TANKS 68K!

Autores:
Marcos Socías Alberto 41619112Z
Sergi Villalonga Gamundí 45193276R

INTRODUCCIÓN

“Tanks68K!” es un juego de acción basado en una batalla de tanques. El jugador controla un tanque verde, cuyo objetivo es destruir a todos los tanques enemigos, representados de color rojo, a lo largo de 5 niveles. Hemos querido introducir mecánicas como el rebote de las balas (tanto vertical, horizontal, como diagonalmente), la posibilidad de apuntar y disparar en 8 direcciones, etc.

CÓMO JUGAR

El tanque jugador se controla con las teclas “WASD” para moverse. Se apunta moviendo el ratón para dirigir el cañón en una de las ocho direcciones posibles y se dispara con el click izquierdo. El jugador tiene 10 de vida y los enemigos 3. En caso de colisión de un enemigo contra el jugador, el enemigo explota y el jugador perderá 3 puntos de vida. Las balas restan 1 punto de vida. El jugador solo puede disparar 3 balas simultáneas. Tanto las balas del jugador, como las de los enemigos rebotan hasta 2 veces. Si superas los 5 niveles, ganarás, en caso contrario, perderás e irás al menú principal para volver a empezar.

ESTRUCTURA DEL CÓDIGO

Nuestro juego se ejecuta con la clase MAIN. Primeramente, hace los includes de todas las clases (las clases de variables se encuentran abajo). En la inicialización, deshabilita las interrupciones, ejecuta las subrutinas de inicialización del sistema (SYSINIT), inicialización de los fps (FPSINIT) e inicialización del código complementario (UTLINIT). Seguidamente, se inicia la subrutina que carga todas las imágenes en memoria (LOADFILES) que a su vez utiliza la clase “FILELOAD” ubicada en la carpeta “LIB” cada vez que carga una imagen. Para finalizar la inicialización, se ejecuta “STAINIT” para comenzar los estados. El update de la clase MAIN lee el teclado y actualiza los estados y los fps. En el plot se dibujan los estados y los fps con una lógica explicada más adelante.

La clase SYSTEM se encarga de gestionar la E/S del juego. Inicializa y gestiona el dibujado de la pantalla, el teclado, el sonido, el ratón y el temporizador de ciclos a través de sus subrutinas. La clase FPS se encarga de comprobar cuáles son los FPS actuales, compararlos con los deseados y dibujarlos.

La clase “UTLCODE” se complementa con las clases “UTLCONST” y “UTLVARS”. En la clase del código se encuentran subrutinas útiles como “UTLCHCOL” que comprueba colisiones entre dos objetos; “RELPOS” que dada una coordenada, te dice en qué casilla está y en qué parte de esta; “KILLBULS” que elimina todas las balas que haya; “PLIMASEL” y “ENIMASEL” que sirven para elegir la imagen del jugador y enemigo que toca en cada momento (según sus movimientos); “TANKCOL” que sirve para detectar las colisiones entre tanques; “COLBYKMK” sirve para gestionar la colisión entre tanque enemigo y jugador (restarle vida al jugador y destruir tanque enemigo); “APUNTADOR” se encarga de gestionar el apuntado del jugador, el bot del jugador de la demo y del enemigo (jugador apunta donde esté el ratón, el bot jugador apunta a los enemigos y los enemigos apuntan al jugador).

La clase “STATES” gestiona todos los estados y sus cambios a través de su subrutina “STAUPD” y de las clases gestionadas “INTRO”, “INSTRUCTIONS”, “GAME”, “GOVER” y “WIN”. Todas esas clases tienen sus respectivos init y upd, vamos a profundizar en la clase GAME ya que todas las demás son simples menús.

La clase “GAME” es la encargada de inicializar los niveles, en el “GAMINIT” realiza la inicialización del nivel 1, junto al reinicio de variables; en el “GAMUPD”, comprueba en qué punto del juego estamos, e inicializa un nivel si es necesario, también se encarga de comprobar si hemos ganado o perdido; en cuanto al “GAMPLOT”, se encarga de mostrar por pantalla los diferentes elementos del juego (tanques, balas, el propio mapa…). Para poder gestionar los niveles, hemos creado la clase “LEVEL” que contiene el init de cada uno de los niveles, donde se encarga del spawn del jugador y de los enemigos.

Las clases agente que tenemos son la clase “TANK” y la clase “BULLET”; en cuanto a la clase tanque, estos se inicializan colocando sus coordenadas, y comprobando si serán jugador o bot; estos se actualizan en función de si el jugador desea moverse o disparar (caso jugador), o si el algoritmo del bot le indica disparar o moverse (bot). Este algoritmo funciona con el siguiente criterio: en 60 de cada 80 ciclos hará un movimiento aleatorio (mantendrá esa dirección durante los 60 ciclos). En los 20 ciclos restantes, seguirá al jugador. A la hora de disparar apuntan siempre al jugador gracias a la subrutina “APUNTADOR”. Durante todo el código de “TANK”, hay branches que dividen el código (solamente cuando es necesario) entre jugador real, jugador bot y enemigo. En cuanto al dibujado de tanques, miramos la imagen que mejor representa el movimiento y apuntado del tanque con “PLIMASEL” y “ENIMASEL” según sea jugador o enemigo, y mostramos por pantalla dicha imagen. Respecto a las balas, estas se crean si un tanque ha disparado, su inicialización coloca la bala y se asegura de que no haga spawn dentro de un bloque; el update lo realiza por su cuenta (sin intervención del jugador), actualiza su posición según su velocidad, y rebota si tiene que rebotar, si detecta que ha colisionado con un tanque, o es su tercera colisión con una pared, esta se autodestruye; el plot de la bala es dibujar y pintar un círculo. Además, cabe destacar que ambas clases “TANK” y “BULLET”, al ser agentes, tienen sus atributos propios y atributos recibidos por parámetro (están explicados en las respectivas cabeceras de las clases).

