI OUT *0x80 00 00* lo cual indica la petición de ser MASTER y se espera a recibir respuesta, esto llega al resto de jugadores que reenvían el mensaje y se ponen en modo SLAVE, al recibir el mensaje de nuevo el jugador se pone en modo MASTER lo que inicia el anillo. Nada más comenzar la conexión el MASTER envía la petición de MIDI-COUNT-PLAYERS así cada jugador en el anillo sabe el número total de jugadores, y el número de SLAVE que es, siendo el MASTER el número 0, el SLAVE conectado a su MIDI OUT el 1, y así sucesivamente, esto sera de utilidad más adelante durante el

bucle principal del juego.

3. Ahora es momento de seleccionar las opciones de la partida, por lo que el MASTER selecciona un laberinto, y las opciones, recordar que el máximo de jugadores es 16 por lo que el número de jugadores más el de drones debe ser menor de 16. Al pulsar *PLAY* se envía el mensaje de MIDI-START-GAME, cuando un SLAVE recibe el mensaje lo reenvía y se queda a la espera del mensaje con las opciones de la partida, cuando el MASTER recibe el mensaje de vuelta forma un mensaje *SysEx* con las opciones de la partida, a falta de los datos del laberinto, y lo envía. Cuando lo recibe de vuelta comienza el envío del laberinto, el cual como se ha indicado se realiza enviando varios mensajes MIDI hasta que se envía todo el laberinto, cuando el último jugador del anillo lo recibe envía un mensaje *0x91 00 00* al MASTER para que comience la partida.

Antes de comenzar la partida, el MASTER genera aleatoriamente la posición inicial de todos los jugadores y drones de la partida y la envía mediante un mensajes *SysEx*, uno por cada jugador y dron, y espera a recibirlos de vuelta. Ahora todos los jugadores del anillo tienen los datos necesarios para iniciar una partida multijugador.

4. Durante el loop principal del juego, el MASTER espera a recibir la acción realizada por todo el resto de jugadores, una vez recibido envía un mensaje JOYSTICK-DATA por cada jugador en el anillo y cada dron. Localmente cada jugador actualiza la posición y orientación en el laberinto del resto de jugadores y drones, así como si ha disparado o no. De esta forma conseguimos que la partida multijugador se mantenga en sincronización, el inconveniente es que la partida se ve ralentizada a la velocidad del anillo.
5. Respecto a los cambios independientes del envío de mensajes, también hay que modificar como el programa calcula los movimientos de los drones. Primero el que respecta al resto de jugadores en el anillo. Para entender mejor la siguiente explicación, hay que tener en cuenta como funciona en el clon originalmente el control de los drones, ya que en una primera instancia el clon no tenía la opción de jugar en multijugador por lo que funciona con una *IA* que determina sus movimientos respecto de estados. Por ello al introducir el concepto de multijugador se parte de la misma lógica con la que se trata a los drones, por lo que cada jugador internamente considera al resto como drones, y los trata de la misma forma, con una característica especial imprescindible, que es que si se trata de otro jugador los movimientos no se controlan mediante una *IA* si no que se espera recibir un mensaje en cada bucle que informe de los movimientos que ha realiza ese jugador.

Por esta característica era importante que al recibir MIDI-COUNT-PLAYERS cada jugador supiese que número de jugador es dentro del anillo y el total de jugadores que hay en el mismo, ya que de esta manera al controlar en cada bucle los movimientos del resto de jugadores y drones, cada jugador sabe, en el caso del MASTER, a que otro jugador pertenece los movimientos que ha recibido, y en el caso de los SLAVES a que otro jugador o dron pertenecen. Para ello se modifica la lógica de drones originalmente, los cuales son controlados por una *IA* y se añade el concepto de jugadores y drones.

6. En el MASTER la lógica respecto a los drones no se modifica, pero se almacena en un mensaje JOYSTCIK-DATA que acción han realizado en este bucle. Una vez el MASTER a calculado los movimientos de los drones y recibe todos los movimientos de los SLAVES, envía por MIDI el JOYSTICK-DATA referente al MASTER, los SLAVES

y los drones, a todos los SLAVES.

Por parte de los SLAVES a cada bucle envían su JOYSTICK-DATA por MIDI y al recibir por parte del MASTER todos los JOYSTICK-DATA los actualiza localmente lo ocurrido en el bucle con la información recibida del MASTER.

Durante el bucle del juego, por parte del MASTER, cada vez que muere un jugador o un dron se calcula una nueva posición aleatoria la cual se envía en un mensaje *SysEx* al resto de jugadores del anillo.

Capítulo 4

Conclusiones

El objetivo de este proyecto era intentar que el videojuego MIDI Maze (1987), desarrollado para el ordenador Atari ST (1985–1993), y que permite un modo multijugador a través de puertos MIDI, consiga comunicarse con un PC ejecutando un clon de MIDI Maze para Windows en una red MIDI. Para ello se ha usado ingeniería inversa sobre MIDI Maze para estudiar su protocolo MIDI.

El primer paso lógico era estudiar MIDI, ya que se aleja de los conocimientos del grado e iba a ser la base de todo el proyecto, así que se necesitaba este conocimiento previo para poder entender los resultados obtenidos en las siguientes pruebas y en la integración final. Durante esta fase se han conseguido todos los conocimientos necesarios sobre el protocolo y su funcionamiento para el desarrollo del proyecto.

A lo largo de la fase de pruebas se realizan múltiples ensayos, enfocando los problemas que surgen desde distintos ángulos e ideando soluciones para ellos. En esta fase también se recopila mucha información de distintas fuentes, que ayudan a contrastar los resultados conseguidos durante el proyecto.

Dados los resultados que nos aportan las pruebas, decidimos analizar directamente el ensamblador de MIDI Maze, lo cual requiere también realizar un estudio del microprocesador *Motorola 68000* para poder analizar correctamente el código ensamblador. Los resultados que obtenemos de este análisis coinciden con los obtenidos anteriormente. Con toda la información obtenida llegamos a demostrar que no es posible realizar la conexión entre MIDI Maze y el clon en Windows, ya que los drivers del propio sistema operativo no lo hace posible. Por lo que, esta conexión PC con MIDI Maze Atari ST no se puede realizar con los medios de los que disponemos en este proyecto, como ya se ha explicado.

Por último, se ha creado un clon de MIDI Maze para Windows que permita jugar en multijugador en una red MIDI de hasta 16 jugadores. Durante esta fase se ha propuesto un protocolo MIDI adaptado para permitir esta conexión, así como diferentes ampliaciones sobre el clon resultante de la asignatura optativa Videojuegos, el cual se ha usado como punto de partida.

En resumen, se han alcanzado los objetivos del proyecto propuestos. Se ha realizado un estudio correcto de MIDI, las pruebas han sido útiles para identificar los problemas y encontrar soluciones y el desarrollo del clon de PC ha permitido crear un multijugador en una red MIDI.

4.1. Cronograma

A continuación se muestra la evolución temporal de las tareas generales que se han realizado en este proyecto (ver Figura 4.1).



Figura 4.1: Diagrama de Gantt con la evolución temporal del desarrollo del proyecto.

- Estudio de MIDI: 2 semanas. Esta fase se divide en dos a lo largo del proyecto, 1 primera semana de estudio del estándar MIDI para sentar las bases del proyecto, y durante el mismo ha sido necesario volver varias veces a investigar distintos aspectos del estándar para dar solución a los problemas encontrados durante el desarrollo.
- Realización de pruebas sobre sobre MIDI: 6 semanas. Se trata de la fase más costosa en tiempo del proyecto, debido a los numerosos problemas encontrados resultado de las pruebas, lo que hacia necesario idear soluciones y consultar numerosas fuentes, sobre todo por el hecho de que existe muy poca información sobre MIDI en MIDI Maze, lo cual ha complicado el proceso.
- Análisis de MIDI Maze de Atari ST: 4 semanas. En esta fase la mayor inversión de tiempo es en el análisis del código en ensamblador. Para ello, primero hay que estudiar la arquitectura del microprocesador M68000 para poder analizar el código ensamblador, se busca documentación y se estudian las características más importantes de esta arquitectura, documentación a la que tendremos que volver en numerosas ocasiones durante el análisis del código ensamblador.
- Trabajo sobre el clon de PC de MIDI Maze: 4 semanas. En esta fase se ha rediseñado el protocolo para que sea posible realizar la conexión, además de ser necesario realizar numerosos cambios sobre el código del clon, siendo los cambios relacionados con la comunicación MIDI los más costosos de realizar, al ser un concepto nuevo. Además se han añadido algunas ampliaciones respecto del clon entregado como resultado de la asignatura de videojuegos para que se asemeje más al videojuego original.
- Realización de la memoria: 2 semana. Incluyendo plasmar los resultados del proyecto, revisiones y correcciones de la misma.

4.2. Posibles ampliaciones

En esta sección vamos a enumerar algunas posibles ampliaciones que se podrían aplicar al resultado final del proyecto, así como posibles soluciones a algunos de los problemas encontrados pero que se salían del alcance del proyecto. Primero vamos a hablar de algunas

posibles soluciones para conseguir comunicar correctamente MIDI Maze con PC:

- Se podría intentar hacer uso de otro sistema operativo para realizar la comunicación. En parte se ha realizado ya una pequeña investigación y según la información encontrada los drivers MIDI de *Apple* también definen una lista de mensajes predeterminados y mediante el resultado de unas sencillas pruebas, *Ubuntu* parece que también lo hace. Habría que realizar una investigación más exhaustiva con distintos sistemas operativos.

Hay que tener en cuenta que si algún sistema operativo hace posible esta conexión, el clon de MIDI Maze para debe ser adaptado a dicho sistema operativo, ya que su diseño actual solo permite su funcionamiento sobre Windows.

- Usar sistemas externos programables como puente de conexión entre PC y MIDI Maze en Atari ST. Por ejemplo, usar un *Raspberry Pi*, que funcione de intermediario entre el Atari ST y el PC, ya que permiten ser programados.

Teniendo en cuenta el resultado final del proyecto, tenemos un clon de MIDI Maze en Windows, capaz de jugar a la vez con 16 jugadores mediante el uso de mensajes MIDI, todo esto entre la misma versión del clon. Por ello podemos proponer las siguientes ampliaciones:

- Permitir jugar a más de 16 jugadores simultáneos.
- Adaptar el código del juego para que sea posible jugar entre distintos sistemas operativos.
- Permitir seleccionar las opciones avanzadas del clon para ser usadas en multijugador.

Hay que tener en cuenta que estas ampliaciones son desde el punto de vista del resultado final, en el que no existe la conexión con MIDI Maze Atari ST, ya que esta ampliaciones no serían posibles, salvo la de adaptar el juego a otros sistemas operativos, dado que MIDI Maze de Atari ST no puede ser modificado.

4.3. Opinión personal

He conseguido alcanzar los objetivos propuestos del proyecto además de otros objetivos intermedios que han surgido a partir de los iniciales, por ello estoy satisfecho con el trabajo realizado.

Es la primera vez que trabajo con MIDI, por lo que se trata de una tecnología nueva que he tenido que investigar, lo cual ha resultado muy interesante y beneficioso, algo que se puede aplicar a muchos aspectos del proyecto sobre tecnologías o aplicaciones nuevas de las que no conocía su existencia anteriormente.

Destacar que la información disponible sobre MIDI Maze es poca y de su uso de MIDI mucho menos, por lo que encontrar dicha información y en varios casos generarla yo mismo a lo largo del proyecto (analizando distintos aspectos del juego, consultando expertos, etc... En conclusión combinando distintas fuentes de información) ha resultado uno de los aspectos más desafiantes del proyecto a nivel personal, aunque también uno de los más enriquecedores.

Este proyecto ha tenido bastantes altibajos, ya que me he encontrado con bastantes dificul-

tades durante la realización. La mayor de todas ha sido no poder conseguir la conexión entre MIDI Maze Atari ST y el clon de PC, ya que aunque el objetivo era intentar la conexión y he conseguido demostrar que no es posible en el contexto del proyecto, durante la realización de las pruebas ha sido algo abrumador no conseguirlo ya que mientras las hacia la sensación era de chocar contra un muro y no avanzar; pero al mirar hacia atrás y analizar todo lo realizado me di cuenta de que dicho trabajo si era útil y no era en vano, lo que me animo a terminar el proyecto de la mejor forma posible.

Bibliografía

- [1] *AtariForum*. Consulta realizada en Atari Forum por parte de el autor del proyecto, Jesús Ángel González con nombre de usuario “Hagane03” en el foro, de la cual se extrae información útil para el proyecto. <https://atari-forum.com/viewtopic.php?t=41808> (accedido por última vez: 24.06.2022).
- [2] *Drivers de MIDI de Windows*. Mensajes de dispositivo de audio para MIDI usados por WINMM de Windows <https://learn.microsoft.com/es-es/windows-hardware/drivers/audio/audio-device-messages-for-midi> (accedido por última vez: 30.10.2022).
- [3] *Entrevista a Markus Fritze*. Entrevista realizada a Markus Fritze por FujiNet <https://github-wiki-see.page/m/FujiNetWIFI/fujinet-platformio/wiki/MIDI-Maze-Notes>. 2020.
- [4] *Especificación de MIDI 1.0*. Resumen de la especificación que define el protocolo MIDI 1.0. <https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message> (accedido por última vez: 03.11.2022).
- [5] *Foro MIDI-OX*. Consulta realizada en el foro de MIDI-OX contestada por uno de los responsables de MIDI-OX <http://www.midiox.com/cgi-bin/yabb/YaBB.pl?board=query;action=print;num=1135137114>. Consulta similar en el mismo foro contestada por un usuario <http://www.midiox.com/cgi-bin/yabb/YaBB.pl?board=wishlist;action=display;num=1084974822> (accedido por última vez: 01.08.2022).
- [6] *función modMessage*. Función de entrada de las peticiones de los dispositivos MIDI ofrecida por el DDL WINM <https://learn.microsoft.com/es-es/windows-hardware/drivers/audio/mod-message> (accedido por última vez: 30.10.2022).
- [7] Steve de Furia y Joe Scacciaferro. *The MIDI Book. Using MIDI and Related Interfaces*. 1.^a ed. MUSIC SALES LTD, 1986.
- [8] K. Gerits, L. Englisch y R. Bruckmann. *Atari ST Internals. The authoritative insider's guide*. 3.^a ed. Abacus Software, 1986.
- [9] Jesús Ángel González, Sergio Cartiel y Santiago Jiménez. *Memoria clon MIDI Maze PC*. Inf. téc. Grupo 1 <http://webdiis.unizar.es/asignaturas/videojuegos/>. Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza, 2021/2022.
- [10] *MarkeyJester*. Documentación sobre la arquitectura de Motorola 68000. https://mrjester.hapisan.com/04_MC68/Index.html (accedido por última vez: 04.08.2022).
- [11] *MIDI Association (MMA)*. La MIDI Association es una asociación sin animo de lucro que desarrolla las especificaciones MIDI. <https://midi.org/> (accedido por última vez: 03.11.2022).
- [12] *Roberto Canonico*. Documentación sobre la arquitectura de Motorola 68000 realizada para la Universidad de Nápoles Federico II. <http://wpage.unina.it/rcanonic/didattica/ce1/docs/68000.pdf>. 2011.
- [13] Julian Storer. *JUCE: An open-source cross-platform C++ application framework*. Raw Material Software Limited. 2004.

Anexo A

Entrevista a Markus Fritze traducida

En este anexo se incluye una entrevista traducida realizada por la plataforma *FujiNet* a Markus Fritze [3] el día 5 de octubre de 2020 a través de Twitter en referencia a MIDI Maze II.

A.1. Preguntas y respuestas

FujiNet: *¿Existe algún documento de referencia del protocolo que usa MIDI Maze*

Markus Fritze: *El protocolo es muy simple: transfiere los datos del laberinto al comienzo y después solo los datos del joystick para un máximo de 16 jugadores. MIDI Maze II también comunica el teclado. Pero esto significa que los clientes tienen que estar sincronizados perfectamente, porque todos reciben los datos del joystick de hasta 16 jugadores y tienen que realizar cálculos para todos ellos.*

FN: *¿Sabes si el protocolo de MIDI Maze original es muy distinto de este (MIDI Maze II)?*

MF: *Es prácticamente el mismo.*

FN: *¿Los ordenadores mandan sus mensajes MIDI cuando se produce una actualización (por ejemplo, cuando se pulsa el botón del joystick) o lo hacen en intervalos de tiempo ?*

MF: *No, los envían (mensajes MIDI) a cada bucle. Si un jugador no pulsa nada en un bucle, se envía 0 en el mensaje.*

FN: *¿Existe algún intervalo definido para cada bucle de ejecución?¿Como se manejan las colisiones?*

MF: *No, esto es algo raro. Es un anillo MIDI, así que esto no es un problema: Para 5 jugadores, 5 bytes se envían en circulo(anillo MIDI). Cada ordenador recibe 4 bytes del resto de jugadores y envía el suyo propio. De esta forma se garantiza que están en orden y sincronizados. Problema: se ralentizan los FPS(Fotogramas por Segundo) a la velocidad del anillo, que es la velocidad que les cuesta a todos los jugadores calcular los movimientos y mostrarlos en pantalla*

A.2. Documentación del protocolo MIDI usado por MIDI Maze Atari ST

Markus Fritze también comparte el protocolo MIDI que usa MIDI Maze II, que según indica es casi idéntico al de MIDI Maze original:

- Joystick Data: Durante la partida cada jugador envía sus datos de joystick al anillo y recibe los datos de los demás jugadores. Cada dato de joystick es un byte en el bitmask estándar de joystick (izquierda, derecha, arriba, abajo y botón para los 5 primeros bits). Solo MIDI-TERMINATE-GAME se reconoce durante la partida.
- Comunicación MIDI: Cuando se lanza un cliente se envía *0x00* al anillo. Si hay respuesta el anillo se completa y el ordenador se convierte en MASTER. Si no hay respuesta, se convierte en SLAVE por defecto. Esto significa, que el ordenador que cierra el anillo es el MASTER.

El primer byte distinto de cero que reciba un SLAVE activa el inicio de partida. Si un SLAVE recibe un comando, se reenvía al resto de SLAVES (y en algún punto vuelve al MASTER).

- MIDI-COUNT-PLAYERS *0x80*: Lo envía el MASTER, seguido de *0x00*. Cada SLAVE que lo recibe incrementa el número y lo reenvía. Cuando el MASTER lo recibe de vuelta sabe cuántas máquinas hay en el anillo. Después el MASTER vuelve a enviar el número de máquinas, para que así los SLAVES lo tengan también.
- MIDI-RESET-SCORE *0x81*: Es un mensaje de un solo byte enviado por el MASTER que reinicia el marcador para todos los jugadores del anillo.
- MIDI-TERMINATE-GAME *0x82*: Se activa como un comando de joystick. Esto lanza una alerta en el MASTER y para la partida. Entonces el MASTER puede seleccionar continuar la partida (MIDI-START-GAME) o acabarla (MIDI-TERMINATE-GAME).
- MIDI-SEND-DATA *0x83*: Se envía después de MIDI-START-GAME y contiene los datos compartidos de la partida:
 - El nombre del jugador, terminado en *0*. Cada jugador lo envía y espera al resto.
 - Un byte con el tamaño del laberinto.
 - Un byte con la velocidad de recarga.
 - Un byte con la velocidad de regeneración.
 - Un byte con la velocidad de reaparición.
 - Un byte con el número de vidas al revivir.
 - Tres bytes con el número de los distintos drones.
 - 4096 (64x64) bytes con los datos del laberinto.

