Paternoster:

Beyond the Pillars of Hercules

Juan Arturo Abaurrea Calafell, 49771137B
Íker García de León Seguí, 41749848H
Marta González Juan, 45187763P

1. Introducción

La práctica consiste en la creación de un videojuego en Easy68K que hace uso exhaustivo de acceso a periféricos, llamadas a sistema e introducción de gráficos. El programa debe satisfacer 3 requisitos de una lista proporcionada por el tutor. En nuestro caso son: uso del ratón, carga de imágenes, y la visualización de los FPS (Frames Por Segundo) actuales y deseados, de esta manera cumpliendo 3 de los 3 requisitos demandados. Además intentamos cumplir con el de acceso a ficheros, pero al no saber hacerlo lo dejamos inacabado.

Nuestro proyecto Paternoster: Beyond the Pillars of Hercules recibe este nombre ya que el juego se puede desarrollar en dos escenarios: el cielo o el infierno; y precisamente el ascensor, es el vínculo entre ambos.

El videojuego se inicia con una pantalla en la que hay que pulsar el ratón en el botón reservado para redirigir al usuario al menú principal. A continuación, el usuario se encuentra en un ascensor, donde debe tomar una decisión: subir al cielo o bajar al infierno. Una vez seleccionada la opción con las teclas de dirección, empieza el juego. Téngase en cuenta que el nivel que se selecciona con las flechas es el contrario al que se envía el personaje, como elemento sorpresa.

El objetivo del juego es destruir todos los bloques rojos en el menor tiempo posible. Los controles para moverse son W, A, S, D y para disparar las 4 flechas del teclado. En la introducción hay que usar el ratón para clicar el botón, mientras que en el final se ha de presionar W, A, S y D a la vez para seguir jugando.

2. Estructura del código

El código se estructura en 15 ficheros. A continuación se describen brevemente:

MAIN: Contiene el programa principal que inicializa las clases y posteriormente un bucle ejecuta constantemente el juego hasta que el jugador haya acabado con todos los bloques rojos.
SYSTEM: Gestiona las constantes y variables de sistema. Inicializa y actualiza la pantalla y el teclado. Contiene las rutinas de servicio a la interrupción.

CONST: Se definen las constantes de la pantalla, el teclado, la pantalla inicial y final, el jugador,los obstáculos, los agentes, los proyectiles y las constantes para identificar las rutinas de servicio a la interrupción, los traps y las rutas del audio. También se definen las constantes de estado.

VARS: Contiene las variables que almacenan el valor del teclado, la posición del jugador, el estado de los proyectiles, el estado del audio, las variables necesarias para dibujar el mapa, las variables de los FPSs, de los obstáculos y de la pantalla final.

ELEVATOR: Contiene la rutina ELEVATORPLOT, que dibuja los gráficos del menú ascensor junto con las instrucciones y ELEVATORUPDATE que comprueba si se ha pulsado alguna de las dos teclas (△ ▽), lo que supone un redireccionamiento a un conjunto de mapas u otro.

MAP: Contiene la subrutina MAPUPDATE que comprueba en qué mapa se encuentra el jugador y a través de sus coordenadas determina si ha cruzado alguna puerta. En cuyo caso copia en la variable LOADEDMAP el identificador del mapa correspondiente.

MAPDATA: Contiene los mapas del juego. Estos están representados en forma de matriz numérica.

AUDIO: Contiene: AUDINIT, subrutina encargada de inicializar el audio cargando los archivos WAV en la memoria de sonido; AUDPLAY, que distingue modo cielo, modo infierno y ascensor para cargar el audio correspondiente. También gestiona el audio de los proyectiles.

BLOCKS: Gestiona el dibujado de los bloques y la actualización de los mapas cada vez que se atraviesa una puerta.

La subrutina BLOCKSPLOT determina el mapa a dibujar y lee la matriz correspondiente definida en MAP, para proceder al dibujado.

Dependiendo del número que lea en la matriz, se accede a la subrutina correspondiente de dibujado a través de una jumplist. Los números van del 0 al 11.

Del 1 al 10, y el 12 se corresponden con un tipo de bloque. Dentro de la subrutina que dibuja cada bloque se preparan las posiciones X e Y para el dibujado del siguiente bloque.

Al final de cada fila hay un 0, cuya subrutina resetea la posición X y suma el valor 32 a la posición Y para posicionarse en la siguiente fila.

Al final de la matriz se encuentra el 11, cuya subrutina resetea las posiciones X e Y para el dibujado del mapa (el mapa se redibuja constantemente debido al doble buffer) .

