

MARCIANOS



68K:

PARTICIPANTES:	DNI:
Aarón Fernández Ramón	45 18 66 05 T
Tomás Dengra Femenia	43 46 72 47 F

INTRODUCCIÓN:

A modo de proyecto final de la asignatura de Estructura de Computadores II se nos pidió programar un videojuego, pues es un modo de practicar la interacción exhaustiva con periféricos. Por este motivo, decidimos programar nuestra propia versión del clásico *Space Invaders*. Para ello, empezamos a crear nuestro juego a partir de fragmentos de código del profesorado y, desde allí, añadimos todas las funcionalidades que nos interesaron. En este documento explicamos en qué consiste el juego y la implementación interna que permite su funcionamiento

CÓMO JUGAR:

Al igual que en el *Space Invaders* el objetivo será sobrevivir a las hordas de marcianos matando la máxima cantidad posible de ellos moviendo la nave con las flechas  y disparando con la tecla .

Los marcianos se generarán por filas y morirán al recibir un disparo, pero deberás tener cuidado, pues ellos también pueden disparar. Además, algunos de ellos se separarán del grupo y tratarán de lanzarse contra ti.

ESTRUCTURA DEL CÓDIGO:

El código se encuentra distribuido en 20 documentos de tipo X68 (haciendo un pequeño abuso de notación, los llamaremos “clases”) y las carpetas LIB -otorgada por el profesorado- y SND. LIB contiene rutinas de manejo de agentes, memoria dinámica y utilidades para el juego (pantalla centrada y vector de números pseudoaleatorios). Por su parte, la carpeta SND contiene los archivos de audio usados durante el juego. Respecto a los 20 documentos de código 68K, encontramos:

- **MAIN:** Es la clase desde la cual se ejecuta el programa. Se encarga de inicializar las primeras clases para que se inicie la ejecución (SYSTEM, UTLCODE de LIB, MOUSE y STATES) y maneja el bucle con el plot y el update de las mismas.
- **SYSTEM:** Inicializa y actualiza la pantalla con doble buffer, el teclado, el sonido y el manejo de memoria dinámica del sistema. También revisa y actualiza los FPS.
- **STATES:** Se encarga del manejo entre los distintos estados del juego mediante una lista de vectores ordenados de estados y un bucle que revisa el estado actual y el siguiente.
- **INTRO:** Es el estado con el que se inicia el juego. Cuenta con un título, “MARCIANOS 68K” y tres botones, “START”, “INSTRUCTIONS” y “EXIT” (en este último se encuentra el SIMHALT). Si bien la partida da comienzo al pulsar “START”, esta pantalla tiene implementado el salto a la pantalla de

instrucciones si no se toca ningún botón durante un tiempo, así como la posibilidad de pulsar la tecla "X" para iniciar la partida (pasando por el estado INSTRUCTIONS) para no tener que separar las manos del teclado.

- **BUTTON:** Define las dimensiones de los botones en la pantalla de introducción, los dibuja cambiando su aspecto si el ratón está encima y define los textos en su interior. También es la clase encargada de forzar el cambio de estado al que corresponda al botón pulsado.
- **MOUSE:** Detecta los movimientos del ratón y el uso de botones en el mismo.
- **INSTRUCTIONS:** Este estado muestra un breve texto de cómo jugar y un letrero que parpadea (gracias al UTLPRINT de la clase UTLCODE de LIB) diciendo "PRESS "X" TO START". Si el usuario no presiona nada durante un tiempo, el juego comienza automáticamente.
- **GAME:** Es el estado en el que se juega la partida. Inicializa la memoria dinámica de la clase DMMCODE de LIB, así como la nave (SHIP), controlada por el jugador y la puntuación (SCORE). A continuación y mientras dure el juego, actualizará el estado de la nave, las filas de marcianos generadas, los disparos tanto de nave como de marcianos, los marcianos auto-guiados y la puntuación. Además, tanto ambos tipos de marcianos como ambos tipos de disparos son de tipo agente, por lo que controla la cantidad de cada uno de ellos en la pantalla para evitar la sobrecarga del juego.
- **SCORE:** Es la clase encargada de imprimir por pantalla (esquina superior izquierda) la puntuación del jugador y de ir actualizándola. Dado que arriba a la derecha se muestran los contadores de "MAX FPS" y "FPS", esta clase también se encarga de escribir y actualizar esta información, aunque el conteo de los FPS no se lleva a cabo aquí, si no en SYSTEM.
- **SHIP:** Es la nave del jugador, pero esta clase también maneja su barra de vida. La inicialización de la nave la sitúa en la parte inferior central de la pantalla y se va actualizando para comprobar las teclas que presiona el jugador para mover la nave (actualiza el dibujo y sus zonas de colisión) o disparar. Además, el bucle de comprobaciones revisa sus colisiones con ambos tipos de marcianos y con los disparos de los mismos, actualizando la barra de vida en ambos casos (-1 vida). Otras colisiones que revisa son las de los bordes de la pantalla (superior, inferior, izquierdo y derecho) para que la nave no pueda salirse de los límites de la misma.
- **SHOOT:** Corresponde al disparo del jugador (nave). El disparo se inicializa un poco por encima de la parte central de la nave, que correspondería al cañón, y avanza hacia el límite superior de la pantalla. Mientras avanza comprueba las colisiones con ambos tipos de marciano y con el límite final de la pantalla, auto-destruyéndose en todos los casos mediante el AGLKILL de la clase AGLCODE de LIB. Cabe destacar el uso de DMMFRSTO de la clase DMMCODE usado para saber con qué marciano en concreto colisiona, dado que se trata de la interacción entre dos agentes distintos.
- **MARCIANOS:** Dibuja los marcianos y controla su movimiento -que depende de las paredes y de su propia fila- y velocidad. Además, usando el vector