- Un byte para indicar si se juega o no en equipos.
- 16 bytes para indicar el equipo de cada jugador.
- Un byte para indicar si el fuego amigo esta o no activado.
- Dos bytes (alto/bajo) para crear la semilla de inicialización del generador aleatorio.

Como se puede ver, esto podría optimizarse (aumentado la velocidad de las transmisión en el anillo) y potencialmente tener *checksum*¹.

Todos los equipos se podrían almacenar en un solo byte (cada uno es de 1 o 2 bits). El equipo del jugador se podría comprimir en 4 bytes (4 equipos = 2 bits por jugador), el número de drones en 2 bytes (4 bits por tipo). Como ventaja de la no optimización el MASTER puede cambiar otros parámetros, por ejemplo el parámetro de tiempo, y los SLAVES recibirían dicho cambio.

Y por último, y además el más importante, los datos del laberinto: solo hace falta transmitir los muros, los cuales solo se pueden colocar en coordenadas determinadas y además un bit por muro es suficiente. Para un laberinto de 14x14, sería $(14-2)/2 = 6$ bits horizontales al cuadrado (dados los muros verticales)= 36 bits o 5 bytes, en lugar de 4096. Incluso para un muro de 62x62, serían solamente 113 bytes. No creo que más compresión RLE (Run-length encoding)² sea significativa, porque la velocidad de MIDI es suficientemente alta - incluso sin ninguna optimización apenas cuesta 2 segundos transferir todos los datos. Pero creo que enviar menos datos haria más estable el comienzo de la partida.

De cualquier forma, MIDI Maze no realiza compresión alguna.

- MIDI-START-GAME *0x84*: Comienza la partida (seguido de MIDI-SEND-DATA), también se envía para continuar la partida al solicitar un MIDI-TERMINATE-GAME.
- MIDI-ABOUT *0x85*: El MASTER envía un byte indicando que se muestre la caja de *ABOUT* del juego a todos los jugadores. El *OK* para cerrar la caja solo se muestra para el MASTER.
- MIDI-NAME-DIALOG *0x86*: Lo envía el MASTER para indicar a los jugadores que pueden seleccionar su nombre para la partida, se realiza en cuatro fases:
 - Como en MIDI-COUNT-PLAYERS se cuenta el número de jugadores.
 - Todos los jugadores en el anillo pueden cambiar su nombre mediante un cuadro de diálogo.
 - Cada jugador espera un byte *0x00* por cada uno del resto de jugadores. El byte *0x00* se manda cuando un jugador pulsa *OK* en el diálogo, por lo que se espera a que todos los jugadores lo hagan para poder continuar.
 - Cada jugador envía su nombre con un byte *0x00* al final del mismo, y espera a recibir el nombre del resto de jugadores. Dada la naturaleza del anillo, el último

¹<https://en.wikipedia.org/wiki/Checksum>

²https://es.wikipedia.org/wiki/Run-length_encoding

nombre recibido es el del propio jugador de nuevo. Esto completa la transmisión de nombres.

- JOYSTICK-DATA: Una vez dentro de la partida cada jugador envía un byte de datos de su joystick y espera recibir el del resto de jugadores a cada loop del juego, en el bitmask estándar de joystick, el cual es un byte donde los 5 primeros bits representan *left, right, down, up, button*, donde si el bit es 1 significa que se ha pulsado dicho botón, y si es 0 que no. Durante el bucle del juego solo se reconoce MIDI-TERMINATE-GAME.

Anexo B

Análisis del código ensamblador M68000 del MIDI Maze de Atari ST

En este anexo se va a incluir el análisis de los puntos más importantes resultado del trabajo realizado en el punto 2.4.2. Como guía de lo que se va a ver en los fragmentos del código de izquierda a derecha en cada línea del código, primero la instrucción desensamblada, seguido de “;”, la dirección de memoria de la instrucción, “:”, y el hexadecimal de la instrucción, en algunos casos se añade al final “//” seguido de un comentario para aclarar ciertos fragmentos. A tener en cuenta, la subrutina *341b2* llama a la BIOS, y la *188f0* a la XBIOS.

En el caso de la descripción de las imágenes primero se indica la dirección de memoria de la que se habla y después se describe el fragmento:

- 018904: Comprobación de lectura, 0 = no hay nada en el input, 1 si que lo hay. Comprueba el valor, realiza TST sobre d0 (donde se almacena el resultado) lo que hace que lo comprueba con 0, si son iguales, d0 = 0, salta a la dirección \$18934 (no hay nada que leer) y continua la ejecución. Si no, se prepara para leer por MIDI (*01891A). [\$18934] → Llama a la subrutina \$188cc, y resta \$37e0c a D0, para luego sumarle #\$8000. Si el resultado de la suma activa los flags correspondientes, se produce un bne a \$01894E (salto incondicional a \$018904, esta misma comprobación). Si no, mueve #-1 a D0 y salta a \$018950 (unlk y rts) (ver Figura B.1).
- 01891A: Lectura de MIDI, lee de MIDI y lo almacena en d0, realiza una and de d0 con #ff, y salta a la dirección \$18950 de forma incondicional, lo cual hace un unlk a6, y realiza un rts que lleva a la dirección \$193BE.

```
move.w #$3,(a7)      ; 018904: 3EBC 0003 //Comprobaciones en MIDI para leer
move.w #$1,-(a7)     ; 018908: 3F3C 0001
jsr $341b2           ; 01890C: 4EB9 0003 41B2
addq.l #2,a7         ; 018912: 548F
tst.l d0             ; 018914: 4A80
beq .l +$1c {$018934} ; 018916: 6700 001C
move.w #$3,(a7)      ; 01891A: 3EBC 0003 //Lectura en MIDI
move.w #$2,-(a7)     ; 01891E: 3F3C 0002
jsr $341b2           ; 018922: 4EB9 0003 41B2
addq.l #2,a7         ; 018928: 548F
and.l #$ff,d0        ; 01892A: C0BC 0000 00FF
bra .l +$1e {$018950} ; 018930: 6000 001E
```

Figura B.1: Fragmento descrito para la dirección 018904

- 019378: Lectura de MIDI, lee de MIDI y lo almacena en D0, realiza una and de D0

con #ff (ver Figura B.2).

- 01938E: Comprobación para leer, si hay algo disponible para leer, D0=-1, se cumple del bne y salta a \$019378 (lectura anterior), si no lo hay, D0=0, continua con la ejecución en la escritura \$0193A4
- 0193A4: Escribe en MIDI, realiza un clear, por lo que en A7 hay un 0 (mensaje de petición de MASTER) y lo escribe por MIDI, llamando a la subrutina \$341b2. Llama a la subrutina \$188f0. Después mueve el contenido de D0 a \$62754, y realiza un tst sobre el, si no es igual a 0 salta a \$0193D4, si lo es continua realizando un clear sobre D0, y saltando a \$0193D6. [\$0193D4]. Mueve#1 a D0, y continua en \$0193D6, el cual mueve el contenido de D0 a -2(a6) y salta a \$01940A.

```

move.w #3,(a7)      ; 019378: 3EBC 0003 //Lectura por MIDI
move.w #2,-(a7)     ; 01937C: 3F3C 0002
jsr $341b2          ; 019380: 4EB9 0003 41B2
addq.l #2,a7        ; 019386: 548F
and.l #ff,d0        ; 019388: C0BC 0000 00FF
move.w #3,(a7)      ; 01938E: 3EBC 0003 //Comprobación para leer MIDI
move.w #1,-(a7)     ; 019392: 3F3C 0001
jsr $341b2          ; 019396: 4EB9 0003 41B2
addq.l #2,a7        ; 01939C: 548F
tst.l d0            ; 01939E: 4A80
bne.s -42 {$019378} ; 0193A0: 66D6
clr.w (a7)          ; 0193A2: 4257
move.w #3,-(a7)     ; 0193A4: 3F3C 0003 //Escritura por MIDI "0x00"
move.w #3,-(a7)     ; 0193A8: 3F3C 0003
jsr $341b2          ; 0193AC: 4EB9 0003 41B2
addq.l #4,a7        ; 0193B2: 588F
move.w #14,(a7)     ; 0193B4: 3EBC 0014
jsr $188f0          ; 0193B8: 4EB9 0001 88F0
move.w d0,$62754    ; 0193BE: 33C0 0006 2754
tst.w $62754        ; 0193C4: 4A79 0006 2754
bne .l +$8 {$0193D4} ; 0193CA: 6600 0008

```

Figura B.2: Fragmento descrito para la dirección 019378

- 01978C: Escritura en MIDI.[01971A] Aquí empezamos con o#7 o#2 en D0 dependiendo de un beq anterior, después se mueve el contenido de \$-4(a6) a A1, y se realiza un add de A1 sobre si mismo, y se le suma #\$41ae0. Después se mueve el contenido de D0 a la dirección de (A1), se mueve -4(a6) a A1, y se le vuelve a sumar a si mismo y #\$38d2e, y se mueve D0 a (A1). Después se mueve -4(a6) a D0, y se realiza un muls de #\$84 sobre D0, y se le suma #\$4129e, y se mueve con movea D0 a A0, y se realiza un clr de 14(A0). Se le suma#1 a -4(a6) y se compara con#10 el contenido, si es menor que 10 se salta a \$01970A, si no, se mueve el contenido de \$62760 a A7 y se llama a la subrutina \$1c96a. Realiza un tst sobre \$62754, si no es 0 salta a \$019830, si si que lo es realiza un tst sobre \$637de, si no es 0 salta a \$019830, si si que lo es realiza un tst sobre 8(a6), si no es 0 salta a \$01980A, si si que lo es mueve #\$11 a (A7), y llama a la subrutina \$341a2(XBIOS). Mueve para escribir a A7 lo que tiene almacenado en D0. Después mueve el contenido de D0 a \$626b0 y realiza un tst, si es positivo salta a \$0197BC, si no mueve #-1 a D0 y salta a \$01B098.[0197BC] -¿Escritura \$019856 (ver Figura B.3).
- 0197C8: Escritura en MIDI. Mueve#11 a A7 y llama a la subrutina \$341a2(XBIOS), después mueve el contenido de D0 a A7 y lo escribe por MIDI. Después mueve D0 a -10(a6) y realiza un tst sobre el mismo, si es positivo salta a \$0197F4, si no mueve#-1 a D0 y salta a \$01B098.[0197F4]. Mueve el contenido de \$626b0 a D0, realiza un asl (left shift) de#8 sobre D0, y suma el contenido de -10(a6) a D0, y finalmente mueve

```

moveq #2,d0          ; 019714: 7002
bra .l +$4 {$01971C} ; 019716: 6000 0004
moveq #7,d0          ; 01971A: 7007
movea.w -4(a6),a1     ; 01971C: 326E FFFC
adda.l a1,a1          ; 019720: D3C9
adda.l #$41ae0,a1     ; 019722: D3FC 0004 1AE0
move.w d0,(a1)        ; 019728: 3280
movea.w -4(a6),a1     ; 01972A: 326E FFFC
adda.l a1,a1          ; 01972E: D3C9
adda.l #$38d2e,a1     ; 019730: D3FC 0003 8D2E
move.w d0,(a1)        ; 019736: 3280
move.w -4(a6),d0      ; 019738: 302E FFFC
muls #$84,d0          ; 01973C: C1FC 0084
add.l #$4129e,d0      ; 019740: D0BC 0004 129E
movea.l d0,a0         ; 019746: 2040
clr.w l4(a0)          ; 019748: 4268 000E
addq.w #1,-4(a6)      ; 01974C: 526E FFFC
cmpi.w #$10,-4(a6)    ; 019750: 0C6E 0010 FFFC
blt.s -78 {$01970A}   ; 019756: 6DB2
move.w $62760,(a7)    ; 019758: 3EB9 0006 2760
jsr $1c96a            ; 01975E: 4EB9 0001 C96A
tst.w $62754          ; 019764: 4A79 0006 2754
bne .l +$c4 {$019830} ; 01976A: 6600 00C4
tst.w $637de          ; 01976E: 4A79 0006 37DE
bne .l +$ba {$019830} ; 019774: 6600 00BA
tst.w 8(a6)           ; 019778: 4A6E 0008
bne .l +$8c {$01980A} ; 01977C: 6600 008C
move.w #$11,(a7)      ; 019780: 3EBC 0011
jsr $341a2            ; 019784: 4EB9 0003 41A2
move.l d0,(a7)        ; 01978A: 2E80
move.w #$3,-(a7)      ; 01978C: 3F3C 0003 //Escritura en MIDI
move.w #$3,-(a7)      ; 019790: 3F3C 0003
jsr $341b2            ; 019794: 4EB9 0003 41B2
addq.l #4,a7          ; 01979A: 588F
move.w #$14,(a7)      ; 01979C: 3EBC 0014
jsr $188f0            ; 0197A0: 4EB9 0001 88F0
move.w d0,$626b0      ; 0197A6: 33C0 0006 26B0
tst.w $626b0          ; 0197AC: 4A79 0006 26B0
bge .l +$8 {$0197BC}  ; 0197B2: 6C00 0008
moveq #-1,d0          ; 0197B6: 70FF
bra .l +$18de {$01B098} ; 0197B8: 6000 18DE

```

Figura B.3: Fragmento descrito para la dirección 01978C

D0 a \$626b0, y salta a \$01982C. (ver Figura B.4).

- 019856: Escritura en MIDI. Mueve el contenido de \$626b0 sobre A7 y lo escribe por MIDI. De nuevo move#14 a A7 y llamada a \$188f0(XBIOS), mueve D0 a -10(a6) y realiza un tst sobre -10(a6), si es mayor que 0 salta a \$019882 (siguiente escritura \$019886), si no mueve#-1 a D0 y salta a \$01B098. (ver Figura B.5, B.6 y B.7).
- 019886: Escritura en MIDI. Mueve a A7 los datos en -10(a6), esto es -10 a la dirección de memoria correspondiente al valor que hay en A6. Si seguimos la ejecución del programa, en \$019870, se mueve el contenido de D0 a la dirección de memoria -10(a6). Se mueve el contenido de \$626b0 a D0, y se realiza un asl de#8 sobre el. Se suma el contenido de -10(a6) a D0, y se mueve D0 a \$626b0, se mueve \$41ade a (A7) y \$6272c a (A7) y se llama a la subrutina \$30b02. Se realiza un tst sobre D0, si D0 es distinto de 0 se salta a \$0198C8, si no continua, mueve#-3 a D0 y salta a \$01B098.
- 0199EA: Escribe por pantalla. Se prepara para escribir por la pantalla, y escribe #\$1b (ver Figuras ?? - 0199FE: Escribe por pantalla. Escribe #\$59. - 019A12: Escribe por pantalla. Escribe #\$29. - 019A40: Escribe por pantalla. Escribe #\$20. - 019A54:

```

move.w #$11, (a7) ; 0197BC: 3EBC 0011
jsr $341a2 ; 0197C0: 4EB9 0003 41A2
move.l d0, (a7) ; 0197C6: 2E80
move.w #$3, -(a7) ; 0197C8: 3F3C 0003 //Escritura en MIDI
move.w #$3, -(a7) ; 0197CC: 3F3C 0003
jsr $341b2 ; 0197D0: 4EB9 0003 41B2
addq.l #4, a7 ; 0197D6: 588F
move.w #$14, (a7) ; 0197D8: 3EBC 0014
jsr $188f0 ; 0197DC: 4EB9 0001 88F0
move.w d0, -10(a6) ; 0197E2: 3D40 FFF6
tst.w -10(a6) ; 0197E6: 4A6E FFF6
bge .l +$8 {$0197F4} ; 0197EA: 6C00 0008
moveq #-1, d0 ; 0197EE: 70FF
bra .l +$18a6 {$01B098} ; 0197F0: 6000 18A6
move.w $626b0, d0 ; 0197F4: 3039 0006 26B0
asl.w #8, d0 ; 0197FA: E140
add.w -10(a6), d0 ; 0197FC: D06E FFF6
move.w d0, $626b0 ; 019800: 33C0 0006 26B0
bra .l +$24 {$01982C}; 019806: 6000 0024

```

Figura B.4: Fragmento descrito para la dirección 0197C8

```

move.w $626b0, (a7); 019850: 3EB9 0006 26B0
move.w #$3, -(a7) ; 019856: 3F3C 0003 //Escritura en MIDI
move.w #$3, -(a7) ; 01985A: 3F3C 0003
jsr $341b2 ; 01985E: 4EB9 0003 41B2
addq.l #4, a7 ; 019864: 588F
move.w #$14, (a7) ; 019866: 3EBC 0014
jsr $188f0 ; 01986A: 4EB9 0001 88F0
move.w d0, -10(a6) ; 019870: 3D40 FFF6
tst.w -10(a6) ; 019874: 4A6E FFF6
bge .l +$8 {$019882}; 019878: 6C00 0008
moveq #-1, d0 ; 01987C: 70FF
bra .l +$1818 {$01B098}; 01987E: 6000 1818
move.w -10(a6), (a7) ; 019882: 3EAE FFF6
move.w #$3, -(a7) ; 019886: 3F3C 0003 //Escritura en MIDI
move.w #$3, -(a7) ; 01988A: 3F3C 0003
jsr $341b2 ; 01988E: 4EB9 0003 41B2
addq.l #4, a7 ; 019894: 588F
move.w $626b0, d0 ; 019896: 3039 0006 26B0
asl.w #8, d0 ; 01989C: E140
add.w -10(a6), d0 ; 01989E: D06E FFF6
move.w d0, $626b0 ; 0198A2: 33C0 0006 26B0
move.w $41ade, (a7) ; 0198A8: 3EB9 0004 1ADE
move.w $6272c, -(a7) ; 0198AE: 3F39 0006 272C
jsr $30b02 ; 0198B4: 4EB9 0003 0B02
addq.l #2, a7 ; 0198BA: 548F
tst.w d0 ; 0198BC: 4A40
bne .l +$8 {$0198C8} ; 0198BE: 6600 0008
moveq #-3, d0 ; 0198C2: 70FD
bra .l +$17d2 {$01B098}; 0198C4: 6000 17D2
tst.w $637de ; 0198C8: 4A79 0006 37DE
beq .l +$8 {$0198D8} ; 0198CE: 6700 0008
jsr $1b3e2 ; 0198D2: 4EB9 0001 B3E2
tst.w $38d10 ; 0198D8: 4A79 0003 8D10
beq .l +$6a {$01994A} ; 0198DE: 6700 006A
clr.w -4(a6) ; 0198E2: 426E FFFC
bra .l +$24 {$01990C} ; 0198E6: 6000 0024

```

Figura B.5: Fragmento descrito para la dirección 019856

Escribe por pantalla. Escribe #\$1b. - 019A68: Escribe por pantalla. Escribe #\$63. - 019A8E: Escribe por pantalla. Escribe #\$f.

