# LEGEND OF STEEL

MAKING OF

TOD STUDIOS

# Tabla de contenido

# 1 Motivación

Desde siempre nos han gustado todo tipo de videojuegos y, además, a pesar de que hoy día la tecnología permite crear gráficos hiperrealistas y juegos cada vez más sofisticados y complejos, nos siguen gustando aquellos que nos han hecho pasar tan buenos ratos desde siempre, y que son los clásicos. Por este motivo, este proyecto se ha presentado ante nosotros tal vez como el que más hemos disfrutado desarrollando en la carrera.

Somos tres estudiantes de 4º curso del grado de Ingeniería Informática en la Universidad de Alicante cursando el itinerario de especialización en computación. Este itinerario cuenta con varias asignaturas, entre ellas una llamada Razonamiento Automático la que imparte el Dr. Francisco José Gallego Durán.

Francisco nos propuso como práctica de la misma, realizar un videojuego en lenguaje ensamblador Z80, con la idea de competir en la CPCRetrodev de este año 2018. Como buenos ingenieros, recibimos encantados el reto por dos sencillas razones: Nos encanta jugar y desarrollar videojuegos, pero sobre todo porque quedaba un mes y medio para la competición, nos gustan los desafíos y este no iba a ser pequeño ;)

# 2 Origen de la idea

Como ya sabemos, Amstrad CPC es una plataforma para la cual hay muchos juegos y muy variados procedentes de su edad dorada allá por los años 80 y principios de los 90. Pero, además, últimamente está volviendo a aumentar su catálogo gracias a las nuevas creaciones realizadas por entusiastas y profesionales de la escena indie/retro. Esto hace cada vez más difícil plantear un tipo de juego que no esté disponible ya en CPC.

No obstante, algo que tal vez se ha echado en falta siempre ha sido un buen dungeon crawler con vista top-down del estilo de Zelda o similares. Así que, inspirados en mecánica de acción similar, y teniendo en cuenta las limitaciones tecnológicas, hemos hecho un diseño de juego que sin duda dará horas de diversión al jugador.

En cuanto a la temática está inspirado más bien en las películas de bárbaros, con un toque de fantasía, donde podemos encontrar de todo, desde orcos hasta poderosos hechiceros, pasando por seres inmortales y extrañas criaturas que harán todo lo posible para impedirnos lograr nuestro objetivo.

# 3 Arte gráfico


Para el desarrollo del arte gráfico, con las restricciones del Modo 0: 160x200 en 16 colores hemos optado por tamaños 8 x 16 para los personajes y 8 x 8 para los items, exceptuando casos especiales como cuando nuestro héroe ataca que pasa a requerir un tamaño de 16 x 16.

Hemos diseñado 3 mundos con 3 tilesets distintos: Mazmorra, caverna y palacio desarrollados en varios mapas de 22 x 20 tiles cada uno. Como se puede observar, los tilesets correspondientes a las cavernas y el palacio son la mitad de tamaño que el de la mazmorra, al final 64Kb son pocos y hay que recortar un poco de todos los sitios.

**Mazmorra:**

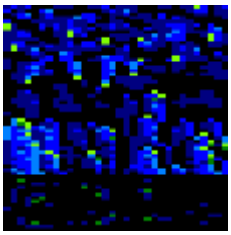
*Tileset empleado*

*Resultado de uno de los mapas generados*


Algo que cabe destacar de este tileset es que, al contrario de lo que sucede en otros juegos similares de 8 bits, los tiles están diseñados para que no se noten las repeticiones de estos, de modo que, por ejemplo, aunque en la imagen hay piedras repetidas, esto no se aprecia a simple vista.