La subrutina ISBLWHICHPROP (is block which property), que determina si el bloque tiene propiedades. Hay dos propiedades: colisión (si el jugador se encuentra con ese bloque no puede avanzar) y obstáculo (impide que un obstáculo se sitúe en ese bloque).

La subrutina BLKCOLL (block collision), ejecuta las propiedades determinadas en ISBLWHICHPROP.

PLAYER: Gestiona al jugador. La subrutina PLAYERUPD comprueba las colisiones, los obstáculos y actualiza la posición del jugador según el valor de las teclas.
La subrutina PLAYERPLOT dibuja el jugador en función de los resultados de PLAYERUPD.
PROJECTILE: Gestiona los proyectiles. La subrutina PROJINIT inicializa el proyectil cuando el jugador dispara. PROJUPD actualiza la posición del proyectil hasta que choca con algo entonces salta a la subrutina PROJHITOBS, esta hace que el proyectil que ha chocado con una pared desaparezca y si choca con un enemigo lo mata y elimina el proyectil.
INTRO: Es el primer estado que se ejecuta y por ello el inicializador ISCINIT activa las interrupciones de ratón. ISCUPD comprueba si el botón ha sido clicado, y en caso afirmativo pasa al ascensor (STANEXT = 1). ISCPLOT dibuja un botón con texto parpadeante y el bitmap localizado en DATA/INTRODATA.
INTRODATA: Contenedor del bitmap del título del videojuego.
OBSTACLE: Agente que al inicalizarse (OBSINIT) recibe por parámetro D1 el mapa que va a ocupar. En ese momento se elige pseudo-aleatoriamente el lugar en el que va a estar sin que colisione con alguna pared o bloques del estilo que están en la variable BLWHICHNTRA. OBSPLOT simplemente dibuja un bloque rojo. HITOBS es llamado por el proyectil que haya colisionado con el obstáculo en cuestión, y lo que hace es destruirlo además de comprobar la condición de victoria (NOBS = 0).
STATES: STAINIT cierra todos los ficheros porque en la documentación de Easy68k indicaba que era recomendable hacerlo si se iban a usar ficheros más adelante, cosa que no pudimos conseguir exitosamente. STAUPD se dedica a comprobar el estado actual y el siguiente y es el encargado de llamar a los inicializadores de los estados y a sus updates. STAPLOT ejecuta la subrutina de dibujado del estado actual. STAEMPTY solamente se usa en caso de que uno de los estados no tenga alguno de las 3 subrutinas; init, update y plot.

3. Principales dificultades

Nos habría gustado cargarle una textura al personaje principal, sin embargo esto ralentizaba mucho el juego.

Al intentar leer de scores.txt siempre devolvía 1 en D0, indicando que se ha alcanzado EOF, además de que el D2 de esta task (53) no cambiaba, no devolvía los bytes realmente leídos como indica la documentación. Debido a esto y a que no hay ninguna otra ayuda por la página oficial no hemos conseguido implementarlo debidamente.

El reproductor de audio directX no reproducía varios sonidos al mismo tiempo correctamente, como dice la documentación devolvía un número diferente a 0 a D0 al tratar de hacerlo, lo que indica un error, aunque el código era correcto. Debido a esto para reproducir 2 sonidos simultáneamente tuvimos que recurrir a usar los 2 reproductores.

4. Añadidos al código suministrado

· UTLCODE: Se ha añadido el UTILPRINT del juego de la paleta de los vídeos didácticos, UTLPRINTNUM, que es una modificación para imprimir un número y UTLCHCOLL, el cual está inspirado en el de SFIGHTER.
· STATES: Se ha ajustado los estados poniendo los 4 del videojuego en vez de los del juego de ejemplo que se hacían en los vídeos didácticos que teníamos que vernos cada semana.
· SYSTEM: Se ha añadido la lógica de los FPS y se ha modificado KBDUPD para que encaje con lo que se quería de él, WASD y las flechas.
· PROJECTILE: Inspirado del juego de la raqueta y las pelotas, se han añadido comprobaciones de colisiones y la eliminación de obstáculos en caso de colisión.

El resto de ficheros son más o menos únicos y hechos por nosotros.

5. Conclusiones

Es comprensible que el lenguaje ensamblador no sea el más adecuado para realizar esta clase de proyectos debido a la complejidad que supone trabajar a bajo nivel, especialmente para la parte gráfica. Pese a esto, es un proyecto adecuado para la asignatura ya que ayuda a entender el acceso a periféricos a un nivel al que no llegamos en otras asignaturas.