- 01A522: Lee de teclado (ver Figura B.9)

```

beq .l +$202 {$019BE6}; 0199E2: 6700 0202
move.w #$1b, (a7)      ; 0199E6: 3EBC 001B
move.w #$2,-(a7)       ; 0199EA: 3F3C 0002 //Escribe por pantalla
move.w #$3,-(a7)       ; 0199EE: 3F3C 0003
jsr $341b2              ; 0199F2: 4EB9 0003 41B2
addq.l #4,a7           ; 0199F8: 588F
move.w #$59, (a7)      ; 0199FA: 3EBC 0059
move.w #$2,-(a7)       ; 0199FE: 3F3C 0002 //Escribe por pantalla
move.w #$3,-(a7)       ; 019A02: 3F3C 0003
jsr $341b2              ; 019A06: 4EB9 0003 41B2
addq.l #4,a7           ; 019A0C: 588F
move.w #$29, (a7)      ; 019A0E: 3EBC 0029
move.w #$2,-(a7)       ; 019A12: 3F3C 0002 //Escribe por pantalla
move.w #$3,-(a7)       ; 019A16: 3F3C 0003
jsr $341b2              ; 019A1A: 4EB9 0003 41B2
addq.l #4,a7           ; 019A20: 588F
tst.w $62658           ; 019A22: 4A79 0006 2658
beq .l +$a {$019A34}   ; 019A28: 6700 000A
move.w #$30, (a7)      ; 019A2C: 3EBC 0030
bra .l +$6 {$019A38}   ; 019A30: 6000 0006
move.w #$18, (a7)      ; 019A34: 3EBC 0018
addi.w #$20, (a7)      ; 019A38: 0657 0020
move.w #$2,-(a7)       ; 019A3C: 3F3C 0002
move.w #$3,-(a7)       ; 019A40: 3F3C 0003 //Escribe por pantalla
jsr $341b2              ; 019A44: 4EB9 0003 41B2
addq.l #4,a7           ; 019A4A: 588F
move.w #$1b, (a7)      ; 019A4C: 3EBC 001B
move.w #$2,-(a7)       ; 019A50: 3F3C 0002
move.w #$3,-(a7)       ; 019A54: 3F3C 0003 //Escribe por pantalla
jsr $341b2              ; 019A58: 4EB9 0003 41B2
addq.l #4,a7           ; 019A5E: 588F
move.w #$63, (a7)      ; 019A60: 3EBC 0063
move.w #$2,-(a7)       ; 019A64: 3F3C 0002
move.w #$3,-(a7)       ; 019A68: 3F3C 0003 //Escribe por pantalla
jsr $341b2              ; 019A6C: 4EB9 0003 41B2
addq.l #4,a7           ; 019A72: 588F
tst.w $62658           ; 019A74: 4A79 0006 2658
beq .l +$a {$019A86}   ; 019A7A: 6700 000A
move.w #$1, (a7)       ; 019A7E: 3EBC 0001
bra .l +$6 {$019A8A}   ; 019A82: 6000 0006
move.w #$f, (a7)       ; 019A86: 3EBC 000F
move.w #$2,-(a7)       ; 019A8A: 3F3C 0002
move.w #$3,-(a7)       ; 019A8E: 3F3C 0003 //Escribe por pantalla
jsr $341b2              ; 019A92: 4EB9 0003 41B2
addq.l #4,a7           ; 019A98: 588F
move.w #$1b, (a7)      ; 019A9A: 3EBC 001B
move.w #$2,-(a7)       ; 019A9E: 3F3C 0002
move.w #$3,-(a7)       ; 019AA2: 3F3C 0003 //Escribe por pantalla
jsr $341b2              ; 019AA6: 4EB9 0003 41B2

```

Figura B.6: Fragmento descrito para la dirección 0199EA

- 01A532: Comprobación lectura teclado. Si es D0=-1, salta a \$01A522 para leer, si no continua la ejecución. Si no, comprueba el contenido de \$637de, si es igual que 0 salta a \$01A5CE, si no mueve el contenido de \$6653a a -4(a6).
- 01A59E: Para analizar esta escritura vamos a la lectura de teclado anterior (\$1A522), ya que lo que se lee se almacena en D0, partimos de eso. Se realiza tst sobre lo leído de teclado, si no es igual a 0, se salta a \$01A522 (volver a leer), si lo es se realiza tst sobre \$637de, si es igual a 0 se salta a \$01A5CE, si no lo es se mueve el contenido de \$6653a a -4(a6), se mueve #\$14 a A7 y se llama a la subrutina \$188f0. Después movemos D0 a A7, movea de -4(a6) a A0, add de A0 sobre A0, y sumar #\$38be0 a A0. Se mueve (A7)+ sobre (a0), movea de -4(a6) sobre A0, y se suma A0 sobre A0, y se suma #\$38be0 a A0 de nuevo. Se realiza tst de (a0), si es mayor que 0 se salta a \$01A58A, si no se mueve #-1 a D0 y se salta a \$01B098. [\$01A58A] -¿Movemos -4(a6) a A0, se suma A0 sobre A0, y se mueve #\$38be0 a D1, y por ultimo se mueve 0(a0,d1.L)

```

move.w #$62, (a7)      ; 019AAE: 3EBC 0062
move.w #$2, -(a7)      ; 019AB2: 3F3C 0002
move.w #$3, -(a7)      ; 019AB6: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019ABA: 4EB9 0003 41B2
addq.l #4, a7          ; 019AC0: 588F
clr.w (a7)             ; 019AC2: 4257
move.w #$2, -(a7)      ; 019AC4: 3F3C 0002
move.w #$3, -(a7)      ; 019AC8: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019ACC: 4EB9 0003 41B2
addq.l #4, a7          ; 019AD2: 588F
move.l #$355a8, (a7)   ; 019AD4: 2EBC 0003 55A8
jsr $1c250             ; 019ADA: 4EB9 0001 C250
jsr $1920e             ; 019AE0: 4EB9 0001 920E
move.w #$1b, (a7)      ; 019AE6: 3EBC 001B
move.w #$2, -(a7)      ; 019AEA: 3F3C 0002
move.w #$3, -(a7)      ; 019AEE: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019AF2: 4EB9 0003 41B2
addq.l #4, a7          ; 019AF8: 588F
move.w #$59, (a7)      ; 019AFA: 3EBC 0059
move.w #$2, -(a7)      ; 019AFE: 3F3C 0002
move.w #$3, -(a7)      ; 019B02: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019B06: 4EB9 0003 41B2
addq.l #4, a7          ; 019B0C: 588F
move.w #$29, (a7)      ; 019B0E: 3EBC 0029
move.w #$2, -(a7)      ; 019B12: 3F3C 0002
move.w #$3, -(a7)      ; 019B16: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019B1A: 4EB9 0003 41B2
addq.l #4, a7          ; 019B20: 588F
tst.w $62658           ; 019B22: 4A79 0006 2658
beq .l +$a {$019B34}; 019B28: 6700 000A
move.w #$30, (a7)      ; 019B2C: 3EBC 0030
bra .l +$6 {$019B38}; 019B30: 6000 0006
move.w #$18, (a7)      ; 019B34: 3EBC 0018
addi.w #$20, (a7)      ; 019B38: 0657 0020
move.w #$2, -(a7)      ; 019B3C: 3F3C 0002
move.w #$3, -(a7)      ; 019B40: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019B44: 4EB9 0003 41B2
addq.l #4, a7          ; 019B4A: 588F
move.w #$1b, (a7)      ; 019B4C: 3EBC 001B
move.w #$2, -(a7)      ; 019B50: 3F3C 0002
move.w #$3, -(a7)      ; 019B54: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019B58: 4EB9 0003 41B2
addq.l #4, a7          ; 019B5E: 588F
move.w #$63, (a7)      ; 019B60: 3EBC 0063
move.w #$2, -(a7)      ; 019B64: 3F3C 0002
move.w #$3, -(a7)      ; 019B68: 3F3C 0003 //Escribe por pantalla
jsr $341b2             ; 019B6C: 4EB9 0003 41B2

```

Figura B.7: Fragmento descrito para la dirección 0199EA

a A7, esto es la suma de A0 y D1 mas 0, a A7 y se escribe por MIDI. Después se resta#1 a -4(a6), si el resultado es mayor de 0 se salta a \$01A5BE, si no se mueve el contenido de \$62760 a D0, y se resta#1 a D0, se mueve D0 a -4(A6), y se realiza un cmp de \$6653a y D0, si no son iguales se salta a \$01A558(volver a empezar la escritura de \$01A59E), si lo son se salta a \$01A648 (ver Figura B.10).

- 01A5EA: Escritura por MIDI. movea de -4(a6) a A0, add de A0 sobre A0, y move de #\$38be0 a D1, finalmente move de 0(a0,d1.L) sobre A7, como antes, y se escribe por MIDI. Después se le resta#1 a -4(a6), si es mayor que 0 se salta a \$01A60A, si no se continua y se mueve el contenido de \$62760 a D0, y se resta#1 a D0, y D0 a -4(a6), y se mueve #\$1e a A7, y se llama a la subrutina \$188f0 (XBIOS) (ver Figura B.11).
- 01A756: Escritura por MIDI. Escribe #\$82 por MIDI (ver Figura B.12).

```

tst.w $62658          ; 019B74: 4A79 0006 2658
beq .l +$a {$019B86} ; 019B7A: 6700 000A
move.w #$1, (a7)      ; 019B7E: 3EBC 0001
bra .l +$6 {$019B8A} ; 019B82: 6000 0006
move.w #$f, (a7)      ; 019B86: 3EBC 000F
move.w #$2, -(a7)     ; 019B8A: 3F3C 0002
move.w #$3, -(a7)     ; 019B8E: 3F3C 0003 //Escribe por pantalla
jsr $341b2            ; 019B92: 4EB9 0003 41B2
addq.l #4, a7         ; 019B98: 588F
move.w #$1b, (a7)     ; 019B9A: 3EBC 001B
move.w #$2, -(a7)     ; 019B9E: 3F3C 0002
move.w #$3, -(a7)     ; 019BA2: 3F3C 0003 //Escribe por pantalla
jsr $341b2            ; 019BA6: 4EB9 0003 41B2
addq.l #4, a7         ; 019BAC: 588F
move.w #$62, (a7)     ; 019BAE: 3EBC 0062
move.w #$2, -(a7)     ; 019BB2: 3F3C 0002
move.w #$3, -(a7)     ; 019BB6: 3F3C 0003 //Escribe por pantalla
jsr $341b2            ; 019BBA: 4EB9 0003 41B2
addq.l #4, a7         ; 019BC0: 588F
clr.w (a7)            ; 019BC2: 4257
move.w #$2, -(a7)     ; 019BC4: 3F3C 0002
move.w #$3, -(a7)     ; 019BC8: 3F3C 0003 //Escribe por pantalla
jsr $341b2            ; 019BCC: 4EB9 0003 41B2
addq.l #4, a7         ; 019BD2: 588F
move.l #$355b8, (a7)  ; 019BD4: 2EBC 0003 55B8

```

Figura B.8: Fragmento descrito para la dirección 0199EA

```

move.w #$2, (a7) ; 01A522: 3EBC 0002 //Lectura de teclado
move.w #$2, -(a7) ; 01A526: 3F3C 0002
jsr $341b2 ; 01A52A: 4EB9 0003 41B2
addq.l #2, a7 ; 01A530: 548F
move.w #$2, (a7) ; 01A532: 3EBC 0002 //Comprobación lectura teclado
move.w #$1, -(a7) ; 01A536: 3F3C 0001
jsr $341b2 ; 01A53A: 4EB9 0003 41B2
addq.l #2, a7 ; 01A540: 548F
tst.l d0 ; 01A542: 4A80
bne.s -36 {$01A522} ; 01A544: 66DC
tst.w $637de ; 01A546: 4A79 0006 37DE
beq .l +$80 {$01A5CE}; 01A54C: 6700 0080
move.w $6653a, -4(a6) ; 01A550: 3D79 0006 653A FFFC
move.w #$14, (a7) ; 01A558: 3EBC 0014

```

Figura B.9: Fragmento descrito para la dirección 01A522

- 01A796: Lectura en MIDI. Lee de MIDI, el suma #\$ff y almacena el resultado en -4(a6).
- 01A7B4: Escritura en MIDI. Mueve la lectura de \$01A796 a A7 y la escribe por MIDI.
- 01B0A4: (Comienza con un link de A6). Comprobación de lectura de teclado. Si hay, D0 = -1, realiza un clear de la dirección de memoria \$38d50 -4(a6) y de continua la ejecución en \$01B0C4. Si no lo hay, D0 = 0, salta a \$01B222 (ver Figura B.13).
- 01B0C4: Lee de teclado y lo almacena en -2(a6), y salta a \$01B1DC.
- 02D718: Lectura de MIDI. Lee de MIDI y le suma #\$ff (ver Figura B.14).
- 02D72A: Comprobación de MIDI. Comprueba si hay algo para leer en MIDI, si no la hay (0), continua la ejecución del programa, y se prepara para escribir \$02D742. Si lo hay (-1), salta a \$02D714 (la lectura anterior).
- 02D742: Escritura por MIDI. Escribe #\$85 (MIDI-ABOUT) por MIDI.

```

move.w #$14, (a7)      ; 01A558: 3EBC 0014
jsr $188f0             ; 01A55C: 4EB9 0001 88F0
move.w d0, -(a7)       ; 01A562: 3F00
movea.w -4(a6), a0     ; 01A564: 306E FFFC
adda.l a0, a0          ; 01A568: D1C8
adda.l #$38be0, a0     ; 01A56A: D1FC 0003 8BE0
move.w (a7)+, (a0)     ; 01A570: 309F
movea.w -4(a6), a0     ; 01A572: 306E FFFC
adda.l a0, a0          ; 01A576: D1C8
adda.l #$38be0, a0     ; 01A578: D1FC 0003 8BE0
tst.w (a0)             ; 01A57E: 4A50
bge .l +$8 {$01A58A}   ; 01A580: 6C00 0008
moveq #-1, d0          ; 01A584: 70FF
bra .l +$b10 {$01B098}; 01A586: 6000 0B10
movea.w -4(a6), a0     ; 01A58A: 306E FFFC
adda.l a0, a0          ; 01A58E: D1C8
move.l #$38be0, d1     ; 01A590: 223C 0003 8BE0
move.w 0(a0, d1.L), (a7) ; 01A596: 3EB0 1800
move.w #$3, -(a7)      ; 01A59A: 3F3C 0003
move.w #$3, -(a7)      ; 01A59E: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 01A5A2: 4EB9 0003 41B2
addq.l #4, a7          ; 01A5A8: 588F
subq.w #1, -4(a6)      ; 01A5AA: 536E FFFC
bge .l +$e {$01A5BE}   ; 01A5AE: 6C00 000E
move.w $62760, d0      ; 01A5B2: 3039 0006 2760
subq.w #1, d0          ; 01A5B8: 5340
move.w d0, -4(a6)      ; 01A5BA: 3D40 FFFC
move.w -4(a6), d0      ; 01A5BE: 302E FFFC
cmp.w $6653a, d0       ; 01A5C2: B079 0006 653A
bne.s -114 {$01A558}   ; 01A5C8: 668E
bra .l +$7c {$01A648}  ; 01A5CA: 6000 007C

```

Figura B.10: Fragmento descrito para la dirección 01A59E

```

movea.w -4(a6), a0     ; 01A5D6: 306E FFFC
adda.l a0, a0          ; 01A5DA: D1C8
move.l #$38be0, d1     ; 01A5DC: 223C 0003 8BE0
move.w 0(a0, d1.L), (a7) ; 01A5E2: 3EB0 1800
move.w #$3, -(a7)      ; 01A5E6: 3F3C 0003
move.w #$3, -(a7)      ; 01A5EA: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 01A5EE: 4EB9 0003 41B2
addq.l #4, a7          ; 01A5F4: 588F
subq.w #1, -4(a6)      ; 01A5F6: 536E FFFC
bge .l +$e {$01A60A}   ; 01A5FA: 6C00 000E
move.w $62760, d0      ; 01A5FE: 3039 0006 2760
subq.w #1, d0          ; 01A604: 5340
move.w d0, -4(a6)      ; 01A606: 3D40 FFFC
move.w #$1e, (a7)      ; 01A60A: 3EBC 001E
jsr $188f0             ; 01A60E: 4EB9 0001 88F0

```

Figura B.11: Fragmento descrito para la dirección 01A5EA

- 02D8F6: Lectura de MIDI- Lee de MIDI y le suma #\$ff (ver Figura B.15).
- 02D90C: Comprobación de MIDI- Comprueba si hay algo para leer en MIDI, si no la hay (0), continua la ejecución del programa, y se prepara para escribir \$02D920. Si lo hay (-1), salta a \$02D8F6 (la lectura anterior).
- 02D920: Escritura por MIDI. Escribe #\$81 por MIDI (MIDI-RESET-SCORE). Llama a la subrutina \$188f0, realiza un clear sobre -2(a6) y salta a \$02D954.
- 02D9CA: Lectura por MIDI. Lee de MIDI y le suma #\$ff (ver Figura B.16).
- 02D9DC: Comprobación de MIDI. Comprueba si hay algo para leer en MIDI, si no la hay (0), continua la ejecución del programa, y se prepara para escribir \$02D9F8. Si lo

```

move.w #$82, (a7)      ; 01A74E: 3EBC 0082
move.w #$3, -(a7)      ; 01A752: 3F3C 0003
move.w #$3, -(a7)      ; 01A756: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 01A75A: 4EB9 0003 41B2
addq.l #4, a7          ; 01A760: 588F
move.w #$14, (a7)      ; 01A762: 3EBC 0014
jsr $188f0             ; 01A766: 4EB9 0001 88F0
move.w d0, -4(a6)      ; 01A76C: 3D40 FFFC
tst.w -4(a6)           ; 01A770: 4A6E FFFC
bge .l +$8 {$01A77E}   ; 01A774: 6C00 0008
moveq #-1, d0          ; 01A778: 70FF
bra .l +$91c {$01B098} ; 01A77A: 6000 091C
bra .l +$4c {$01A7CC}  ; 01A77E: 6000 004C
jsr $1bf60             ; 01A782: 4EB9 0001 BF60
move.w #$a, (a7)       ; 01A788: 3EBC 000A
jsr $2fa0e             ; 01A78C: 4EB9 0002 FA0E
move.w #$3, (a7)       ; 01A792: 3EBC 0003
move.w #$2, -(a7)      ; 01A796: 3F3C 0002 //Lectura de MIDI
jsr $341b2             ; 01A79A: 4EB9 0003 41B2
addq.l #2, a7          ; 01A7A0: 548F
and.l #$ff, d0         ; 01A7A2: C0BC 0000 00FF
move.w d0, -4(a6)      ; 01A7A8: 3D40 FFFC
move.w -4(a6), (a7)    ; 01A7AC: 3EAE FFFC
move.w #$3, -(a7)      ; 01A7B0: 3F3C 0003
move.w #$3, -(a7)      ; 01A7B4: 3F3C 0003 //Escritura en MIDI
jsr $341b2             ; 01A7B8: 4EB9 0003 41B2
addq.l #4, a7          ; 01A7BE: 588F

```

Figura B.12: Fragmento descrito para la dirección 01A756

```

link a6, #-8           ; 01B09C: 4E56 FFF8
move.w #$2, (a7)       ; 01B0A0: 3EBC 0002
move.w #$1, -(a7)      ; 01B0A4: 3F3C 0001 //Comprobación de lectura
jsr $341b2             ; 01B0A8: 4EB9 0003 41B2
addq.l #2, a7          ; 01B0AE: 548F
tst.l d0               ; 01B0B0: 4A80
beq .l +$16e {$01B222} ; 01B0B2: 6700 016E
clr.w $38d50           ; 01B0B6: 4279 0003 8D50
clr.w -4(a6)           ; 01B0BC: 426E FFFC
move.w #$2, (a7)       ; 01B0C0: 3EBC 0002
move.w #$2, -(a7)      ; 01B0C4: 3F3C 0002 //Lectura de teclado
jsr $341b2             ; 01B0C8: 4EB9 0003 41B2
addq.l #2, a7          ; 01B0CE: 548F
and.l #$ff, d0         ; 01B0D0: C0BC 0000 00FF
move.w d0, -2(a6)      ; 01B0D6: 3D40 FFFE
bra .l +$100 {$01B1DC} ; 01B0DA: 6000 0100

```

Figura B.13: Fragmento descrito para la dirección 01B0A4

```

move.w #$3, (a7)       ; 02D714: 3EBC 0003
move.w #$2, -(a7)      ; 02D718: 3F3C 0002 //Lectura de MIDI
jsr $341b2             ; 02D71C: 4EB9 0003 41B2
addq.l #2, a7          ; 02D722: 548F
and.l #$ff, d0         ; 02D724: C0BC 0000 00FF
move.w #$3, (a7)       ; 02D72A: 3EBC 0003 //Comprobacion de MIDI
move.w #$1, -(a7)      ; 02D72E: 3F3C 0001
jsr $341b2             ; 02D732: 4EB9 0003 41B2
addq.l #2, a7          ; 02D738: 548F
tst.l d0               ; 02D73A: 4A80
bne.s -42 {$02D714}    ; 02D73C: 66D6
move.w #$85, (a7)      ; 02D73E: 3EBC 0085
move.w #$3, -(a7)      ; 02D742: 3F3C 0003
move.w #$3, -(a7)      ; 02D746: 3F3C 0003 //Escritura por MIDI

```

Figura B.14: Fragmento descrito para la dirección 02D718

hay (-1), salta a \$02D9CA (la lectura anterior).

