ESTRUCTURA DE COMPUTADORES II

Desarrollo de un videojuego en EASy68K

SPACE SHOOTER

Curso 2023-2024
Profesor: Burguera Burguera, Antoni

Autores:

Pau Toni Bibiloni Martínez, **46391835F**

Joan Martorell Coll, **43233750Y**

INTRODUCCIÓN

Este programa, desarrollado en el lenguaje de ensamblador EASy68K, es un emocionante juego de disparos de desplazamiento vertical. En este juego, el jugador asume el control de una nave espacial y se embarca en una misión llena de acción para esquivar y destruir naves enemigas, representadas como OVNIs, que aparecen aleatoriamente desde la parte superior de la pantalla.

Al comienzo de cada partida, el jugador empieza con tres vidas. Cada colisión con una nave enemiga resulta en la pérdida de una vida. Cuando el jugador agota todas sus vidas, la partida termina.

La nave del jugador está equipada con la capacidad de disparar proyectiles verticalmente. Cuando un enemigo es alcanzado por un disparo, este es eliminado, lo cual se representa mediante una animación de explosión. Cada enemigo destruido suma 5 puntos al marcador.

El objetivo principal del juego es acumular la mayor cantidad de puntos posible antes de perder todas las vidas. Esto introduce un elemento de estrategia y habilidad, ya que el jugador debe equilibrar el esquivar a los enemigos con el atacarlos para ganar puntos.

En la pantalla de inicio del juego, se muestran las tres puntuaciones más altas alcanzadas por los jugadores. Estas puntuaciones se almacenan en la memoria del programa, por lo que persisten incluso después de cerrar el juego. Esto proporciona un objetivo constante para los jugadores, incentivándolos a superar las puntuaciones más altas en sus futuras partidas.

CÓMO JUGAR

Este juego, con su jugabilidad intuitiva y directa, ofrece una experiencia de usuario fluida y agradable. Al iniciar el programa, se presenta al jugador con una pantalla de inicio. Para comenzar la partida, el jugador simplemente tiene que presionar la tecla ESPACIO. Este simple gesto pone en marcha la acción y sumerge al jugador en el emocionante mundo del juego.

Una vez en el juego, el jugador tiene el control total de una nave espacial. Para mover la nave, el jugador puede usar las teclas de flecha del teclado. Estas permiten la navegación en las cuatro direcciones posibles: arriba, abajo, izquierda y derecha. Esto proporciona al jugador una gran flexibilidad y control sobre el movimiento de la nave, permitiéndole esquivar a los enemigos y maniobrar a través del campo de juego con facilidad.

Además de mover la nave, el jugador también tiene la capacidad de atacar a las naves enemigas. Para hacer esto, el jugador puede presionar la tecla ESPACIO, que activa el sistema de armas de la nave y dispara a los enemigos

Cuando la partida termina y aparece la pantalla "GAME OVER", el jugador puede iniciar una nueva partida presionando nuevamente la tecla ESPACIO. Esto llevará al jugador de vuelta a la pantalla principal, permitiéndole comenzar una nueva partida e intentar superar su puntuación anterior.

ESTRUCTURA DEL CÓDIGO

El código del juego está organizado de manera modular, con cada módulo representando una entidad o funcionalidad específica del juego. A continuación, se presenta una descripción detallada de las clases que componen la estructura general:

1. **AGENTLST**: Esta clase es la encargada de gestionar la lista de agentes en el juego. Se ocupa de la creación, actualización y renderizado de los agentes, lo que incluye tanto a la nave del jugador como a las naves enemigas.
2. **AUDIO**: La clase AUDIO maneja todos los aspectos relacionados con el sonido y la música en el juego. Proporciona una capa de abstracción para la manipulación de elementos auditivos, facilitando la reproducción de efectos de sonido y música de fondo.
3. **CONST** y **SYSCONST**: Estas clases contienen constantes esenciales que se utilizan en todo el juego. Esto incluye las dimensiones de la pantalla, los límites de FPS, y otras constantes fundamentales.
4. **COUNTDOWN**: La clase COUNTDOWN maneja una cuenta regresiva, que puede ser utilizada para eventos como el inicio del juego o situaciones de tiempo limitado. Esto añade un elemento de presión y urgencia al juego, aumentando su dificultad y emoción.
5. **ENEMIGO**: Esta clase se encarga de gestionar los enemigos en el juego. Incluye la creación de nuevas naves enemigas, su movimiento a través de la pantalla, y su renderizado.
6. **MAIN**: La clase MAIN es el corazón del juego. Inicia y controla el ciclo de vida del juego, sirviendo como el punto de entrada al código y coordinando las interacciones entre las demás clases.
7. **PLAYER**: Esta clase se encarga de la representación y gestión del jugador en el juego. Incluye funciones para la creación de la nave del jugador, su movimiento, el disparo de proyectiles, y su renderizado en la pantalla.
8. **SHOT**: La clase SHOT representa los disparos o balas en el juego. Maneja su creación, movimiento a través de la pantalla, y renderizado.
9. **SPAWNER**: Esta clase se encarga de la generación dinámica de enemigos y de estrellas de fondo. Esto contribuye a la variabilidad y desafío del juego.
10. **STAR**: La clase STAR gestiona las estrellas en el fondo del juego. Incluye su creación, movimiento a través de la pantalla, y renderizado. Esto contribuye a la estética visual del juego, creando una sensación de profundidad y movimiento.
11. **STATES**: Esta clase maneja los diferentes estados del juego, como el menú principal, el desarrollo del juego, la pantalla de pausa, entre otros. Esto permite al juego tener diferentes modos y comportamientos, dependiendo del estado actual.
12. **SYSTEM** y **SYSVAR**: Estas clases se ocupan de las operaciones del sistema, tales como la inicialización del juego, la carga de recursos y la gestión de la memoria. Esto incluye tareas como la configuración inicial del juego, la carga de imágenes y sonidos, y la gestión de la memoria para evitar fugas y optimizar el rendimiento.
13. **UTIL**: La clase UTIL contiene funciones de utilidad que son empleadas en diversos puntos del juego, proporcionando herramientas comunes.
14. **VAR**: La clase VAR contiene variables que se utilizan de manera global en el juego. Esto facilita el acceso y la manipulación de datos que necesitan ser compartidos entre diferentes partes del código.

15. **XPLOSION**: La clase XPLOSION maneja las explosiones en el juego. Incluye funciones para la creación de nuevas explosiones, su movimiento a través de la pantalla, y su renderizado. Esto añade un elemento visual espectacular al juego, aumentando su impacto y satisfacción.
16. **RECORDSCORE**:Esta clase se encarga del registro y visualización de las puntuaciones más altas. Esto contribuye a la competitividad y persistencia de los logros.

Cada una de estas clases se encuentra en su propio archivo .X68, y todas se integran de manera modular mediante inclusiones en el archivo principal main.X68. Este enfoque promueve la modularidad y facilita el mantenimiento y la expansión del código. Al separar las diferentes partes del código en módulos independientes, se facilita la comprensión de cada parte en su propio contexto, y se evita la complejidad y confusión que puede surgir cuando diferentes funcionalidades están entrelazadas. Esto también permite un mantenimiento y una depuración más eficientes, ya que los errores pueden ser aislados y corregidos en sus respectivos módulos sin afectar al resto del código. Además, la modularidad promueve la reutilización del código, ya que los módulos pueden ser reutilizados en diferentes partes del juego.

IMPLEMENTACIONES

MÚSICA

El archivo AUDIO.X68 gestiona los sonidos del juego a través de dos funciones principales: AUDINIT, que inicializa los sonidos, y AUDPLAY, que reproduce los sonidos.
Los sonidos se activan en diferentes situaciones durante el juego:

- **INTRO.WAV**: Este sonido se reproduce en bucle durante la pantalla inicial del juego.
- **COUNTDOWN.WAV**: Este sonido se reproduce durante la cuenta regresiva que precede al inicio de una nueva partida. Se reproduce una vez por cada número de la cuenta regresiva.
- **SUI.WAV** o **CNTDWNFINAL.WAV**: Ambos sonidos pueden ser utilizados para el último número de la cuenta regresiva que da inicio a la partida. Actualmente, solo se utiliza SUI.WAV.
- **SHOT.WAV**: Este sonido se reproduce cada vez que el jugador dispara.
- **IMPACT.WAV**: Este sonido se reproduce cuando un disparo del jugador impacta contra una nave enemiga.
- **HERIDO.WAV**: Este sonido se reproduce cuando el jugador colisiona con una nave enemiga.
- **GAMEOVER.WAV**. Este sonido se reproduce cuando el jugador pierde todas sus vidas y aparece la pantalla de "Game Over". Se reproduce una sola vez.

USO DE FICHEROS

En la carpeta "DATA", se encuentra el archivo RECORDSCORE.X68, que se encarga de gestionar las puntuaciones más altas del juego y almacenarlas en el archivo SCORES.DAT. Este archivo contiene tres funciones principales:

- **SCOREAD**: Esta función lee las tres puntuaciones más altas almacenadas en el archivo SCORES.DAT.
- **SCOREC**: Esta función guarda una nueva puntuación en el archivo SCORES.DAT.
- **SCOUPD**: Esta función utiliza las dos funciones anteriores para determinar si la puntuación más reciente del jugador se encuentra entre las tres puntuaciones más altas. Si es así, la nueva puntuación se guarda en el archivo SCORES.DAT.