Para la gestión de agentes y memoria, utilizamos las clases proporcionadas “AGLCODE”, “DMMCODE”, “DMMVARS” y “DMMCONST”.

La clase “SCORE” muestra la puntuación del jugador (tanques enemigos derrotados).

Para el dibujado del mapa, hacemos uso de la clase “MAP”; esta, gracias al método “MAPPLOT”, recorre la matriz situada en “MAPDATA”, y dependiendo del valor, sitúa un cuadrado de un color u otro; como he brevemente mencionado, hacemos uso de una estructura matricial, con un doble bucle (uno recorre filas, y el otro columnas).

Para el almacenamiento de variables y constantes, hemos hecho uso de las clases “VARS”, “SYSVARS”, “CONST”, y “SYSCONST”; que almacenan variables y constantes respectivamente, y unas se encargan de las de sistema, y otras de las de juego.

PRINCIPALES AGREGADOS AL CÓDIGO SUMINISTRADO

Nuestro juego empezó bajo la base del juego “PONG” de los tutoriales, para hacer el tanque, empezamos con la base de la pala, añadiéndole movimiento horizontal; a pesar de estas ayudas, nosotros hemos tenido que hacernos cargo de muchos fragmentos del código; todos los extras los hicimos por cuenta propia (tuvimos tutorías, y charlas para organizarlo), las colisiones de los tanques fue trabajo nuestro, el cambio de nivel lo hicimos bajo nuestras propias ideas, etcétera.

STATES, INTRO, INSTRUCTIONS, GOVER, WIN y SCORE las hemos acomodado a nuestro caso particular sin mucha modificación. El cambio más grande en estas clases sería el dibujado de las imágenes en INTRO y WIN, además de cambios en la lógica para pasar de un estado a otro (cambiar teclas por ejemplo). La clase GAME ha sido modificada para tener más niveles, y para alterar variables nuestras.

La clase UTLCODE, contiene en gran medida el código auxiliar que se nos ha sido otorgado, no obstante, nosotros hemos introducido subrutinas que nos eran necesarias para una mejor utilización del código: RELPOS, KILLBULS, PLIMASEL, ENIMASEL, TANKCOL, COLBYKMK.

En cuanto a las clases VARS, SYSVARS, CONST, SYSCONST; hemos introducido todas las variables/constantes que hemos necesitado para la implementación del juego.

Como antes hemos mencionado, tanto TANK como BULLET son clases que son heredadas de PAD y BALL respectivamente, pero han sido tan modificadas, que ya no hay rastros de sus orígenes.

PRINCIPALES DIFICULTADES

A lo largo del desarrollo del videojuego, nos surgieron diferentes errores/bugs, los más importantes de ellos son:

-Las colisiones de tanques; las colisiones entre un tanque y la pared son bastante eficaces, pero tienen un grave problema, en las esquinas los tanques pegan mini saltos; esto es debido al método de colisiones que implementamos, aunque en la actualidad, estos saltos están bastante camuflados. Otro problema que tuvimos con las colisiones, fue entre los propios tanques; estos se bloqueaban, o provocaban errores aún mayores; pero llegamos a la conclusión de que si dos tanques enemigos colisionan, no es demasiado grave; pero si colisiona un enemigo con el jugador, lo mejor sería que algo sucediese, e hicimos que si un enemigo y el jugador colisionan, el enemigo muera, causando graves daños al jugador.

-Las diagonales han causado problemas leves; esto es debido a que el uso de decimales es complejo en el 68k, y nosotros deseábamos trabajar con enteros; los decimales ayudarían a realizar movimientos diagonales con más lógica, para conseguir una lógica similar a la real, hemos buscado números cuyos decimales se aproximen a un número entero; si bien no es completamente realista, hemos tratado de aproximarlo.

-Carga de la imagen: Al principio intentábamos cargarla con un fichero .txt. Eso hacía que fuera un proceso muy lento de carga, así que pasamos a utilizar un fichero .bin. Cuando utilizábamos el txt, perdimos bastante tiempo intentando adaptarlo en el código de FILELOAD, ya que dos bytes de un caracter del txt lo teníamos que pasar a 1 byte de ensamblador con desplazamiento de bits y ese tipo de técnicas. Finalmente con el fichero bin nos salió más fácil la solución.