- 2D9F8: Escritura por MIDI. Escribe #\$86 por MIDI. Llama a la subrutina \$188f0, y

```

move.w #$3, (a7) ; 02D8F2: 3EBC 0003
move.w #$2, -(a7) ; 02D8F6: 3F3C 0002 //Lectura de MIDI
jsr $341b2 ; 02D8FA: 4EB9 0003 41B2
addq.l #2, a7 ; 02D900: 548F
and.l #$ff, d0 ; 02D902: C0BC 0000 00FF
move.w #$3, (a7) ; 02D908: 3EBC 0003
move.w #$1, -(a7) ; 02D90C: 3F3C 0001 //Comprobacion del estado de MIDI
jsr $341b2 ; 02D910: 4EB9 0003 41B2
addq.l #2, a7 ; 02D916: 548F
tst.l d0 ; 02D918: 4A80
bne.s -42 {$02D8F2} ; 02D91A: 66D6
move.w #$81, (a7) ; 02D91C: 3EBC 0081
move.w #$3, -(a7) ; 02D920: 3F3C 0003
move.w #$3, -(a7) ; 02D924: 3F3C 0003 //Escritura por MIDI
jsr $341b2 ; 02D928: 4EB9 0003 41B2
addq.l #4, a7 ; 02D92E: 588F
move.w #$14, (a7) ; 02D930: 3EBC 0014
jsr $188f0 ; 02D934: 4EB9 0001 88F0
clr.w -2(a6) ; 02D93A: 426E FFFE
bra .l +$14 {$02D954} ; 02D93E: 6000 0014

```

Figura B.15: Fragmento descrito para la dirección 02D8F6

mueve el contenido de D0 a -32(a6). Si el contenido de D0 es mayor de 0, se cumple bge y salta a \$02DA2C (siguiente escritura), si no mueve #\$3727c a (A7), mueve #\$1 a -(A7), llama a la subrutina \$3440a, le suma#2 a A7, y salta a \$02DEA8.

```

move.w #$2, -(a7) ; 02D9CA: 3F3C 0002 //Lectura de MIDI
jsr $341b2 ; 02D9CE: 4EB9 0003 41B2
addq.l #2, a7 ; 02D9D4: 548F
and.l #$ff, d0 ; 02D9D6: C0BC 0000 00FF
move.w #$3, (a7) ; 02D9DC: 3EBC 0003
move.w #$1, -(a7) ; 02D9E0: 3F3C 0001 //Comprobacion de MIDI
jsr $341b2 ; 02D9E4: 4EB9 0003 41B2
addq.l #2, a7 ; 02D9EA: 548F
tst.l d0 ; 02D9EC: 4A80
bne.s -42 {$02D9C6} ; 02D9EE: 66D6
move.w #$86, (a7) ; 02D9F0: 3EBC 0086
move.w #$3, -(a7) ; 02D9F4: 3F3C 0003
move.w #$3, -(a7) ; 02D9F8: 3F3C 0003 //Escritura por MIDI
jsr $341b2 ; 02D9FC: 4EB9 0003 41B2
addq.l #4, a7 ; 02DA02: 588F
move.w #$14, (a7) ; 02DA04: 3EBC 0014
jsr $188f0 ; 02DA08: 4EB9 0001 88F0
move.w d0, -32(a6) ; 02DA0E: 3D40 FFE0
bge .l +$18 {$02DA2C} ; 02DA12: 6C00 0018
move.l #$3727c, (a7) ; 02DA16: 2EBC 0003 727C
move.w #$1, -(a7) ; 02DA1C: 3F3C 0001
jsr $3440a ; 02DA20: 4EB9 0003 440A
addq.l #2, a7 ; 02DA26: 548F
bra .l +$47e {$02DEA8} ; 02DA28: 6000 047E

```

Figura B.16: Fragmento descrito para la dirección 02D9CA

- 02DA34: Escritura por MIDI. Escribe #\$1 por MIDI. Llama a la subrutina \$188f0, mueve D0 a -32(a6), si es menor que 0, se cumple blt y salta a \$02DA16, si no mueve D0 a \$62760 y continua a la siguiente escritura (ver Figura B.17).
- 02DA60: Escritura por MIDI. Escribe por MIDI el registro D0, si vamos a hacia atrás en la ejecución, en el registro D0 esta el contenido de la dirección de memoria -32(a6). Después llama a la subrutina \$188f0, realiza tst sobre D0, si es menor que 0 salta a \$02DA16, si no realiza cmp de #10 con \$62760, si \$62760 es menor o igual que \$62760, salta a \$02DA9C, si no mueve #\$3729d a (A7), y #\$1 a -(A7) y llama a la subrutina \$3440a, y salta a \$02DEA8.

```

move.w #$1, (a7)      ; 02DA2C: 3EBC 0001
move.w #$3, -(a7)     ; 02DA30: 3F3C 0003
move.w #$3, -(a7)     ; 02DA34: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02DA38: 4EB9 0003 41B2
addq.l #4, a7         ; 02DA3E: 588F
move.w #$14, (a7)     ; 02DA40: 3EBC 0014
jsr $188f0            ; 02DA44: 4EB9 0001 88F0
move.w d0, -32(a6)    ; 02DA4A: 3D40 FFE0
blt.s -58 {$02DA16}   ; 02DA4E: 6DC6
move.w -32(a6), d0     ; 02DA50: 302E FFE0
move.w d0, $62760     ; 02DA54: 33C0 0006 2760
move.w d0, (a7)       ; 02DA5A: 3E80
move.w #$3, -(a7)     ; 02DA5C: 3F3C 0003
move.w #$3, -(a7)     ; 02DA60: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02DA64: 4EB9 0003 41B2
addq.l #4, a7         ; 02DA6A: 588F
move.w #$14, (a7)     ; 02DA6C: 3EBC 0014
jsr $188f0            ; 02DA70: 4EB9 0001 88F0
tst.w d0              ; 02DA76: 4A40
blt.s -100 {$02DA16}  ; 02DA78: 6D9C
cmpi.w #$10, $62760   ; 02DA7A: 0C79 0010 0006 2760
ble .l +$18 {$02DA9C} ; 02DA82: 6F00 0018
move.l #$3729d, (a7)  ; 02DA86: 2EBC 0003 729D
move.w #$1, -(a7)     ; 02DA8C: 3F3C 0001
jsr $3440a            ; 02DA90: 4EB9 0003 440A
addq.l #2, a7         ; 02DA96: 548F
bra .l +$40e {$02DEA8} ; 02DA98: 6000 040E

```

Figura B.17: Fragmento descrito para la dirección 02DA34

- 02DAF0: Lectura por MIDI. Lee de MIDI y le suma #\$ff.
- 02DB06: Comprobación de MIDI. Comprueba si hay algo para leer en MIDI, si no la hay (0), continua la ejecución del programa, y se prepara para escribir \$02DB1A. Si lo hay (-1), salta a \$02DAEC (la lectura anterior) (ver Figura B.18).
- 02DB1A: Escritura por MIDI. Escribe #\$80 por MIDI. Llama a la subrutina \$188f0, y mueve D0 a -32(a6), si el contenido es mayor que 0 salta a \$02DB52 (siguiente escritura), si no mueve #\$372e9 a (A7), y #\$1 a -(A7) y llama a la subrutina \$3440a, y finalmente salta a \$02DEA8.
- 02DB5A: Escritura por MIDI. Escribe #\$1 por MIDI. Llama a la subrutina \$188f0, mueve D0 a -32(a6), si el contenido es menor de 0 salta a \$02DB3C, si no mueve D0 a \$62760 y continua (escritura siguiente) (ver Figura B.19).
- 02DB86: Escritura por MIDI. Mueve D0 de la anterior escritura (02DB5A) a (A7) y lo escribe por MIDI. Llama a la subrutina \$188f0, realiza tst sobre D0, si es menor que 0 salta a \$02DB3C, si no continua y mueve #\$3730a a (a7), y #\$1 a -(a7), llama a la subrutina \$3440a y salta a \$02DEA8.
- 02DC3A: Escribe #\$80 por MIDI (MIDI-COUNT-PLAYERS). Llamada a \$188f0 y comprobaciones (ver Figura B.20).
- 02DC82: Escribe #\$84 por MIDI (MIDI-START-GAME). Llamada a \$188f0 y comprobaciones.
- 02E86A: Lee de MIDI y le suma #\$ff.
- 02E880: Comprobación de MIDI. Comprueba si hay algo para leer por MIDI, si lo hay

```

move.w #$3, (a7) ; 02DAEC: 3EBC 0003
move.w #$2, -(a7) ; 02DAF0: 3F3C 0002 //Lectura por MIDI
jsr $341b2 ; 02DAF4: 4EB9 0003 41B2
addq.l #2, a7 ; 02DAFA: 548F
and.l #$ff, d0 ; 02DAFC: C0BC 0000 00FF
move.w #$3, (a7) ; 02DB02: 3EBC 0003
move.w #$1, -(a7) ; 02DB06: 3F3C 0001 //Comprobacion de MIDI
jsr $341b2 ; 02DB0A: 4EB9 0003 41B2
addq.l #2, a7 ; 02DB10: 548F
tst.l d0 ; 02DB12: 4A80
bne.s -42 {$02DAEC} ; 02DB14: 66D6
move.w #$80, (a7) ; 02DB16: 3EBC 0080
move.w #$3, -(a7) ; 02DB1A: 3F3C 0003
move.w #$3, -(a7) ; 02DB1E: 3F3C 0003 //Escritura por MIDI
jsr $341b2 ; 02DB22: 4EB9 0003 41B2
addq.l #4, a7 ; 02DB28: 588F
move.w #$14, (a7) ; 02DB2A: 3EBC 0014
jsr $188f0 ; 02DB2E: 4EB9 0001 88F0
move.w d0, -32(a6) ; 02DB34: 3D40 FFE0
bge .l +$18 {$02DB52} ; 02DB38: 6C00 0018
move.l #$372e9, (a7) ; 02DB3C: 2EBC 0003 72E9
move.w #$1, -(a7) ; 02DB42: 3F3C 0001
jsr $3440a ; 02DB46: 4EB9 0003 440A
addq.l #2, a7 ; 02DB4C: 548F
bra .l +$358 {$02DEA8}; 02DB4E: 6000 0358

```

Figura B.18: Fragmento descrito para la dirección 02DAF0

```

move.w #$1, (a7) ; 02DB52: 3EBC 0001
move.w #$3, -(a7) ; 02DB56: 3F3C 0003
move.w #$3, -(a7) ; 02DB5A: 3F3C 0003 //Escritura por MIDI
jsr $341b2 ; 02DB5E: 4EB9 0003 41B2
addq.l #4, a7 ; 02DB64: 588F
move.w #$14, (a7) ; 02DB66: 3EBC 0014
jsr $188f0 ; 02DB6A: 4EB9 0001 88F0
move.w d0, -32(a6) ; 02DB70: 3D40 FFE0
blt.s -58 {$02DB3C} ; 02DB74: 6DC6
move.w -32(a6), d0 ; 02DB76: 302E FFE0
move.w d0, $62760 ; 02DB7A: 33C0 0006 2760
move.w d0, (a7) ; 02DB80: 3E80
move.w #$3, -(a7) ; 02DB82: 3F3C 0003
move.w #$3, -(a7) ; 02DB86: 3F3C 0003 //Escritura por MIDI
jsr $341b2 ; 02DB8A: 4EB9 0003 41B2
addq.l #4, a7 ; 02DB90: 588F
move.w #$14, (a7) ; 02DB92: 3EBC 0014
jsr $188f0 ; 02DB96: 4EB9 0001 88F0
tst.w d0 ; 02DB9C: 4A40
blt.s -100 {$02DB3C}; 02DB9E: 6D9C
cmpi.w #$10, $62760 ; 02DBA0: 0C79 0010 0006 2760
ble .l +$18 {$02DBC2} ; 02DBA8: 6F00 0018
move.l #$3730a, (a7) ; 02DBAC: 2EBC 0003 730A
move.w #$1, -(a7) ; 02DBB2: 3F3C 0001
jsr $3440a ; 02DBB6: 4EB9 0003 440A
addq.l #2, a7 ; 02DBBC: 548F
bra .l +$2e8 {$02DEA8} ; 02DBBE: 6000 02E8

```

Figura B.19: Fragmento descrito para la dirección 02DB5A

(-1) salta a \$02E866 y lo lee, si no lo hay (0) salta a \$02E8A4 (comprobar para leer de teclado) (ver Figura B.21).

- 02E894: Lee de teclado.
- 02E8A4: Comprueba para leer de teclado. Si no hay nada salta a \$02E9FC

```

move.w #$80, (a7)           ; 02DC32: 3EBC 0080
move.w #$3, -(a7)           ; 02DC36: 3F3C 0003
move.w #$3, -(a7)           ; 02DC3A: 3F3C 0003 //Escritura por MIDI
jsr $341b2                   ; 02DC3E: 4EB9 0003 41B2
addq.l #4, a7                ; 02DC44: 588F
move.w #$14, (a7)           ; 02DC46: 3EBC 0014
jsr $188f0                   ; 02DC4A: 4EB9 0001 88F0
move.w d0, -32(a6)           ; 02DC50: 3D40 FFE0
blt .l -$11a {$02DB3C}       ; 02DC54: 6D00 FEE6
tst.w -30(a6)                ; 02DC58: 4A6E FFE2
bge .l +$14 {$02DC72}        ; 02DC5C: 6C00 0014
clr.l (a7)                   ; 02DC60: 4297
move.w #$101, -(a7)          ; 02DC62: 3F3C 0101
jsr $344ec                   ; 02DC66: 4EB9 0003 44EC
addq.l #2, a7                ; 02DC6C: 548F
bra .l +$238 {$02DEA8}       ; 02DC6E: 6000 0238
tst.w 8(a6)                  ; 02DC72: 4A6E 0008
bne .l +$26 {$02DC9E}        ; 02DC76: 6600 0026
move.w #$84, (a7)            ; 02DC7A: 3EBC 0084
move.w #$3, -(a7)            ; 02DC7E: 3F3C 0003
move.w #$3, -(a7)            ; 02DC82: 3F3C 0003 //Escritura por MIDI
jsr $341b2                   ; 02DC86: 4EB9 0003 41B2
addq.l #4, a7                ; 02DC8C: 588F
move.w #$14, (a7)           ; 02DC8E: 3EBC 0014
jsr $188f0                   ; 02DC92: 4EB9 0001 88F0
tst.w d0                     ; 02DC98: 4A40
blt .l -$160 {$02DB3C}       ; 02DC9A: 6D00 FEA0
tst.w 8(a6)                  ; 02DC9E: 4A6E 0008
bne .l +$e {$02DCB2}         ; 02DCA2: 6600 000E
jsr $2f35e                   ; 02DCA6: 4EB9 0002 F35E
tst.w d0                     ; 02DCAC: 4A40
blt .l -$174 {$02DB3C}       ; 02DCAE: 6D00 FE8C
move.w -34(a6), (a7)          ; 02DCB2: 3EAE FFDE
move.w 8(a6), -(a7)           ; 02DCB6: 3F2E 0008
jsr $19466                   ; 02DCBA: 4EB9 0001 9466
addq.l #2, a7                ; 02DCC0: 548F
move.w d0, -32(a6)           ; 02DCC2: 3D40 FFE0