Estas funciones trabajan en conjunto para mantener un registro de las tres puntuaciones más altas del juego, proporcionando a los jugadores un objetivo constante para superar en sus futuras partidas.

VISUALIZACIÓN DE IMÁGENES

La carpeta "IMAGES" contiene varios archivos que gestionan las imágenes y los sprites utilizados en el juego:

- **intro.png**: Esta es la imagen que se muestra en la pantalla de inicio del juego. Está en formato PNG.
- **converter.py**: Este es un programa Python que convierte la imagen intro.png en una representación pixel a pixel en formato hexadecimal. La salida se imprime en la consola.
- **SPRITES.X68**: Este archivo contiene dos sprites:
    - INTRO: Este sprite es la representación en hexadecimal de la imagen intro.png, generada por el programa converter.py.
    - HEART. Este sprite es similar a INTRO, pero en lugar de representar los colores en hexadecimal, utiliza solo cuatro colores: 0 para vacío, 1 para rojo, 2 para negro y 3 para blanco. Este sprite tiene la forma de un corazón y se utiliza para representar las vidas del jugador.

Para renderizar el sprite INTRO, el juego recorre la lista de píxeles y dibuja un cuadrado de tamaño NxN para cada píxel, ampliando así la imagen. El sprite HEART se maneja de manera similar, pero sólo utiliza cuatro colores y se repite tantas veces como vidas tenga el jugador.

Todas las variables que afectan a las imágenes se encuentran en su sección correspondiente en el archivo CONST.X68.

FRAMES POR SEGUNDO

Para calcular la tasa de frames por segundo (FPS) del juego, se utilizan varias variables que se inicializan en el archivo MAIN.X68:

- **FPSCOUNT**: Esta variable se inicializa a cero y se incrementa en uno en cada ciclo del juego, contabilizando así el número de frames.
- **FPSREAL**: Esta variable se inicializa con el objetivo de FPS que se desea alcanzar y se actualiza con el valor de FPSCOUNT cada segundo. Es la tasa de FPS que se muestra en pantalla.
- **TIME**: Esta variable almacena el tiempo actual.

El cálculo de la tasa de FPS se realiza de la siguiente manera: en cada ciclo del juego, se incrementa FPSCOUNT en uno y se calcula el tiempo transcurrido desde el último valor guardado en TIME. Si este tiempo es mayor o igual a un segundo, se actualiza FPSREAL con el valor de FPSCOUNT, se reinicia FPSCOUNT a cero y se actualiza TIME con el tiempo actual. Si el tiempo transcurrido es menor a un segundo, el ciclo del juego continúa sin realizar ninguna acción.

Este método permite calcular y mostrar la tasa de FPS en tiempo real, proporcionando una medida de rendimiento del juego.

PRINCIPALES DIFICULTADES

Durante el desarrollo del proyecto, nos hemos enfrentado a varias dificultades que han requerido soluciones creativas y técnicas. Aquí están algunas de las principales:

- **Integración de audio**: La integración de efectos de sonido y música de fondo en el juego ha presentado desafíos tanto técnicos como creativos. Hemos tenido que trabajar con las limitaciones de la biblioteca de audio y encontrar la música y los efectos de sonido adecuados para la atmósfera del juego.
- **Optimización del rendimiento**: Mantener una tasa de frames por segundo (FPS) constante y alta ha sido otro desafío, especialmente dado el número de objetos y operaciones que se manejan en cada frame. Hemos tenido que optimizar nuestro código y utilizar técnicas como la actualización selectiva y el culling para mantener un rendimiento suave.

CONCLUSIÓN

En conclusión, este proyecto de juego demuestra una aplicación efectiva de los principios de programación modular. Cada componente del juego, desde la gestión de los agentes hasta la reproducción de audio y la manipulación de la interfaz de usuario, se ha encapsulado en módulos independientes. Esto no solo facilita la comprensión y el mantenimiento del código, sino que también permite una mayor flexibilidad y reutilización.

La estructura del código refleja una cuidadosa planificación y diseño, con cada clase y módulo desempeñando un papel específico en la funcionalidad general del juego. Esto resulta en un código limpio y organizado que es fácil de seguir y modificar.

Además, el uso de constantes y variables globales, así como de funciones de utilidad, demuestra una atención cuidadosa a la eficiencia y la optimización. Estos elementos permiten un rendimiento eficiente del juego y facilitan la manipulación de datos y la implementación de funcionalidades.

En general, este proyecto es un excelente ejemplo de cómo la programación modular puede ser utilizada para crear un juego complejo y funcional. Sirve como un modelo útil para futuros proyectos de desarrollo de juegos y demuestra las ventajas de un diseño de código bien estructurado y modular.