pseudo-aleatorio de la clase UTLCODE de LIB se seleccionan ciertos marcianos para que disparen y otros para que se transformen en marcianos auto-guiados.

- **SHOOTMAR:** Se inicializa un poco por debajo del marciano al que le ha tocado disparar y desciende hasta detectar colisión con el borde inferior o con la nave. Se destruye en ambos casos.
- **MARGUIDE:** Este marciano se inicializa en la posición donde estaba cuando era de tipo “básico”. A continuación, se separa de su fila y comienza a seguir al jugador actualizando su movimiento propio en función de la posición de este.
- **GOVER:** Se muestra cuando las vidas del jugador (y su barra) llegan a 0. Este estado tiene un texto que pone “GAME OVER”, otro que pone “BETTER NEXT TIME” y, por último, un texto que parpadea diciendo “PRESS “X” TO CONTINUE”. Al presionar la tecla “X”, cambia de estado al de INTRO.
- **WIN:** Es una clase no utilizada por el momento, pero está implementada con la finalidad de posibilitar una victoria cambiando el actual sistema de récord de puntuación por otro de lograr una cierta puntuación para ganar. Dichas mecánicas han sido ideas surgidas durante el desarrollo del juego y están orientadas a aportar mayor flexibilidad al código del mismo.
- **SYSCONST:** Contiene los valores de las variables manejadas por el sistema, incluyendo tamaño de la pantalla, constantes de teclado, valores de FPS y números de trap.
- **SYSVARS:** Contiene las reservas de espacio que necesitarán las variables del sistema como el teclado y las interrupciones temporizadas, así como los valores iniciales de los FPS.
- **CONST:** Contiene los valores de las constantes del programa relacionadas con el disparo del jugador y de los marcianos, los botones, la nave, los marcianos básicos y auto-guiados, la puntuación y los estados.
- **VARS:** Define los espacios y valores iniciales de variables del programa relacionadas con la nave, la puntuación, el estado y el ratón.

AÑADIDOS DEL CÓDIGO:

Partimos de un código base proporcionado en uno de los ejemplos del profesorado: el PONG1. En esta versión del clásico juego *Pong* había dos palas que se movían al unísono y una pelota que aparecía en el centro. Si la pelota colisionaba con un borde superior o inferior, rebotaba pero, si llegaba a alguno de los lados, se sumaba un punto a la pala del lado opuesto.

Partiendo de este código, convertimos una pala en nave, otorgándole, además del movimiento vertical que ya tenía, movimiento horizontal para que pudiera moverse por toda la pantalla con total libertad. A continuación, transformamos el código de la pelota para que fuera los distintos tipos de disparo, tanto de marcianos como de la

nave del jugador. El código de la bola también se reutilizó, en parte, para hacer los marcianos, especialmente por su colisión con las paredes.

Por otra parte, el código de los marcianos auto-guiados lo creamos a partir del código de los marcianos básicos una vez estos estuvieron creados.

También añadimos un sistema de estados que permitía una pantalla de inicio (introducción), otra de instrucciones y una de Game Over. Por último, añadimos la interacción con ratón a la pantalla de introducción y un contador de FPS al juego.

IMPLEMENTACIÓN DE LAS CARACTERÍSTICAS ESCOGIDAS:

Hay varias características que nos interesaba implementar.