```

Figura B.20: Fragmento descrito para la dirección 02DC3A

- 02E8BC: Comprobación de MIDI. Comprueba si hay algo para leer por MIDI, si lo hay (-1) continua la ejecución \$02E8D6, y lo lee, si no lo hay (0) salta a \$02E90A (comprobar para leer de teclado).
- 02E8D6: Lectura por MIDI. Lee de MIDI y le suma #\$ff y guarda el resultado de D0 en la dirección \$-4(a6). Si el resultado del add #\$ff, d0 es 0, esto activara el flag de Z, y el programa continuara con su ejecución \$02E8F6. En caso contrario el programa salta a \$02E906 (bra a \$02EA00).
- 02E8F6: Escritura por MIDI. Realiza un clr de A7, y escribe por MIDI, por ello escribe un #\$0 en MIDI. Y salta a \$02E90A. Y salta a \$02E90A.
- 02E90A: Comprueba para leer de teclado.
- 02E920: Lee del teclado.
- 02EA1C: Escritura por MIDI. Carga de memoria \$-4(a6) directamente a A7 y lo escribe por MIDI (ver Figura B.22).
- 02EB22: Escritura por MIDI. Carga de memoria \$-4(a6) directamente a A7 y lo escribe por MIDI. Mueve la memoria de \$-4(a6) a D0, le resta #1 y almacena el resultado en la dirección \$6653a, y salta a \$02EB60 (ver Figura B.23).

```

move.w #$3,(a7)          ; 02E866: 3EBC 0003
move.w #$2,-(a7)         ; 02E86A: 3F3C 0002 //Lectura de MIDI
jsr $341b2               ; 02E86E: 4EB9 0003 41B2
addq.l #2,a7             ; 02E874: 548F
and.l #$ff,d0            ; 02E876: C0BC 0000 00FF
move.w #$3,(a7)          ; 02E87C: 3EBC 0003
move.w #$1,-(a7)         ; 02E880: 3F3C 0001 //Comprobación de MIDI
jsr $341b2               ; 02E884: 4EB9 0003 41B2
addq.l #2,a7             ; 02E88A: 548F
tst.l d0                 ; 02E88C: 4A80
bne.s -42 {$02E866}      ; 02E88E: 66D6
bra .l +$12 {$02E8A4}    ; 02E890: 6000 0012
move.w #$2,(a7)          ; 02E894: 3EBC 0002 //Lee de teclado
move.w #$2,-(a7)         ; 02E898: 3F3C 0002
jsr $341b2               ; 02E89C: 4EB9 0003 41B2
addq.l #2,a7             ; 02E8A2: 548F
move.w #$2,(a7)          ; 02E8A4: 3EBC 0002 //Comprueba para leer de teclado
move.w #$1,-(a7)         ; 02E8A8: 3F3C 0001
jsr $341b2               ; 02E8AC: 4EB9 0003 41B2
addq.l #2,a7             ; 02E8B2: 548F
tst.l d0                 ; 02E8B4: 4A80
bne.s -36 {$02E894}      ; 02E8B6: 66DC
bra .l +$142 {$02E9FC}   ; 02E8B8: 6000 0142
move.w #$3,(a7)          ; 02E8BC: 3EBC 0003
move.w #$1,-(a7)         ; 02E8C0: 3F3C 0001 //Comprobacion de MIDI
jsr $341b2               ; 02E8C4: 4EB9 0003 41B2
addq.l #2,a7             ; 02E8CA: 548F
tst.l d0                 ; 02E8CC: 4A80
beq .l +$3a {$02E90A}    ; 02E8CE: 6700 003A
move.w #$3,(a7)          ; 02E8D2: 3EBC 0003
move.w #$2,-(a7)         ; 02E8D6: 3F3C 0002 //Lectura de MIDI
jsr $341b2               ; 02E8DA: 4EB9 0003 41B2
addq.l #2,a7             ; 02E8E0: 548F
and.l #$ff,d0            ; 02E8E2: C0BC 0000 00FF
move.w d0,-4(a6)         ; 02E8E8: 3D40 FFFC
bne .l +$18 {$02E906}    ; 02E8EC: 6600 0018
clr.w (a7)               ; 02E8F0: 4257
move.w #$3,-(a7)         ; 02E8F2: 3F3C 0003
move.w #$3,-(a7)         ; 02E8F6: 3F3C 0003 //Escritura en MIDI
jsr $341b2               ; 02E8FA: 4EB9 0003 41B2
addq.l #4,a7             ; 02E900: 588F
bra .l +$6 {$02E90A}     ; 02E902: 6000 0006
bra .l +$f8 {$02EA00}    ; 02E906: 6000 00F8
move.w #$2,(a7)          ; 02E90A: 3EBC 0002 //Comprueba para leer de teclado
move.w #$1,-(a7)         ; 02E90E: 3F3C 0001
jsr $341b2               ; 02E912: 4EB9 0003 41B2
addq.l #2,a7             ; 02E918: 548F
tst.l d0                 ; 02E91A: 4A80
beq .l +$de {$02E9FC}    ; 02E91C: 6700 00DE
move.w #$2,(a7)          ; 02E920: 3EBC 0002 //Lee de teclado
move.w #$2,-(a7)         ; 02E924: 3F3C 0002
jsr $341b2               ; 02E928: 4EB9 0003 41B2
addq.l #2,a7             ; 02E92E: 548F

```

Figura B.21: Fragmento descrito para la dirección 02E86A

```

move.w -4(a6),(a7) ; 02EA18: 3EAE FFFC
move.w #$3,-(a7)  ; 02EA1C: 3F3C 0003
move.w #$3,-(a7)  ; 02EA20: 3F3C 0003 //Escritura por MIDI
jsr $341b2        ; 02EA24: 4EB9 0003 41B2

```

Figura B.22: Fragmento descrito para la dirección 02EA1C

- 02EB54: Escritura por MIDI. Carga de memoria $\$-4(a6)$ a D0, y lo guarda en $\$62754$, mueve el contenido de D0 a A7, y le suma #1, y lo escribe por MIDI. Después si el resultado de D0 es negativo salta a $\$02EE1A$, en caso contrario continua en $\$02EB7E$.
- 02EB7E: Escribe por MIDI el contenido de la dirección de memoria $\$-4(a6)$ contenido en D0 y continua en $\$02EB92$.
- 02EB92: Lectura por MIDI. Lee de MIDI y le suma $\#\$ff$ (ver Figura B.24).

```

move.w -4(a6), (a7)      ; 02EB1E: 3EAE FFFC
move.w #$3, -(a7)       ; 02EB22: 3F3C 0003
move.w #$3, -(a7)       ; 02EB26: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02EB2A: 4EB9 0003 41B2
addq.l #4, a7           ; 02EB30: 588F
move.w -4(a6), d0        ; 02EB32: 302E FFFC
subq.w #1, d0           ; 02EB36: 5340
move.w d0, $6653a        ; 02EB38: 33C0 0006 653A
bra .l +$20 {$02EB60}    ; 02EB3E: 6000 0020
move.w -4(a6), d0        ; 02EB42: 302E FFFC
move.w d0, $62754        ; 02EB46: 33C0 0006 2754
move.w d0, (a7)          ; 02EB4C: 3E80
addq.w #1, (a7)          ; 02EB4E: 5257
move.w #$3, -(a7)       ; 02EB50: 3F3C 0003
move.w #$3, -(a7)       ; 02EB54: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02EB58: 4EB9 0003 41B2
addq.l #4, a7           ; 02EB5E: 588F
move.w #$14, (a7)        ; 02EB60: 3EBC 0014
jsr $188f0              ; 02EB64: 4EB9 0001 88F0
move.w d0, -4(a6)        ; 02EB6A: 3D40 FFFC
blt .l +$2aa {$02EE1A}; 02EB6E: 6D00 02AA
move.w -4(a6), d0        ; 02EB72: 302E FFFC
move.w d0, $62760        ; 02EB76: 33C0 0006 2760
move.w d0, (a7)          ; 02EB7C: 3E80
move.w #$3, -(a7)       ; 02EB7E: 3F3C 0003
move.w #$3, -(a7)       ; 02EB82: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02EB86: 4EB9 0003 41B2
addq.l #4, a7           ; 02EB8C: 588F

```

Figura B.23: Fragmento descrito para la dirección 02EB22

- 02EBAC: Escribe #\$80 (MIDI-COUNT-PLAYERS) por MIDI y salta a \$02EE1A de forma incondicional.

```

move.w #$3, (a7)         ; 02EB8E: 3EBC 0003
move.w #$2, -(a7)        ; 02EB92: 3F3C 0002 //Lectura por MIDI
jsr $341b2              ; 02EB96: 4EB9 0003 41B2
addq.l #2, a7           ; 02EB9C: 548F
and.l #$fff, d0          ; 02EB9E: C0BC 0000 00FF
move.w #$80, (a7)        ; 02EBA4: 3EBC 0080
move.w #$3, -(a7)        ; 02EBA8: 3F3C 0003
move.w #$3, -(a7)        ; 02EBAC: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02EBB0: 4EB9 0003 41B2
addq.l #4, a7           ; 02EBB6: 588F
bra .l +$260 {$02EE1A}   ; 02EBB8: 6000 0260

```

Figura B.24: Fragmento descrito para la dirección 02EB92

- 02ED56: Escritura por MIDI. Guarda D0 en \$-4(a6) para activar los flags correspondientes, por ello si D0 es negativo salta a \$02EE1A, en caso de no serlo realiza un tst sobre la dirección de memoria \$637de, si es#0 salta a 02ED72 (siguiente escritura \$02ED84, por lo que esto es un if), si no lo es continua su ejecución y carga el contenido de la dirección de memoria \$-4(a6) (D0), sobre A7 y lo escribe por MIDI. Posteriormente a lo contenido en \$-4(a6) le resta#1 y lo almacena en \$6653a y salta incondicionalmente a \$02ED90 (ver Figura B.25).
- 02ED84: Escritura por MIDI. Carga el contenido de \$-4(a6) a D0, los almacena en \$62754 y mueve D0 a A7, le suma#1 y lo envía por MIDI. Posteriormente mueve D0 a \$-4(a6) para activar los flags, si D0 es negativo salta a \$02EE1A, si no continua en \$02EDB2.
- 02EDB2: Escritura por MIDI. Carga de \$-4(a6) a D0, lo almacena en \$62760 y mueve D0 a A7, y lo escribe por MIDI. Después realiza un tst sobre \$637de, si el contenido

```

bne .l +$ea {$02EE1A} ; 02ED2E: 6600 00EA
move.w #$14, (a7)      ; 02ED32: 3EBC 0014
jsr $188f0             ; 02ED36: 4EB9 0001 88F0
move.w d0, -4(a6)      ; 02ED3C: 3D40 FFFC
blt .l +$d8 {$02EE1A} ; 02ED40: 6D00 00D8
tst.w $637de           ; 02ED44: 4A79 0006 37DE
beq .l +$26 {$02ED72} ; 02ED4A: 6700 0026
move.w -4(a6), (a7)    ; 02ED4E: 3EAE FFFC
move.w #$3, -(a7)      ; 02ED52: 3F3C 0003
move.w #$3, -(a7)      ; 02ED56: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02ED5A: 4EB9 0003 41B2
addq.l #4, a7          ; 02ED60: 588F
move.w -4(a6), d0      ; 02ED62: 302E FFFC
subq.w #1, d0          ; 02ED66: 5340
move.w d0, $6653a      ; 02ED68: 33C0 0006 653A
bra .l +$20 {$02ED90} ; 02ED6E: 6000 0020
move.w -4(a6), d0      ; 02ED72: 302E FFFC
move.w d0, $62754      ; 02ED76: 33C0 0006 2754
move.w d0, (a7)        ; 02ED7C: 3E80
addq.w #1, (a7)        ; 02ED7E: 5257
move.w #$3, -(a7)      ; 02ED80: 3F3C 0003
move.w #$3, -(a7)      ; 02ED84: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02ED88: 4EB9 0003 41B2
addq.l #4, a7          ; 02ED8E: 588F
move.w #$14, (a7)      ; 02ED90: 3EBC 0014
jsr $188f0             ; 02ED94: 4EB9 0001 88F0
move.w d0, -4(a6)      ; 02ED9A: 3D40 FFFC
blt .l +$7a {$02EE1A} ; 02ED9E: 6D00 007A

```

Figura B.25: Fragmento descrito para la dirección 02ED56

de \$637de es#0 salta a 02EE12, en caso de no serlo continua la ejecución y realiza un clr sobre -2(a6) y salta a \$02EDFC (ver Figura B.26).

- 02EDD6: Escritura por MIDI. Realiza un clr sobre A7 y escribe por MIDI, por lo que enviara por MIDI un#0, y continua en \$02EDE6.
- 02EDE2: Lectura por MIDI. Lee de MIDI y le suma #\$ff. Añade #1 a -\$2(a6) y lo guarda en D0, realiza un cmp de \$62760 y D0, si el contenido de \$62760 es menor que el de D0 vuelve a salta a \$02EDD0 para volver a escribir un#0 por MIDI, en caso contrario continua con su ejecución, llama a la subrutina \$2f080 y salta a \$02EE1A.
- 02F044: Escritura por MIDI. Realiza un clr sobre A7 y escribe por MIDI, por lo que enviara por MIDI un#0, y continua en \$02F050 (ver Figura B.27) .
- 02F050: Lectura por MIDI. Lee de MIDI y le suma #\$ff. Añade #1 a -\$2(a6) y lo guarda en D0, realiza un cmp de \$62760 y D0, si el contenido de \$62760 es menor que el de D0 vuelve a salta a \$02F03E para volver a escribir un#0 por MIDI, en caso contrario continua con su ejecución, llama a la subrutina \$2f080, y realiza un unlk de A6 y rts.
- 02F100: Escribe el contenido de D0 por MIDI (ver Figura B.28).
- 02F18E: Escritura por MIDI. Escribe el contenido de D0 por MIDI (ver Figura B.29).
- 02F28A: Mostrar por pantalla. Carga la dirección de memoria -\$6(a6) usando MOVEA sobre A0, mueve el contenido de A0 a D0, realiza un ext sobre D0, para extender su valor en caso de ser negativo, y lo escribe por pantalla (ver Figura B.30).
- 02F30E: Lo mismo que en \$02F28A. Pero el bne salta a \$02F304, y si no salta mueve

```

move.w -4(a6),d0      ; 02EDA2: 302E FFFC
move.w d0,$62760      ; 02EDA6: 33C0 0006 2760
move.w d0,(a7)        ; 02EDAC: 3E80
move.w #$3,-(a7)      ; 02EDAE: 3F3C 0003
move.w #$3,-(a7)      ; 02EDB2: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02EDB6: 4EB9 0003 41B2
addq.l #4,a7          ; 02EDBC: 588F
tst.w $637de         ; 02EDBE: 4A79 0006 37DE
beq .l +$4c {$02EE12} ; 02EDC4: 6700 004C
clr.w -2(a6)          ; 02EDC8: 426E FFFE
bra .l +$2e {$02EDFC} ; 02EDCC: 6000 002E
clr.w (a7)            ; 02EDD0: 4257
move.w #$3,-(a7)      ; 02EDD2: 3F3C 0003
move.w #$3,-(a7)      ; 02EDD6: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02EDDA: 4EB9 0003 41B2
addq.l #4,a7          ; 02EDE0: 588F
move.w #$3,(a7)       ; 02EDE2: 3EBC 0003
move.w #$2,-(a7)      ; 02EDE6: 3F3C 0002 //Lectura por MIDI
jsr $341b2            ; 02EDEA: 4EB9 0003 41B2
addq.l #2,a7          ; 02EDF0: 548F
and.l #$ff,d0         ; 02EDF2: C0BC 0000 00FF
addq.w #1,-2(a6)      ; 02EDF8: 526E FFFE
move.w -2(a6),d0      ; 02EDFC: 302E FFFE
cmp.w $62760,d0       ; 02EE00: B079 0006 2760
blt.s -56 {$02EDD0}   ; 02EE06: 6DC8
jsr $2f080            ; 02EE08: 4EB9 0002 F080
bra .l +$a {$02EE1A}   ; 02EE0E: 6000 000A

```

Figura B.26: Fragmento descrito para la dirección 02EDB2

```

clr.w (a7)            ; 02F03E: 4257
move.w #$3,-(a7)      ; 02F040: 3F3C 0003
move.w #$3,-(a7)      ; 02F044: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02F048: 4EB9 0003 41B2
addq.l #4,a7          ; 02F04E: 588F
move.w #$3,(a7)       ; 02F050: 3EBC 0003
move.w #$2,-(a7)      ; 02F054: 3F3C 0002 //Lectura por MIDI
jsr $341b2            ; 02F058: 4EB9 0003 41B2
addq.l #2,a7          ; 02F05E: 548F
and.l #$ff,d0         ; 02F060: C0BC 0000 00FF
addq.w #1,-2(a6)      ; 02F066: 526E FFFE
move.w -2(a6),d0      ; 02F06A: 302E FFFE
cmp.w $62760,d0       ; 02F06E: B079 0006 2760
blt.s -56 {$02F03E}   ; 02F074: 6DC8
jsr $2f080            ; 02F076: 4EB9 0002 F080
unlk a6               ; 02F07C: 4E5E
rts                  ; 02F07E: 4E75

```

Figura B.27: Fragmento descrito para la dirección 02F044

8(a6) a (a7) y le suma#1 a (a7), llama a la subrutina \$1c1e4 y salta a \$02F35A (ver Figura B.31).

- 02F36A: Lectura por MIDI. Lee por MIDI y le suma #\$ff (ver Figura B.32).
- 02F380: Comprobación de lectura por MIDI. Comprueba si hay algún valor para leer, si lo hay (-1) salta a \$02F366 para leerlo, si no lo hay (0) continua la ejecución en \$02F398.
- 02F398: Escritura por MIDI. Escribe por MIDI #\$83 (MIDI-SEND-DATA). Llama a la subrutina \$188f0, mueve D0 a 6(a6), realiza tst si es mayor que 0 salta a \$02F3C0, si no mueve#-1 a D0 y salta a \$02F6C8.
- 02F3E0: Escritura por MIDI. Carga de memoria \$38d14 sobre A7 y lo escribe por

```

movea.l -8(a6),a0      ; 02F0EA: 206E FFF8
move.w -4(a6),d1      ; 02F0EE: 322E FFFC
ext.l d1              ; 02F0F2: 48C1
move.b 0(a0,d1.L),d0  ; 02F0F4: 1030 1800
ext.w d0              ; 02F0F8: 4880
move.w d0,(a7)        ; 02F0FA: 3E80
move.w #$3,-(a7)      ; 02F0FC: 3F3C 0003
move.w #$3,-(a7)      ; 02F100: 3F3C 0003 //Escritura por MIDI
jsr $341b2            ; 02F104: 4EB9 0003 41B2
addq.l #4,a7          ; 02F10A: 588F
movea.l -8(a6),a0      ; 02F10C: 206E FFF8
move.w -4(a6),d1      ; 02F110: 322E FFFC
ext.l d1              ; 02F114: 48C1
move.b 0(a0,d1.L),d0  ; 02F116: 1030 1800
ext.w d0              ; 02F11A: 4880
addq.w #1,-4(a6)      ; 02F11C: 526E FFFC
tst.b d0              ; 02F120: 4A00
bne.s -104 {$02F0BC}  ; 02F122: 6698
subq.w #1,-2(a6)      ; 02F124: 536E FFFE
bge .l +$e {$02F138}  ; 02F128: 6C00 000E
move.w $62760,d0      ; 02F12C: 3039 0006 2760
subq.w #1,d0          ; 02F132: 5340
move.w d0,-2(a6)      ; 02F134: 3D40 FFFE
move.w -2(a6),d0      ; 02F138: 302E FFFE
cmp.w $6653a,d0       ; 02F13C: B079 0006 653A
bne .l -$ae {$02F096} ; 02F142: 6600 FF52
bra .l +$f8 {$02F240} ; 02F146: 6000 00F8

```

Figura B.28: Fragmento descrito para la dirección 02F100

```

move.w $62754,-2(a6) ; 02F14A: 3D79 0006 2754 FFFE
movea.w -2(a6),a0    ; 02F152: 306E FFFE
adda.l a0,a0         ; 02F156: D1C8
move.l #$36dc0,d1    ; 02F158: 223C 0003 6DC0
move.w 0(a0,d1.L),d0 ; 02F15E: 3030 1800
muls #$18,d0         ; 02F162: C1FC 0018
add.l $6651e,d0      ; 02F166: D0B9 0006 651E
movea.l d0,a0        ; 02F16C: 2040
move.l 12(a0),-8(a6) ; 02F16E: 2D68 000C FFF8
clr.w -4(a6)         ; 02F174: 426E FFFC
movea.l -8(a6),a0    ; 02F178: 206E FFF8
move.w -4(a6),d1     ; 02F17C: 322E FFFC
ext.l d1             ; 02F180: 48C1
move.b 0(a0,d1.L),d0 ; 02F182: 1030 1800
ext.w d0             ; 02F186: 4880
move.w d0,(a7)       ; 02F188: 3E80
move.w #$3,-(a7)     ; 02F18A: 3F3C 0003
move.w #$3,-(a7)     ; 02F18E: 3F3C 0003 //Escritura por MIDI
jsr $341b2           ; 02F192: 4EB9 0003 41B2
addq.l #4,a7         ; 02F198: 588F
movea.l -8(a6),a0    ; 02F19A: 206E FFF8
move.w -4(a6),d1     ; 02F19E: 322E FFFC
ext.l d1             ; 02F1A2: 48C1
move.b 0(a0,d1.L),d0 ; 02F1A4: 1030 1800
ext.w d0             ; 02F1A8: 4880
addq.w #1,-4(a6)     ; 02F1AA: 526E FFFC
tst.b d0             ; 02F1AE: 4A00
bne.s -58 {$02F178}  ; 02F1B0: 66C6
subq.w #1,-2(a6)     ; 02F1B2: 536E FFFE
bge .l +$e {$02F1C6} ; 02F1B6: 6C00 000E
move.w $62760,d0     ; 02F1BA: 3039 0006 2760