CARACTERÍSTICAS ADICIONALES ESCOGIDAS

-Uso de ficheros: Utilizamos ficheros .bin para codificar las imágenes. Los cargamos a través de la clase “FILELOAD” (carga un fichero) que utilizamos las veces necesarias mediante la clase “LOADFILES” (carga todos los ficheros necesarios).

-Visualización de imágenes: Como se ha mencionado anteriormente, utilizamos ficheros .bin para codificar las imágenes JPG (menú principal, sprites del tanque jugador/enemigo e imagen de victoria). Las codificamos a través de un código de Python, cuya clase se llama “ImageLoader.py”. Las dibujamos píxel a píxel, cosa que ralentiza el rendimiento del juego, así que para solucionar el problema, redujimos el tamaño de los tanques de 24x24 a 16x16. Además, para que la carga de la imagen del menú principal fuese más rápida, redujimos su tamaño para que no ocupase toda la pantalla y dejamos un marco verde. Las imágenes del menú principal y de victoria se pintan en la subrutina “ISCINIT” de la clase “INTRO” y a través de “WININIT” de la clase “WIN” respectivamente. La imagen de los tanques, al ser un tratamiento considerablemente distinto, se pinta en la subrutina “TANPLOT” de la clase “TANK”. Además, se utilizan las subrutinas “PLIMASEL” y “ENIMASEL” de la clase “UTLCODE” para elegir qué imagen de cada tanque pintar en cada frame. Aquí se ve el código Python del codificador de la imagen a binario:

ImageLoader.py

```python
from skimage.io import imread

theImage = imread('../IMAGES/tanksIntro.jpg')
numRows, numCols, _ = theImage.shape
outData = []

for r in range(numRows):
    for c in range(numCols):
        theRed, theGreen, theBlue = theImage[r, c]
        outData += [0, theBlue, theGreen, theRed]

with open('OUTPUT/MENUIMAGE.bin', 'wb') as outFile:
    outFile.write(bytearray(outData))
```

-Uso del ratón: Cuando comienza una partida, el jugador deberá mover el ratón para apuntar y pulsar el click izquierdo si desea disparar. Una vez pulsado, se consiguen las coordenadas deseadas para disparar, y buscamos cuál de las rectas predefinidas cumple mejor con la trayectoria deseada; modificamos los valores de velocidad y origen para la bala, y esta empieza su recorrido. Todo esto es gestionado en

-Attract Mode: Después de pasar aproximadamente 15 segundos en el menú principal, se dirigirá al jugador a una demostración del gameplay del juego. En esta, el jugador es sustituído por una pequeña IA (dispara hacia los enemigos y se mueve aleatoriamente). Por otro lado, los bots enemigos funcionan igual que en el juego principal. Solo incluye el nivel 1 (para evitar mostrar todo el juego por adelantado), si éste finaliza, volverá al menú principal. También se puede regresar al menú con la barra espaciadora. Toda la demo se gestiona con condicionales a lo largo del código (clases “TANK” y “GAME” por ejemplo) utilizando la variable “DEMO”.

-Fotogramas por segundo: Nuestra constante de FPS se llama “SCRFPS” y se encuentra en la clase “SYSCONST”. Se miden los fps reales con una variable que utilizamos como contador “ACTFPS” y se comparan con los deseados en la esquina superior derecha de la pantalla. La gestión de este extra se encuentra en la clase “FPS”.

-Tasa de fotogramas por segundo dinámica: Nuestros FPS dinámicos se gestionan en el plot de la clase “MAIN”. Utilizamos la variable “SHWFPS” porque la variable “ACTFPS” se actualiza constantemente (ya que es un contador), y por tanto no podemos acceder a su valor cuando queramos. Por tanto, SHWFPS sí que contiene ese valor real. Hacemos una gestión sencilla, si los FPS son menores a 20, reducimos la frecuencia de los plot (hacemos que no se dibuje cada ciertos ciclos con el bit test del bit 16). Si los FPS son menores a 15, reducimos aún más la frecuencia de los plot (esta vez comparamos con un bit test del bit 8). Hemos comprobado que funcione en nuestros ordenadores y lo hace correctamente, pero, en caso de que falle y el juego vaya mal, sería recomendable modificar los valores del 20 y el 15 a 0s. Por otro lado, en caso de querer comprobar la reducción de plots, se pueden poner valores más altos para sustituir ese 20 y 15.

CONCLUSIÓN

En conclusión, el juego empieza mostrando una imagen, si pulsamos el espacio, el juego inicia; hay un tanque controlado por el jugador y varios enemigos, todos los tanques pueden disparar balas, y estas rebotan, si todos los enemigos mueren, ganas; si mueres, pierdes. Si en la pantalla de inicio esperas 16 segundos, inicia un modo automático, donde el jugador va solo. En el gameplay se puede observar los fps de la partida, y estos tratarán de ajustarse a lo necesitado. A lo largo de la ejecución se escucharán sonidos del juego.

Hemos tratado de hacer un juego divertido, dinámico, y que traiga nostalgia; pero también hemos tratado de implementar la mayor cantidad de extras posibles, estos extras ya mencionados funcionan como deben, y estamos orgullosos con el resultado obtenido.