1. Queríamos que la zona con la puntuación y los FPS, juntamente con la barra de vida, se mostraran siempre por encima de todos los elementos del juego. Por ello, hacemos el plot (del buffer de imagen) de ambas antes que el del resto de elementos.
2. Usando el KBDEGE podíamos hacer que la nave sólo disparara al levantar la mano de la tecla y no al mantener pulsado, pero nos seguía pareciendo que se podía disparar demasiado rápido. Fue por esto que añadimos un temporizador al disparo que limitaba la cadencia del mismo.
3. Dado que resultaba interesante que el usuario viera de forma intuitiva que se detectaban las interacciones del ratón con los botones, hicimos que dichos botones se rellenaran de rojo cuando el ratón estuviera encima. Para ello, comprobamos las coordenadas del ratón en relación a los márgenes del botón y actualizamos el color del mismo si el ratón se encuentra dentro del margen.
4. Para ver los FPS a los que va la partida, sumamos los frames que se imprimen en cada unidad de tiempo y acumulamos en el contador de FPS los que estén en un mismo segundo.
5. Como queríamos que sólo algunos marcianos dispararan y otros se transformaran utilizamos el vector pseudoaleatorio del UTLCODE de LIB. A continuación, recorremos el vector y, si en 32 iteraciones (potencia de 2) toca un número de la lista, seleccionamos el marciano correspondiente para que dispare o se transforme en marciano auto-guiado.
6. Con el fin de evitar problemas por exceso de agentes y también para que el juego no contara con una dificultad demasiado elevada decidimos limitar la cantidad de agentes marcianos que podía haber al mismo tiempo. Para ello, implementamos un contador en el propio código que genera las filas que, al llegar a 31, deja de crear filas hasta que este contador disminuya (algunos marcianos mueran, ya sea por colisionar con el final de la pantalla o con el jugador o uno de sus disparos).

DIFICULTADES:

Por supuesto, a medida que programábamos el juego nos encontramos con algunas dificultades inesperadas que tuvimos que solventar para obtener el mejor resultado posible.

En primer lugar, como estábamos interesados en un control ágil de la nave, esta debía poder moverse hacia todas direcciones y disparar al mismo tiempo. Tuvimos que cambiar la tecla de disparo de *ESPACIO* a *X*, dado que usando la primera tecla la nave no era capaz de disparar mientras se movía arriba a la izquierda o abajo a la derecha.

Hubo también otros aspectos como el movimiento de las filas de marcianos, la barra de vida o la implementación de los estados que costó programar en un principio por la complejidad de realizarlas en el lenguaje ensamblador, pero fueron finalmente logradas.

Además, nos dimos cuenta de que, si había demasiados agentes a la vez (más de 50) el juego dejaba de funcionar. Fue en parte por este motivo que tuvimos que implementar el limitador de marcianos antes comentado.

Por último, cabe decir que quisimos añadir más requisitos de los necesarios para darle mejor fondo a la introducción del juego. En concreto, queríamos cargar una imagen en este estado inicial. Sin embargo, no logramos que esta fuera lo suficientemente ligera como para que el Easy68k la soportara sin dar errores.

CONCLUSIÓN:

Este juego, al estar implementado sobre Easy68K, ha sido programado a muy bajo nivel. Esto hace que sea necesario un código más extenso y, quizá, más difícil de interpretar que en otros lenguajes de programación más habituales. Es por esto que, a pesar de la aparente simplicidad del juego y de las partes reutilizadas del código, su creación ha consumido muchas horas y nos ha obligado a ser bastante cuidadosos durante su programación, tanto en el formato de las instrucciones por encontrarse casi en lenguaje máquina (siendo el lenguaje ensamblador el inmediatamente superior a este) como en los nombres de las variables (\leq a 8 caracteres) y los comentarios para evitar un desorden que habría hecho inviable su corrección de errores por su inteligibilidad.

OPINIÓN PERSONAL:

Tal y como decía el vídeo introductorio del profesor acerca de este proyecto final, programar un juego puede ser incluso más divertido que jugarlo. Sin embargo, hemos de decir que esta afirmación es cierta más bien en los momentos en los que todo sale bien. Cabe destacar que el lenguaje del 68K es bastante arcaico y se mueve a muy bajo nivel, por lo que detectar y corregir los errores de este puede resultar a menudo tedioso.

Si bien es cierto que hemos disfrutado algunos momentos y que resulta satisfactorio tener un juego hecho por nosotros mismos, debemos admitir que este proyecto nos ha ocupado una gran cantidad de horas, pudiendo ir justos de tiempo incluso habiéndolo iniciado a mediados de octubre. Hemos querido hacerlo lo mejor posible, claro está, y es por eso que hemos invertido muchas horas en ello, pero hay que tener en cuenta que esas horas se han consumido a costa de otras asignaturas.