```

Figura B.29: Fragmento descrito para la dirección 02F18E

MIDI. Después comprueba el valor en la dirección de memoria \$38d14 la cual contiene el valor de D0, si el valor es positivo o 0, salta a \$02F40C, si no continua la ejecución, mueve #-1 a D0 y salta a \$02F6C8.

```

movea.l -6(a6),a0      ; 02F27C: 206E FFFA
move.b (a0),d0         ; 02F280: 1010
ext.w d0               ; 02F282: 4880
move.w d0,(a7)         ; 02F284: 3E80
move.w #$2,-(a7)       ; 02F286: 3F3C 0002
move.w #$3,-(a7)       ; 02F28A: 3F3C 0003 //Mostrar por pantalla
jsr $341b2             ; 02F28E: 4EB9 0003 41B2
addq.l #4,a7           ; 02F294: 588F
addq.l #1,-6(a6)       ; 02F296: 52AE FFFA
movea.l -6(a6),a0      ; 02F29A: 206E FFFA
tst.b (a0)             ; 02F29E: 4A10
bne.s -38 {$02F27C}    ; 02F2A0: 66DA
bra .l +$b6 {$02F35A} ; 02F2A2: 6000 00B6

```

Figura B.30: Fragmento descrito para la dirección 02F28A

```

movea.l -6(a6),a0      ; 02F304: 206E FFFA
move.b (a0),d0         ; 02F308: 1010
ext.w d0               ; 02F30A: 4880
move.w d0,(a7)         ; 02F30C: 3E80
move.w #$2,-(a7)       ; 02F30E: 3F3C 0002
move.w #$3,-(a7)       ; 02F312: 3F3C 0003 //Mostrar por pantalla
jsr $341b2             ; 02F316: 4EB9 0003 41B2
addq.l #4,a7           ; 02F31C: 588F
addq.l #1,-6(a6)       ; 02F31E: 52AE FFFA
movea.l -6(a6),a0      ; 02F322: 206E FFFA
tst.b (a0)             ; 02F326: 4A10
bne.s -38 {$02F304}    ; 02F328: 66DA
move.w 8(a6),(a7)      ; 02F32A: 3EAE 0008
addq.w #1,(a7)         ; 02F32E: 5257
jsr $1cle4             ; 02F330: 4EB9 0001 C1E4
bra .l +$22 {$02F35A}; 02F336: 6000 0022

```

Figura B.31: Fragmento descrito para la dirección 02F30E

```

move.w #$3,(a7)        ; 02F366: 3EBC 0003
move.w #$2,-(a7)       ; 02F36A: 3F3C 0002 //Lectura por MIDI
jsr $341b2             ; 02F36E: 4EB9 0003 41B2
addq.l #2,a7           ; 02F374: 548F
and.l #$ff,d0          ; 02F376: C0BC 0000 00FF
move.w #$3,(a7)        ; 02F37C: 3EBC 0003
move.w #$1,-(a7)       ; 02F380: 3F3C 0001 //Comprabacion de MIDI
jsr $341b2             ; 02F384: 4EB9 0003 41B2
addq.l #2,a7           ; 02F38A: 548F
tst.l d0               ; 02F38C: 4A80
bne.s -42 {$02F366}    ; 02F38E: 66D6
move.w #$83,(a7)       ; 02F390: 3EBC 0083
move.w #$3,-(a7)       ; 02F394: 3F3C 0003
move.w #$3,-(a7)       ; 02F398: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02F39C: 4EB9 0003 41B2
addq.l #4,a7           ; 02F3A2: 588F
move.w #$14,(a7)       ; 02F3A4: 3EBC 0014
jsr $188f0             ; 02F3A8: 4EB9 0001 88F0
move.w d0,-6(a6)       ; 02F3AE: 3D40 FFFA
tst.w -6(a6)           ; 02F3B2: 4A6E FFFA
bge .l +$8 {$02F3C0}   ; 02F3B6: 6C00 0008
moveq #-1,d0           ; 02F3BA: 70FF
bra .l +$30a {$02F6C8} ; 02F3BC: 6000 030A

```

Figura B.32: Fragmento descrito para la dirección 02F36A

- 02F40C: Escritura por MIDI. Carga de la memoria \$39334 directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F442, si no lo es, mueve -1 a D0 y salta a \$02F6C8 (ver Figura B.33).
- 02F44C: Escritura por MIDI. Carga de la memoria \$4129c directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F478, si no lo es,

mueve -1 a D0 y salta a \$02F6C8.

- 02F482: Escritura por MIDI. Carga de la memoria \$62730 directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F4E4, si no lo es, mueve -1 a D0 y salta a \$02F6C8.

```

move.w $38d14, (a7)      ; 02F3D6: 3EB9 0003 8D14
move.w #$3, -(a7)        ; 02F3DC: 3F3C 0003
move.w #$3, -(a7)        ; 02F3E0: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F3E4: 4EB9 0003 41B2
addq.l #4, a7            ; 02F3EA: 588F
move.w #$14, (a7)        ; 02F3EC: 3EBC 0014
jsr $188f0               ; 02F3F0: 4EB9 0001 88F0
move.w d0, $38d14        ; 02F3F6: 33C0 0003 8D14
tst.w $38d14             ; 02F3FC: 4A79 0003 8D14
bge .l +$8 {$02F40C}     ; 02F402: 6C00 0008
moveq #-1, d0            ; 02F406: 70FF
bra .l +$2be {$02F6C8}   ; 02F408: 6000 02BE
move.w $39334, (a7)      ; 02F40C: 3EB9 0003 9334
move.w #$3, -(a7)        ; 02F412: 3F3C 0003
move.w #$3, -(a7)        ; 02F416: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F41A: 4EB9 0003 41B2
addq.l #4, a7            ; 02F420: 588F
move.w #$14, (a7)        ; 02F422: 3EBC 0014
jsr $188f0               ; 02F426: 4EB9 0001 88F0
move.w d0, $39334        ; 02F42C: 33C0 0003 9334
tst.w $39334             ; 02F432: 4A79 0003 9334
bge .l +$8 {$02F442}     ; 02F438: 6C00 0008
moveq #-1, d0            ; 02F43C: 70FF
bra .l +$288 {$02F6C8}   ; 02F43E: 6000 0288
move.w $4129c, (a7)      ; 02F442: 3EB9 0004 129C
move.w #$3, -(a7)        ; 02F448: 3F3C 0003
move.w #$3, -(a7)        ; 02F44C: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F450: 4EB9 0003 41B2
addq.l #4, a7            ; 02F456: 588F
move.w #$14, (a7)        ; 02F458: 3EBC 0014
jsr $188f0               ; 02F45C: 4EB9 0001 88F0
move.w d0, $4129c        ; 02F462: 33C0 0004 129C
tst.w $4129c             ; 02F468: 4A79 0004 129C
bge .l +$8 {$02F478}     ; 02F46E: 6C00 0008
moveq #-1, d0            ; 02F472: 70FF
bra .l +$252 {$02F6C8}   ; 02F474: 6000 0252
move.w $62730, (a7)      ; 02F478: 3EB9 0006 2730
move.w #$3, -(a7)        ; 02F47E: 3F3C 0003
move.w #$3, -(a7)        ; 02F482: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F486: 4EB9 0003 41B2
addq.l #4, a7            ; 02F48C: 588F
move.w #$14, (a7)        ; 02F48E: 3EBC 0014
jsr $188f0               ; 02F492: 4EB9 0001 88F0
move.w d0, $62730        ; 02F498: 33C0 0006 2730
tst.w $62730             ; 02F49E: 4A79 0006 2730
bge .l +$8 {$02F4AE}     ; 02F4A4: 6C00 0008
moveq #-1, d0            ; 02F4A8: 70FF
bra .l +$21c {$02F6C8}   ; 02F4AA: 6000 021C

```

Figura B.33: Fragmento descrito para la dirección 02F40C

- 02F4B8: Escritura por MIDI. Carga de la memoria \$637d0 directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F4E4, si no lo es, mueve -1 a D0 y salta a \$02F6C8 (ver Figura B.34).
- 02F4EE: Escritura por MIDI. Carga de la memoria \$6653e directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F516, si no lo es, mueve -1 a D0 y salta a \$02F6C8.

- 02F520: Escritura por MIDI. Carga de la memoria \$66540 directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F548, si no lo es, mueve -1 a D0 y salta a \$02F6C8.
- 02F552: Escritura por MIDI. Carga de la memoria \$66542 directamente sobre A7 y lo escribe por MIDI. Comprueba si es mayor que 0, si lo es salta a \$02F57A, si no lo es, mueve -1 a D0 y salta a \$02F6C8.

```

move.w $637d0, (a7)      ; 02F4AE: 3EB9 0006 37D0
move.w #$3, -(a7)       ; 02F4B4: 3F3C 0003
move.w #$3, -(a7)       ; 02F4B8: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02F4BC: 4EB9 0003 41B2
addq.l #4, a7           ; 02F4C2: 588F
move.w #$14, (a7)       ; 02F4C4: 3EBC 0014
jsr $188f0              ; 02F4C8: 4EB9 0001 88F0
move.w d0, $637d0       ; 02F4CE: 33C0 0006 37D0
tst.w $637d0            ; 02F4D4: 4A79 0006 37D0
bge .l +$8 {$02F4E4}    ; 02F4DA: 6C00 0008
moveq #-1, d0           ; 02F4DE: 70FF
bra .l +$1e6 {$02F6C8}; 02F4E0: 6000 01E6
move.w $6653e, (a7)     ; 02F4E4: 3EB9 0006 653E
move.w #$3, -(a7)       ; 02F4EA: 3F3C 0003
move.w #$3, -(a7)       ; 02F4EE: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02F4F2: 4EB9 0003 41B2
addq.l #4, a7           ; 02F4F8: 588F
move.w #$14, (a7)       ; 02F4FA: 3EBC 0014
jsr $188f0              ; 02F4FE: 4EB9 0001 88F0
move.w d0, -6(a6)       ; 02F504: 3D40 FFFA
tst.w -6(a6)            ; 02F508: 4A6E FFFA
bge .l +$8 {$02F516}    ; 02F50C: 6C00 0008
moveq #-1, d0           ; 02F510: 70FF
bra .l +$1b4 {$02F6C8}; 02F512: 6000 01B4
move.w $66540, (a7)     ; 02F516: 3EB9 0006 6540
move.w #$3, -(a7)       ; 02F51C: 3F3C 0003
move.w #$3, -(a7)       ; 02F520: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02F524: 4EB9 0003 41B2
addq.l #4, a7           ; 02F52A: 588F
move.w #$14, (a7)       ; 02F52C: 3EBC 0014
jsr $188f0              ; 02F530: 4EB9 0001 88F0
move.w d0, -6(a6)       ; 02F536: 3D40 FFFA
tst.w -6(a6)            ; 02F53A: 4A6E FFFA
bge .l +$8 {$02F548}    ; 02F53E: 6C00 0008
moveq #-1, d0           ; 02F542: 70FF
bra .l +$182 {$02F6C8}; 02F544: 6000 0182
move.w $66542, (a7)     ; 02F548: 3EB9 0006 6542
move.w #$3, -(a7)       ; 02F54E: 3F3C 0003
move.w #$3, -(a7)       ; 02F552: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02F556: 4EB9 0003 41B2
addq.l #4, a7           ; 02F55C: 588F
move.w #$14, (a7)       ; 02F55E: 3EBC 0014
jsr $188f0              ; 02F562: 4EB9 0001 88F0
move.w d0, -6(a6)       ; 02F568: 3D40 FFFA
tst.w -6(a6)            ; 02F56C: 4A6E FFFA
bge .l +$8 {$02F57A}    ; 02F570: 6C00 0008
moveq #-1, d0           ; 02F574: 70FF
bra .l +$150 {$02F6C8}; 02F576: 6000 0150

```

Figura B.34: Fragmento descrito para la dirección 02F4B8

- 02F59E: Escritura por MIDI. Mueve (movea) \$-2(a6) a A0, después mueve #\$62788 a D1, y realiza un move 0(a0,d1.L),d0, sobre D0, carga D0 en A7, y lo escribe por MIDI, compara #\$32 con -2(a6) si el contenido de -2(a6) es menos que 32 salta a \$02F5C2, si no mueve d0 a -6(a6) y continua en \$02F5C2. [\$02F5C2] -¿Realiza un tst sobre -6(a6), si es mayor que 0 salta a \$02F5D0, si no mueve -1 a D0 y salta a \$02F6C8. [\$02F5D0] -¿Suma 1 a -2(a6), lo almacena en D0, y compara con -4(a6), si -4(a6) es menor que

-2(a6) salta a \$02F588 (bucle de escritura \$02F59E), si no lo es limpia el contenido de -2(a6) y salta a \$02F5F8, donde se compara -2(a6) con #\$32, si 32 es menos se salta a \$02F5E6 (subrutina \$188f0), si no, se realiza un tst de -6(a6) si es mayor que 0 se salta a \$02F60E, si no se mueve -1 a D0 y se salta a \$02F6C8 (ver Figura B.35).

```

movea.w -2(a6),a0      ; 02F588: 306E FFFE
move.l  #$62788,d1     ; 02F58C: 223C 0006 2788
move.b  0(a0,d1.L),d0  ; 02F592: 1030 1800
ext.w   d0             ; 02F596: 4880
move.w  d0,(a7)        ; 02F598: 3E80
move.w  #$3,-(a7)      ; 02F59A: 3F3C 0003
move.w  #$3,-(a7)      ; 02F59E: 3F3C 0003 //Escritura por MIDI
jsr     $341b2         ; 02F5A2: 4EB9 0003 41B2
addq.l  #4,a7          ; 02F5A8: 588F
cmpi.w  #$32,-2(a6)    ; 02F5AA: 0C6E 0032 FFFE
blt     .l +$10 {$02F5C2} ; 02F5B0: 6D00 0010
move.w  #$14,(a7)      ; 02F5B4: 3EBC 0014
jsr     $188f0         ; 02F5B8: 4EB9 0001 88F0
move.w  d0,-6(a6)      ; 02F5BE: 3D40 FFFA
tst.w   -6(a6)         ; 02F5C2: 4A6E FFFA
bge     .l +$8 {$02F5D0} ; 02F5C6: 6C00 0008
moveq   #-1,d0         ; 02F5CA: 70FF
bra     .l +$fa {$02F6C8} ; 02F5CC: 6000 00FA

```

Figura B.35: Fragmento descrito para la dirección 02F59E

- 02F618: Escritura por MIDI. Carga de la memoria \$38d10 directamente sobre A7 y lo escribe por MIDI. Hace un clear de -2(a6) y salta incondicionalmente a \$02F654 (ver Figura B.36).
- 02F644: Escritura por MIDI. Mueve -2(a6) a D0, realiza un muls (signed multiplication) de #\$84 sobre D0, movea de D0 sobre A0, move de #\$4129e sobre D1, y un move de 24(a0,d1.L) sobre A7 y lo escribe por MIDI. Suma 1 a -2(a6) y lo compara con #\$10, si -2(a6) es menor que 10, salta a \$02F62C, si no lo es se hace un clear de -2(a6) y se salta a \$02F676.
- 02F696: Escritura por MIDI. Mueve de memoria \$62768 a A7 y lo escribe por MIDI. Realiza un tst sobre \$62768, si es mayor que 0, salta a \$02F6C2, si no mueve -1 a D0 y salta a \$02F6C8. [\$02F6C2] -¿Realiza un clear sobre D0, y salta a \$02F6C8 (sale de la subrutina).
- 02F704: Escritura por MIDI . Escribe #\$83 (MIDI-SEND-DATA) en MIDI . Después mueve D0 sobre \$38d14, realiza un tst sobre el, si es mayor que 0 salta a \$02F736, si no mueve -1 sobre D0 y salta a \$02FA0A (ver Figura B.37).
- 02F740: Escritura por MIDI. Mueve de la dirección de memoria \$38d14 a A7 y lo escribe por MIDI. Después mueve D0 a \$39334 y comprueba si es mayor de 0, si lo es salta a \$02F76C, si no mueve -1 a D0 y salta a \$02FA0A.
- 02F776: Escritura por MIDI. Mueve de la dirección de memoria \$39334 a A7 y lo escribe por MIDI. Después mueve D0 a \$39334 y comprueba si es mayor de 0, si lo es salta a \$02F76C, si no mueve -1 a D0 y salta a \$02FA0A.
- 02F7AC: Escritura por MIDI. Mueve de la dirección de memoria \$4129c a A7 y lo escribe por MIDI. Después mueve D0 a \$62730 y comprueba si es mayor de 0, si lo es salta a \$02F7D8, si no mueve -1 a D0 y salta a \$02FA0A.

```

move.w $38d10, (a7)      ; 02F60E: 3EB9 0003 8D10
move.w #$3, -(a7)       ; 02F614: 3F3C 0003
move.w #$3, -(a7)       ; 02F618: 3F3C 0003 //Escritura por MIDI
jsr $341b2              ; 02F61C: 4EB9 0003 41B2
addq.l #4, a7           ; 02F622: 588F
clr.w -2(a6)            ; 02F624: 426E FFFE
bra .l +$2a {$02F654}; 02F628: 6000 002A
move.w -2(a6), d0        ; 02F62C: 302E FFFE
muls #$84, d0           ; 02F630: C1FC 0084
movea.l d0, a0          ; 02F634: 2040
move.l #$4129e, d1      ; 02F636: 223C 0004 129E
move.w 24(a0, d1.L), (a7); 02F63C: 3EB0 1818
move.w #$3, -(a7)       ; 02F640: 3F3C 0003
move.w #$3, -(a7)       ; 02F644: 3F3C 0003 //Escritura en MIDI
jsr $341b2              ; 02F648: 4EB9 0003 41B2
addq.l #4, a7           ; 02F64E: 588F
addq.w #1, -2(a6)       ; 02F650: 526E FFFE
cmpi.w #$10, -2(a6)     ; 02F654: 0C6E 0010 FFFE
blt.s -48 {$02F62C}     ; 02F65A: 6DD0
clr.w -2(a6)            ; 02F65C: 426E FFFE
bra .l +$14 {$02F676}; 02F660: 6000 0014
move.w #$14, (a7)       ; 02F664: 3EBC 0014
jsr $188f0              ; 02F668: 4EB9 0001 88F0
move.w d0, -6(a6)       ; 02F66E: 3D40 FFFA
addq.w #1, -2(a6)       ; 02F672: 526E FFFE
cmpi.w #$10, -2(a6)     ; 02F676: 0C6E 0010 FFFE
ble.s -26 {$02F664}     ; 02F67C: 6FE6
tst.w -6(a6)            ; 02F67E: 4A6E FFFA
bge .l +$8 {$02F68C}    ; 02F682: 6C00 0008
moveq #-1, d0           ; 02F686: 70FF
bra .l +$3e {$02F6C8}; 02F688: 6000 003E
move.w $62768, (a7)     ; 02F68C: 3EB9 0006 2768
move.w #$3, -(a7)       ; 02F692: 3F3C 0003
move.w #$3, -(a7)       ; 02F696: 3F3C 0003 //Escritura en MIDI
jsr $341b2              ; 02F69A: 4EB9 0003 41B2
addq.l #4, a7           ; 02F6A0: 588F
move.w #$14, (a7)       ; 02F6A2: 3EBC 0014
jsr $188f0              ; 02F6A6: 4EB9 0001 88F0
move.w d0, $62768       ; 02F6AC: 33C0 0006 2768
tst.w $62768            ; 02F6B2: 4A79 0006 2768
bge .l +$8 {$02F6C2}    ; 02F6B8: 6C00 0008
moveq #-1, d0           ; 02F6BC: 70FF
bra .l +$8 {$02F6C8}    ; 02F6BE: 6000 0008

```

Figura B.36: Fragmento descrito para la dirección 02F618

- 02F7E2: Escritura por MIDI. Mueve de la dirección de memoria \$62730 a A7 y lo escribe por MIDI. Después mueve D0 a \$637d0 y comprueba si es mayor de 0, si lo es salta a \$02F80E, si no mueve -1 a D0 y salta a \$02FA0A (ver Figura B.38).
- 02F818: Escritura por MIDI. Mueve de la dirección de memoria \$637d0 a A7 y lo escribe por MIDI. Después mueve D0 a \$6653e y comprueba si es mayor de 0, si lo es salta a \$02F844, si no mueve -1 a D0 y salta a \$02FA0A.
- 02F84E: Escritura por MIDI. Mueve de la dirección de memoria \$6653e a A7 y lo escribe por MIDI. Después mueve D0 a \$66540 y comprueba si es mayor de 0, si lo es salta a \$02F87A, si no mueve -1 a D0 y salta a \$02FA0A.
- 02F884: Escritura por MIDI. Mueve de la dirección de memoria \$66540 a A7 y lo escribe por MIDI. Después mueve D0 a \$66542 y comprueba si es mayor de 0, si lo es salta a \$02F8B0, si no mueve -1 a D0 y salta a \$02FA0A.
- 02F8BA: Escritura por MIDI. Mueve de la dirección de memoria \$66542 a A7 y lo

```

move.w #$83, (a7)      ; 02F6FC: 3EBC 0083
move.w #$3, -(a7)      ; 02F700: 3F3C 0003
move.w #$3, -(a7)      ; 02F704: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02F708: 4EB9 0003 41B2
addq.l #4, a7          ; 02F70E: 588F
jsr $2f080             ; 02F710: 4EB9 0002 F080
move.w #$14, (a7)      ; 02F716: 3EBC 0014
jsr $188f0             ; 02F71A: 4EB9 0001 88F0
move.w d0, $38d14      ; 02F720: 33C0 0003 8D14
tst.w $38d14           ; 02F726: 4A79 0003 8D14
bge .l +$8 {$02F736}   ; 02F72C: 6C00 0008
moveq #-1, d0          ; 02F730: 70FF
bra .l +$2d6 {$02FA0A}; 02F732: 6000 02D6
move.w $38d14, (a7)    ; 02F736: 3EB9 0003 8D14
move.w #$3, -(a7)      ; 02F73C: 3F3C 0003
move.w #$3, -(a7)      ; 02F740: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02F744: 4EB9 0003 41B2
addq.l #4, a7          ; 02F74A: 588F
move.w #$14, (a7)      ; 02F74C: 3EBC 0014
jsr $188f0             ; 02F750: 4EB9 0001 88F0
move.w d0, $39334      ; 02F756: 33C0 0003 9334
tst.w $39334           ; 02F75C: 4A79 0003 9334
bge .l +$8 {$02F76C}   ; 02F762: 6C00 0008
moveq #-1, d0          ; 02F766: 70FF
bra .l +$2a0 {$02FA0A}; 02F768: 6000 02A0
move.w $39334, (a7)    ; 02F76C: 3EB9 0003 9334
move.w #$3, -(a7)      ; 02F772: 3F3C 0003
move.w #$3, -(a7)      ; 02F776: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02F77A: 4EB9 0003 41B2
addq.l #4, a7          ; 02F780: 588F
move.w #$14, (a7)      ; 02F782: 3EBC 0014
jsr $188f0             ; 02F786: 4EB9 0001 88F0
move.w d0, $4129c      ; 02F78C: 33C0 0004 129C
tst.w $4129c           ; 02F792: 4A79 0004 129C
bge .l +$8 {$02F7A2}   ; 02F798: 6C00 0008
moveq #-1, d0          ; 02F79C: 70FF
bra .l +$26a {$02FA0A}; 02F79E: 6000 026A
move.w $4129c, (a7)    ; 02F7A2: 3EB9 0004 129C
move.w #$3, -(a7)      ; 02F7A8: 3F3C 0003
move.w #$3, -(a7)      ; 02F7AC: 3F3C 0003 //Escritura por MIDI
jsr $341b2             ; 02F7B0: 4EB9 0003 41B2
addq.l #4, a7          ; 02F7B6: 588F
move.w #$14, (a7)      ; 02F7B8: 3EBC 0014
jsr $188f0             ; 02F7BC: 4EB9 0001 88F0
move.w d0, $62730      ; 02F7C2: 33C0 0006 2730
tst.w $62730           ; 02F7C8: 4A79 0006 2730
bge .l +$8 {$02F7D8}   ; 02F7CE: 6C00 0008
moveq #-1, d0          ; 02F7D2: 70FF
bra .l +$234 {$02FA0A}; 02F7D4: 6000 0234

```

Figura B.37: Fragmento descrito para la dirección 02F704

escribe por MIDI. Después mueve #\$1000 a -4(a6) y limpia el contenido de -4(a6), y salta de forma incondicional a \$02F918 (ver Figura B.39).

- 02F8F8: Escritura por MIDI. Mueve D0 a -6(a6), realiza tst sobre -6(a6), si es mayor que 0 salta a \$02F8F0, si no, mueve -1 a D0 y salta a \$02FA0A. [02F8F0] -¿Mueve el contenido de -6(a6) a A7 y lo escribe por MIDI. Después mueve el contenido de -6(a6) a D0, el de -2(a6) a A1, suma(adda) #\$62788 a A1, y mueve D0 a A1, y suma#1 a -2(a6). Finalmente mueve el contenido de -2(a6) a D0, y compara el contenido de -4(a6) con D0, si D0 es menor que -4(a6) salta a \$02F8D4, si no continua su ejecución, llama a la subrutina \$188f0, mueve D0 a \$38d10, realiza tst, si es mayor que 0 salta a \$02F942, si no, mueve #-1 a D0 y salta a \$02FA0A.

```

move.w $62730, (a7)      ; 02F7D8: 3EB9 0006 2730
move.w #$3, -(a7)        ; 02F7DE: 3F3C 0003
move.w #$3, -(a7)        ; 02F7E2: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F7E6: 4EB9 0003 41B2
addq.l #4, a7            ; 02F7EC: 588F
move.w #$14, (a7)        ; 02F7EE: 3EBC 0014
jsr $188f0               ; 02F7F2: 4EB9 0001 88F0
move.w d0, $637d0        ; 02F7F8: 33C0 0006 37D0
tst.w $637d0             ; 02F7FE: 4A79 0006 37D0
bge .l +$8 {$02F80E}     ; 02F804: 6C00 0008
moveq #-1, d0            ; 02F808: 70FF
bra .l +$1fe {$02FA0A}   ; 02F80A: 6000 01FE
move.w $637d0, (a7)      ; 02F80E: 3EB9 0006 37D0
move.w #$3, -(a7)        ; 02F814: 3F3C 0003
move.w #$3, -(a7)        ; 02F818: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F81C: 4EB9 0003 41B2
addq.l #4, a7            ; 02F822: 588F
move.w #$14, (a7)        ; 02F824: 3EBC 0014
jsr $188f0               ; 02F828: 4EB9 0001 88F0
move.w d0, $6653e        ; 02F82E: 33C0 0006 653E
tst.w $6653e             ; 02F834: 4A79 0006 653E
bge .l +$8 {$02F844}     ; 02F83A: 6C00 0008
moveq #-1, d0            ; 02F83E: 70FF
bra .l +$1c8 {$02FA0A}   ; 02F840: 6000 01C8
move.w $6653e, (a7)      ; 02F844: 3EB9 0006 653E
move.w #$3, -(a7)        ; 02F84A: 3F3C 0003
move.w #$3, -(a7)        ; 02F84E: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F852: 4EB9 0003 41B2
addq.l #4, a7            ; 02F858: 588F
move.w #$14, (a7)        ; 02F85A: 3EBC 0014
jsr $188f0               ; 02F85E: 4EB9 0001 88F0
move.w d0, $66540        ; 02F864: 33C0 0006 6540
tst.w $66540             ; 02F86A: 4A79 0006 6540
bge .l +$8 {$02F87A}     ; 02F870: 6C00 0008
moveq #-1, d0            ; 02F874: 70FF
bra .l +$192 {$02FA0A}   ; 02F876: 6000 0192
move.w $66540, (a7)      ; 02F87A: 3EB9 0006 6540
move.w #$3, -(a7)        ; 02F880: 3F3C 0003
move.w #$3, -(a7)        ; 02F884: 3F3C 0003 //Escritura por MIDI
jsr $341b2               ; 02F888: 4EB9 0003 41B2
addq.l #4, a7            ; 02F88E: 588F
move.w #$14, (a7)        ; 02F890: 3EBC 0014
jsr $188f0               ; 02F894: 4EB9 0001 88F0
move.w d0, $66542        ; 02F89A: 33C0 0006 6542
tst.w $66542             ; 02F8A0: 4A79 0006 6542
bge .l +$8 {$02F8B0}     ; 02F8A6: 6C00 0008
moveq #-1, d0            ; 02F8AA: 70FF
bra .l +$15c {$02FA0A}   ; 02F8AC: 6000 015C

```

Figura B.38: Fragmento descrito para la dirección 02F7E2

- 02F94C: Escritura por MIDI. Mueve de la dirección de memoria \$38d10 a A7 y lo escribe por MIDI. Después limpia la dirección -2(a6) y salta a \$02F9C6 incondicionalmente.
- 02F9B6: Escritura por MIDI. Mueve -2(a6) a D0 y multiplica D0 por #\$84, y mueve D0 a A0. Después mueve #\$4129e a D1, realiza un move.w 24(a0,d1.L),(a7). Y escribe por MIDI. Después suma #1 a -2(a6), y lo compara con #10, si el contenido de -2(a6) es menor de #10 salta a \$02F960, si no continua con el código correspondiente a la siguiente escritura (\$02F9F8). (ver Figura B.40).
- 02F9F8: Escritura por MIDI. Llama a la subrutina \$188f0, mueve D0 a \$62768, y realiza tst sobre \$62768, si es mayor que 0 salta a \$02F9EE, si no mueve -1 a D0 y salta a \$02FA0A. [\$02F9EE]. Mueve el contenido de \$62768 a A7 y lo escribe por MIDI. Limpia el contenido de D0 y salta a \$02FA0A.

move.w \$66542, (a7)	; 02F8B0: 3EB9 0006 6542
move.w #\$3, -(a7)	; 02F8B6: 3F3C 0003
move.w #\$3, -(a7)	; 02F8BA: 3F3C 0003 //Escritura por MIDI
jsr \$341b2	; 02F8BE: 4EB9 0003 41B2
addq.l #4, a7	; 02F8C4: 588F
move.w #\$1000, -4(a6)	; 02F8C6: 3D7C 1000 FFFC
clr.w -2(a6)	; 02F8CC: 426E FFFE
bra .l +\$46 {\$02F918}	; 02F8D0: 6000 0046
move.w #\$14, (a7)	; 02F8D4: 3EBC 0014
jsr \$188f0	; 02F8D8: 4EB9 0001 88F0
move.w d0, -6(a6)	; 02F8DE: 3D40 FFFA
tst.w -6(a6)	; 02F8E2: 4A6E FFFA
bge .l +\$8 {\$02F8F0}	; 02F8E6: 6C00 0008
moveq #-1, d0	; 02F8EA: 70FF
bra .l +\$11c {\$02FA0A}	; 02F8EC: 6000 011C
move.w -6(a6), (a7)	; 02F8F0: 3EAE FFFA
move.w #\$3, -(a7)	; 02F8F4: 3F3C 0003
move.w #\$3, -(a7)	; 02F8F8: 3F3C 0003 //Escritura por MIDI
jsr \$341b2	; 02F8FC: 4EB9 0003 41B2
addq.l #4, a7	; 02F902: 588F
move.w -6(a6), d0	; 02F904: 302E FFFA
movea.w -2(a6), a1	; 02F908: 326E FFFE
adda.l #\$62788, a1	; 02F90C: D3FC 0006 2788
move.b d0, (a1)	; 02F912: 1280
addq.w #1, -2(a6)	; 02F914: 526E FFFE
move.w -2(a6), d0	; 02F918: 302E FFFE
cmp.w -4(a6), d0	; 02F91C: B06E FFFC
blt.s -78 {\$02F8D4}	; 02F920: 6DB2
move.w #\$14, (a7)	; 02F922: 3EBC 0014
jsr \$188f0	; 02F926: 4EB9 0001 88F0
move.w d0, \$38d10	; 02F92C: 33C0 0003 8D10
tst.w \$38d10	; 02F932: 4A79 0003 8D10
bge .l +\$8 {\$02F942}	; 02F938: 6C00 0008
moveq #-1, d0	; 02F93C: 70FF
bra .l +\$ca {\$02FA0A}	; 02F93E: 6000 00CA
move.w \$38d10, (a7)	; 02F942: 3EB9 0003 8D10
move.w #\$3, -(a7)	; 02F948: 3F3C 0003
move.w #\$3, -(a7)	; 02F94C: 3F3C 0003 //Escritura por MIDI
jsr \$341b2	; 02F950: 4EB9 0003 41B2
addq.l #4, a7	; 02F956: 588F
clr.w -2(a6)	; 02F958: 426E FFFE
bra .l +\$68 {\$02F9C6}	; 02F95C: 6000 0068

Figura B.39: Fragmento descrito para la dirección 02F8BA

move.w -2(a6),d0	; 02F99E: 302E FFFE
muls #\$84,d0	; 02F9A2: C1FC 0084
movea.l d0,a0	; 02F9A6: 2040
move.l #\$4129e,d1	; 02F9A8: 223C 0004 129E
move.w 24(a0,d1.L),(a7)	; 02F9AE: 3EB0 1818
move.w #\$3,-(a7)	; 02F9B2: 3F3C 0003
move.w #\$3,-(a7)	; 02F9B6: 3F3C 0003 //Escritura por MIDI
jsr \$341b2	; 02F9BA: 4EB9 0003 41B2
addq.l #4,a7	; 02F9C0: 588F
addq.w #1,-2(a6)	; 02F9C2: 526E FFFE
cmpi.w #\$10,-2(a6)	; 02F9C6: 0C6E 0010 FFFE
blt.s -110 {\$02F960}	; 02F9CC: 6D92
move.w #\$14,(a7)	; 02F9CE: 3EBC 0014
jsr \$188f0	; 02F9D2: 4EB9 0001 88F0
move.w d0,\$62768	; 02F9D8: 33C0 0006 2768
tst.w \$62768	; 02F9DE: 4A79 0006 2768
bge .l +\$8 {\$02F9EE}	; 02F9E4: 6C00 0008
moveq #-1,d0	; 02F9E8: 70FF
bra .l +\$1e {\$02FA0A}	; 02F9EA: 6000 001E
move.w \$62768,(a7)	; 02F9EE: 3EB9 0006 2768
move.w #\$3,-(a7)	; 02F9F4: 3F3C 0003
move.w #\$3,-(a7)	; 02F9F8: 3F3C 0003 //Escritura por MIDI
jsr \$341b2	; 02F9FC: 4EB9 0003 41B2
addq.l #4,a7	; 02FA02: 588F
clr.w d0	; 02FA04: 4240
bra .l +\$2 {\$02FA0A}	; 02FA06: 6000 0002

Figura B.40: Fragmento descrito para la dirección 02F9B6

Anexo C

Materiales de MIDI Maze Atari ST

En este anexo se va a mostrar el material con el que fue lanzado al mercado MIDI Maze en 1987.

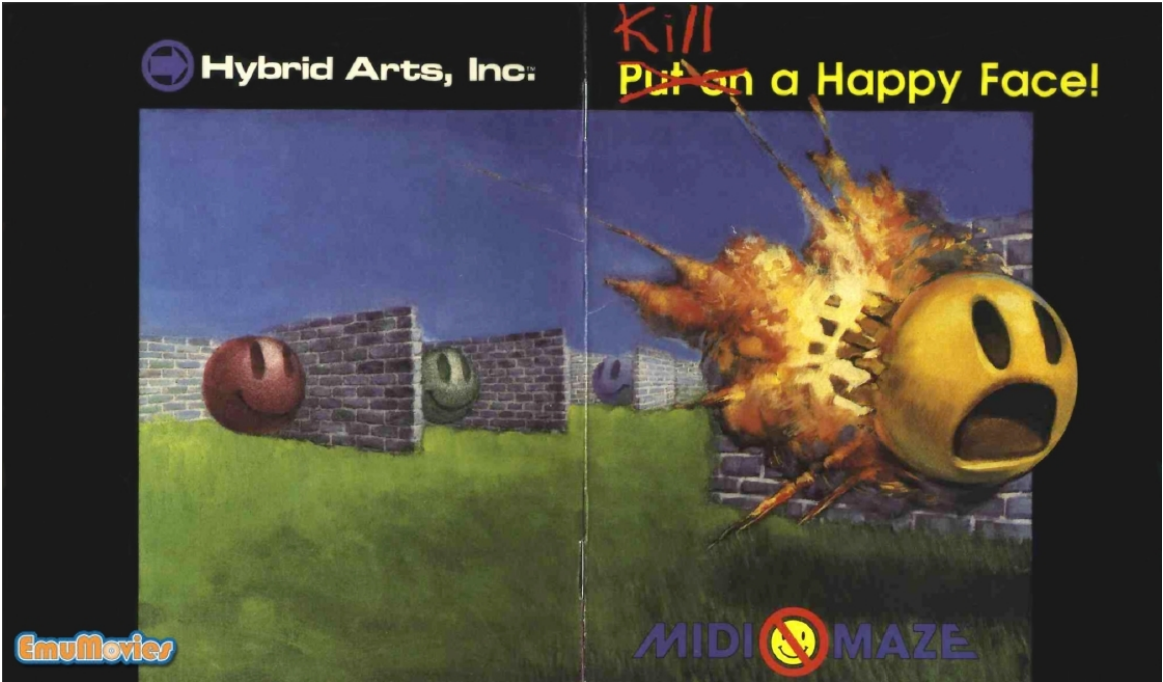


Figura C.1: Portada del manual de MIDI Maze.

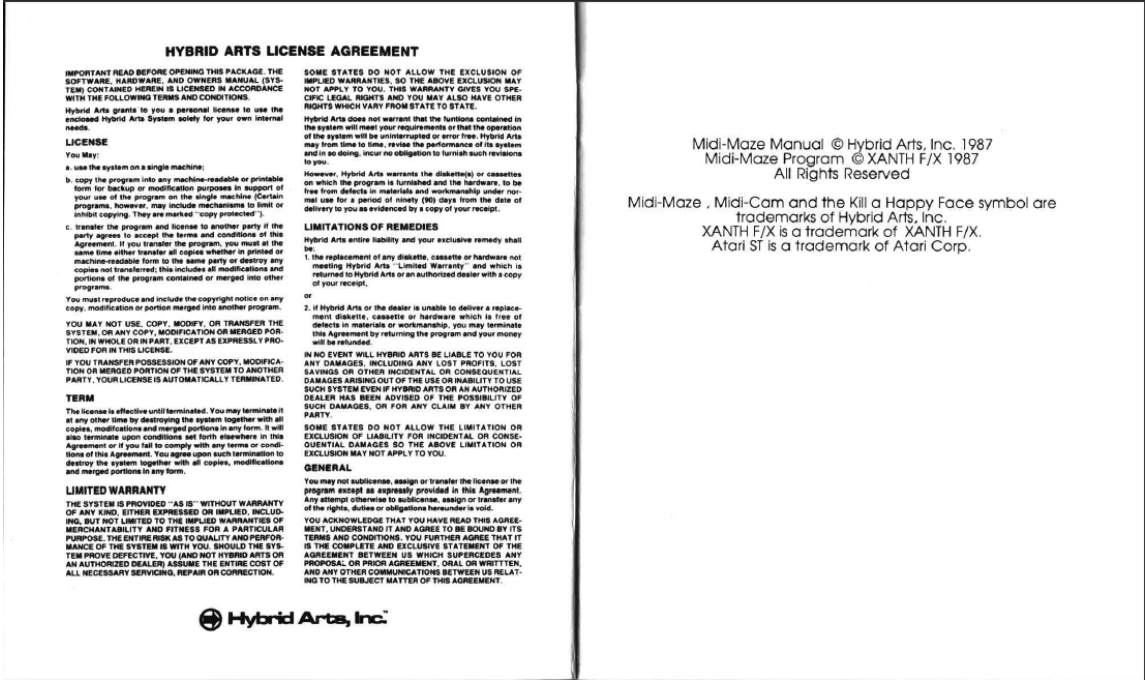


Figura C.2: Paginas 1 y 2 del manual de MIDI Maze.

CREDITS: Game Design and Programming XANTH F/X Control Panel Art LERVOLD Manual Art Vixen Cover Art Swofford Typing RAD Moose About the Authors: XANTH F/X is comprised mostly of the design team that produced "Fuji-Boink", "Shiny Bubbles" and other graphic demos on the Atari ST™ and ATARI XE series computers.	Table of contents <table><tr><td>1. A Note from XANTH F/X</td><td>page 6</td></tr><tr><td>2. Welcome to MIDI-MAZE™</td><td>page 7</td></tr><tr><td>3. Setup & Requirements</td><td>page 8</td></tr><tr><td>4. Gem Controls</td><td>page 12</td></tr><tr><td>5. Play Controls</td><td>page 13</td></tr><tr><td>6. Smiley busting made easy!</td><td>page 15</td></tr><tr><td>7. Maze construction</td><td>page 18</td></tr><tr><td>8. Midi-cam option</td><td>page 19</td></tr></table>	1. A Note from XANTH F/X	page 6	2. Welcome to MIDI-MAZE™	page 7	3. Setup & Requirements	page 8	4. Gem Controls	page 12	5. Play Controls	page 13	6. Smiley busting made easy!	page 15	7. Maze construction	page 18	8. Midi-cam option	page 19
1. A Note from XANTH F/X	page 6																
2. Welcome to MIDI-MAZE™	page 7																
3. Setup & Requirements	page 8																
4. Gem Controls	page 12																
5. Play Controls	page 13																
6. Smiley busting made easy!	page 15																
7. Maze construction	page 18																
8. Midi-cam option	page 19																

Figura C.3: Paginas 3 y 4 del manual de MIDI Maze.

1. A Note from XANTH F/X This Program is our first commercial work and it is imperative that we be successful. We have contributed many programs to the public domain and will continue to do so, IF we can make a living programming for the Atari ST. Please do not "Pirate" or accept illegal copies of this program. We intend to upgrade this program significantly, it is the first of a series of advanced graphic video games. However, if we do not meet our sales goals we will be forced to upgrade our programs on OTHER computers! Hybrid Arts, Inc. 6 MIDI-MAZE User Manual	2. Welcome to MIDI-MAZE™ Once upon a time in a measure far, far from sanity, there existed sixteen notes, or more... But in our fable only sixteen ascended the scale to jam in... MIDI-MAZE!!! We take you back to a time before chords or even polyphony. Scores of octaves came to watch the barbaric finale. The air reverberating with the chorus of "Have a Nice Day!". Scary, isn't it? Dear Readers... It was note against note, color against color. It was not Pretty! Your mission young smiley, is to defend your color, edit the "bad" notes and may the best note win! The refrain is simple: "Sixteen notes enter, One note leaves" In this program we relive the simpler time. You become a player in the maze, one on one or one on many... Enjoy!!! Hybrid Arts, Inc. 7 MIDI-MAZE User Manual
--	---

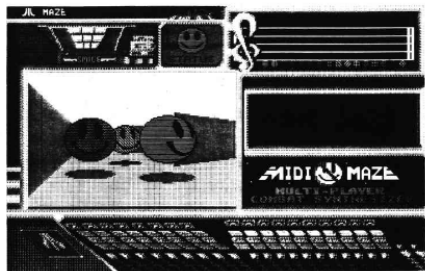
Figura C.4: Paginas 5 y 6 del manual de MIDI Maze.

3. Set-up and Requirements Midi-maze will run on any ATARI ST. One MIDI cable is needed for each computer if you are using the multiple machine option. A joystick is recommended, however, if you want use the mouse pressing the "ALT" key and the "M" key together will select mouse control. The "ALT-J" combination will select back to joystick control. Single player and Solo mode: Insert disk and turn-on ST. Midi-Maze will auto-boot. After the boot process is complete the computer will either say "master machine" or "slave machine," depending on if you have a midi cable connected from the MIDI-IN to MIDI-OUT port. If you have a midi cable connected, you should get the prompt "master machine" in that case skip the next step! If your machine says "slave machine" you need to press the "ALT" and the "S" key at the same time. Pressing the "ALT-S" key combination will activate the Solo-mode, which does not require the midi cables to be connected. Hitting the "ALT-S" combination again, returns the computer to "slave machine" mode. Multiple players and the "Master machine": All MIDI-cables should be connected, MIDI-IN to MIDI-OUT (see "MIDI-ring" below). Insert MIDI-MAZE disk into each computer one at a time and turn-on ST. All the machines will say "slave machine" except for the last machine in the sequence, which will say "Master Machine". The "Master machine" is the last computer to be activated in the "MIDI-Ring". Please note. The slave machines have no control at all, and all selections are made by the "Master machine", unless they activate the "Solo-mode" as described above. If solo-mode is in effect, they take themselves out of the MIDI-Ring and should be physically removed if the other players want to play multiple machines. Any number of players can activate the "Solo-mode" but they will all need to have at least a disk with some maze files on it. The "Master machine" may also become a "Solo machine" if desired. If you need to re-establish the MIDI-Ring use the "ALT-S" to turn the solo machines back to "Slaves". If for some reason you have turned the master into a "Slave machine" "ALT-R" will return it to "master" status. Hybrid Arts, Inc. 8 MIDI-MAZE User Manual	The MIDI-Ring: Up to 16 machines can be connected in the following manner. You will need one MIDI-cable for each machine you want to connect. Insert one end of each cable into the MIDI-IN port and the other to the MIDI-OUT port. You are creating a "MIDI-ring" you should be able to start at the first ST and follow the MIDI-cables, MIDI-IN to MIDI-OUT all the way around back to the first ST (see diagram). note: If after activating all the machines, all the computers say "slave" you probably have a MIDI cable connected wrong! Check all your connections and make sure they're hooked up correctly! When you think its hooked up right; Press ALT and the R key together on the machine you want to be the master. After you press the "ALT-R" this machine will be the "Master machine" and will need to have the maze files in the disk drive. Hybrid Arts, Inc. 9 MIDI-MAZE User Manual
---	---

Figura C.5: Paginas 7 y 8 del manual de MIDI Maze.

Viewscreen: This is your window to the maze. What you see is what you get! In the center is a tiny colored sighting bead that indicates: 1. Your color. 2. That you have a "bullet" ready and 3. The destination of the bullet. A whole new dimension to the term, "drawing a bead on someone".

Joystick control: Forward, back, right, left and fire button. Need we say more? Easy to learn, but difficult to master! Hint: Shoot anything that moves. Advance hint: don't get hit!



The "Score" score: shows each of the players Face-busting totals. The first Smiley to the top of the score wins the Set.

Happiness Quotient: Your Happiness indicator located above the viewing window, indicates how many hits your "ego" can take. A nice full smile indicates 3 hits; a straight across grin, 2 hits and a full frown only 1 hit. When wearing a frown, you really don't want to see people!

Pop chart: This is where your opponents deflated egos are kept. When the box is filled (10 smileybuster symbols), you win! You can tell at a glance how many and who, you have been "editing".



Figura C.6: Paginas 9 y 10 del manual de MIDI Maze.

4. GEM Controls

Starting from the top: The GEM pull-downs are of the standard format. All Game controls are located in the "MAZE" pull-down.

'Load...': This is where you load a maze into the program. This function automatically looks for a file with a ".MAZ" extension.

'Reset score': To reset the cumulative score that keeps track of each player wins, since power up or the last "Reset score".

'Names': This feature will allow all human players to name their Smiley.

'Play..I': By selecting this window or by hitting return once, you can now select any of the Midi-Maze options. Please note that if you wish to play the same options again, all you have to do is hit return!

'Quit': Rather self-explanatory, this command causes your machine to reboot.

5. Play controls

'# Machine/s on-line': this indicated the number of computers that are networked together via the MIDI port. (Not including MidiCam machines).

'Reload Time': The default setting of "Fast" gives the maximum setting for fair play. The "Slow" setting gives a more patient and stalking game, forcing the players to choose their shots and be more accurate!

'Regen Time': Regen time is the time it takes for the players to recover from a hit. The "Slow" option means you stay weaker for a good long time!

'Revive Time': The amount of time you stay dead and staring at the opponents "Have a nice day" is regulated by this switch.

'1 Life', '2 Lives' and '3 Lives': settings decide if you come back to life with a Frown, Grin or Smile, respectively.

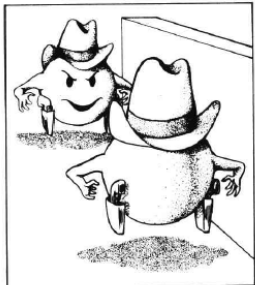
Figura C.7: Paginas 11 y 12 del manual de MIDI Maze.

Drone Controls: Proof positive that computers are very fast morons, the first drone setting "Very Dumb" provides "Target" class drones, "Plain Dumb" or "Standard Drones" at least defend themselves and "Not so Dumb" provides "Ninja" class drones that are very sneaky and fast. The computer opponents are usually no match for a large contingent of humans but will devastate small groups or a single player. Vary the game difficulty by trying various maze and drone combinations.

'Singles' play is the standard setting. However, if you have more players and would like to play teams, the "Teams" setting will give you a selection window that you can assign any player or drone to any one of 4 teams: yellow, blue, orange or green. Note: You can have drones on human teams, if you shoot a drone teammate they may shoot you back! If the drones are playing on a team, they are more difficult to beat since their scores are cumulative. Also you can shoot members of your own team, they will be weakened, but you can't kill them. If "Friendly Fire" is selected your shots, will not hurt your teammates.

6. Smiley Busting Made Easy

Welcome to the Midi-Maze Training Grounds! Here we will learn how to properly "Bust Face".



Standard Tactics:

The "Face-Off": Center the "bad note" on your screen, Either head toward him firing or sit back and lob your shots at him till he pops. Advantage: This is the easiest way to aim since you are traveling in the direction you are firing, and the target usually gets easier to hit as you get closer. Disadvantage: In most cases, if you're both facing each other it's usually a shooting match; the person who has a higher initial "Happiness Quotient" usually wins unless, you're good at "evasive boogie-ing".

Figura C.8: Paginas 13 y 14 del manual de MIDI Maze.

The "Run-away" Technique:

If you have someone behind you, its a good plan to beat feet out of there since you probably figured they're behind because you just got shot! If they're behind you, running is the best tactic unless they're really weak, because if you turn around, they'll probably hit you again before you can shoot them. While running, "Evasive boogie-ing" from side to side makes you a lot harder to hit than just running straight down the hallways! Also consecutive right or left turns, in rapid succession, makes it hard for a pursuing Smiley to "draw a bead" on you! Ideally you want to get enough space or time to turn around safely.

"Reverse Cycle-ology":

Remember, you can move and fire backwards as easily as forward. There are a few advantages traveling backward. When firing backwards, you will notice that your shots will hit a pursuer about twice as quickly as his shots will reach you because he's running straight into your fire, while you are backing away from him! Nifty huh? Also, if he's following you, you know where he's going (most of the time). Of course, there are a few disadvantages too, since you aren't watching where you're going, you might end up in a dead end, or even worse on someone else's score.

"Sneaky Smiley" behavior:

You may have noticed after playing the game, "back-bopping" is the safest and quickest way to advance the smiley ladder of fame. Laying in wait for someone to just float in front of your sights, or shooting smileys as they are still weak from regeneration or another duel are questionable tactics. Sinking around corners and sneaking up on the unwary opponent from behind, though not the most ethical behavior, works as well in the game as it does in real life. We encourage you to "fight like a smiley" but if your "Happiness Quotient" is low, we recommend that you find a safe place to hide or refrain from fighting full strength Smileys.

Advanced Tactics:

Here are some maneuvers that you might try if you're hard pressed.

The moving 180° or 360° turn, while running down a long corridor or one with a turn-off a straight stretch, do a forward right turn till you're facing 90° to your original direction, then do a reverse left turn till you're looking straight back along the way you came and move backward! If done correctly, you should have your pursuer in your sights. That should make his day! **Round-the-corner shots** are nice if you can pop your head round the corner just enough to hit your target then back round the corner fast enough to avoid the return fire! Here is a risky trick (if it does not work you're fried): if someone is following you really close and you're both close to one of walls,

The Quick reverse will put you behind if you can squeeze by him on the other wall. These are but a few of the possible combinations. Just experiment and see what works best for you!

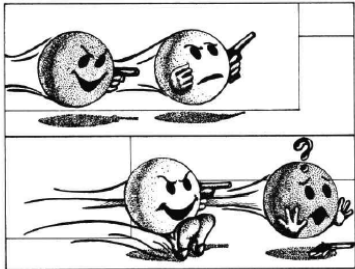
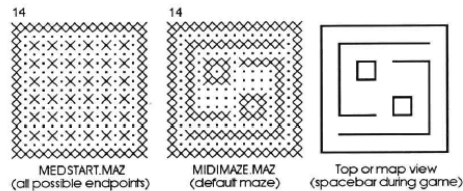


Figura C.9: Paginas 14 y 15 del manual de MIDI Maze.

7. Maze construction

There is a standard maze (MIDIMAZE.MAZ) that is loaded in with the program and any number that can be loaded after system initialization. If you would like to create your own mazes, they can be created with any word processor or text editor that saves files in ASCII.

There are some very simple rules you have to follow or your maze may do some unpredictable things. The walls of the maze consist of two parts: the wall-endpoints and wall-segments. Both the endpoints and the segments are represented by "X"s. Open spaces, or corridors are represented by periods ".". Endpoints can only be on the even rows or columns, and the middle segments have to be on the odd rows or columns. Endpoints without walls are 2 dimensional posts which are invisible, but can be run into. The number at the beginning of the file is the size of the maze both vertical and horizontal minus 1. In the example below the dimensions are 15 by 15 (numbered 0-14). The easiest way to make a maze is to load in an existing maze and modify it.



8. Midi-Cam™ Option

Included in the Midi-Maze program is an option called Midi-Cam, this feature allows a unused (non-player) computer to provide a audience view and spot coverage of the game much like television coverage of a sporting event!

This feature is available only on a computer not involved in game play but connected via midi to the other machines. The audience view is a overhead or map view of the maze **including** the positions of all the players and computer drones. Further features includes a viewscreen that automatically or manually switches between the various players and drones both active and non-active. This camera view lets people watching the game see the entire game and enjoy the action from the computer and players viewpoint.

Midi-cam can be attached and activated at any time **between** games (anytime you have a mouse pointer on the master-machine). The map and cameras display as soon as the game starts.

Midi-Cam Controls:

The Midi-Cam option only has a few manual controls. To select the Midi-Cam option press **ALT-C**. To switch back to a slave machine, press **ALT-S**. You should only operate the Midi-Cam from a slave machine. These controls are only active between games. During a game you can watch all the action on the top view of the map or by the remote viewscreen. The remote viewscreen will display the same view as the player (human or drone) would see.

To manually select players 1-10 press the number key from 1 to 0. To select players 11-16 press **SHIFT** and the number key from 1 to 6. The **SPACE BAR** selects the map view and also the returns the Midi-Cam to automatic scanning. The joystick also selects the players and the **FIRE-BUTTON** doubles as the **SPACE BAR**.

Figura C.10: Paginas 16 y 17 del manual de MIDI Maze.



(a) Disquete del videojuego MIDI Maze para Atari ST.



(b) Disquete del videojuego MIDI Maze para Atari ST y manual.

Figura C.11: Materiales del juego MIDI Maze para Atari ST.

Anexo D

Hardware prestado por la *Asociación RetroAcción*

Gran parte del hardware usado en el proyecto ha sido prestado por la *Asociación RetroAcción* (ver Figura D.1). A continuación se presenta una lista de dichos materiales y un breve resumen de que uso se les ha dado:



Figura D.1: Transporte de todo el material prestado.

- 2 Atari ST con joystick, dos monitores, disquetes de MIDI Maze y cables MIDI (ver Figura D.2): Dado el objetivo que estamos intentando conseguir, se han usado para poder ejecutar MIDI Maze en Atari ST y así ver de primera mano como se realiza la comunicación y para intentar visualizar los mensajes de MIDI con un PC. En definitiva, en gran parte de las pruebas.
- 2 adaptadores USB-MIDI, que han sido esenciales para la realización de todas las pruebas que incluyesen un PC ya que sin ellos no se puede conectar dispositivos MIDI, incluyendo los Atari ST, a un PC.
- Teclado MIDI (ver Figura ??): Se ha hecho uso de este teclado para comprobar el correcto funcionamiento de los adaptadores USB-MIDI así como la conexión de dispositivos MIDI a un PC.



(a) Atari ST.



(b) Monitor Amper.



(c) Joystick.



(d) Disquete MIDI Maze.

Figura D.2: Materiales de Atari ST

- Un MIDI patchbay (ver Figura D.3): Usado en las pruebas relacionadas con MIDI THRU.



Figura D.3: MIDI patchbay.

A lo largo de la Sección 2.1 se describe con más detalle para que y como se ha usado este hardware.