**Caverna:**

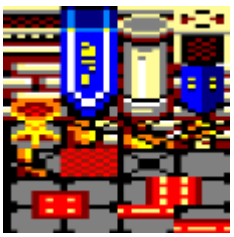
*Tileset empleado*



*Resultado de uno de los mapas generados*

**Palacio:**

*Tileset empleado*



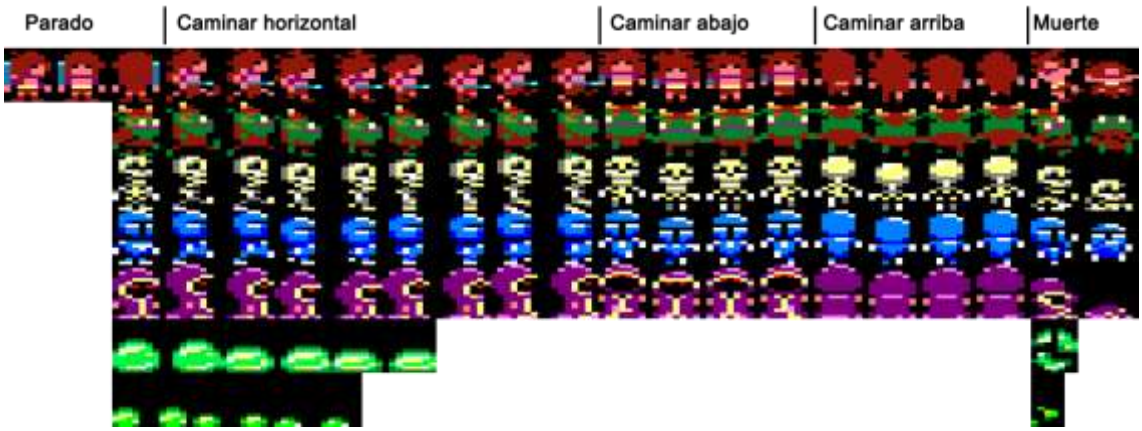
*Resultado de uno de los mapas generados*

## 3.1 Animaciones

La mayoría de animaciones, constan de 4 frames que, en el caso de la animación para correr a izquierda o derecha, finalmente se tuvo que hacer en 8 para conseguir una mayor suavidad en el movimiento (ver apartado “Limitaciones de video”)

**Tilesets de animaciones para todos los personajes**

(Por orden: Héroe principal, orco, esqueleto, guardia, mago, babosa grande, y pequeña)




**Tileset para los items**

Corazón, llave, pollo, monedas, baúl, pócima.

# 4 Música y sonido

Para desarrollar la música y los efectos de sonido hemos utilizado Arkos Tracker. Por comodidad hemos optado por utilizar Arkos Tracker 1 para la música ya que para esta versión, CPCtelera nos proporciona un conversor automático en el propio toolchain de compilación. Para los efectos hemos utilizado Arkos tracker 2 que, aunque nos ha obligado a adaptar los archivos a mano, nos permite separar player de efectos del de la música. La necesidad de esto se verá más adelante en el apartado “trabajar con escasa memoria”.

Hemos empleado un teclado Solton MS60 para dar forma a las melodías antes de escribirlas en el programa. También hemos diseñado la gran parte de los instrumentos que la componen, así como los instrumentos para los efectos de sonido in-game.

Como retos dentro de este proceso cabe destacar la dificultad que hemos encontrado al dar forma a un instrumento intentando que suene lo más parecido a lo que tienes en mente con los ajustes que el programa ofrece. Aunque está bastante documentado es un proceso que requiere de experiencia y así lo íbamos notando en los últimos que diseñamos. En los primeros casos nos hemos basado en instrumentos de ejemplo a los que hemos hecho modificaciones y posteriormente hemos conseguido crear algunos propios desde cero.

En cuanto a retos dentro de la melodía, son infinitos. Al final, componer es un mundo que requiere unos conocimientos y cualidades muy concretas. Dentro de nuestras posibilidades hemos intentado llevar a cabo una melodía interesante y pegadiza sacando el máximo partido a los 3 canales que nos ofrece el AY-3-8912 con el que cuenta el Amstrad CPC 464.

**Finalmente hemos creado 10 distintos efectos para la experiencia in-game:**

- Golpeo del héroe a un enemigo.
- Golpeo de un enemigo al héroe.
- Muerte del héroe.
- Muerte de un enemigo.
- Héroe termina el juego.
- Golpe de espada.
- Recoger monedas.
- Recoger pollo/corazón/pócima.
- Recoger llave.
- Mago lanza bola de fuego.

# 5 Programación

Como decíamos al principio, el juego está desarrollado 100% en ensamblador Z80. Este factor ha sido el principal responsable de la ardua y pronunciada curva de aprendizaje inicial. No obstante, a medida que ha ido avanzando el desarrollo y gracias a la enorme cantidad de ayuda que hay online, hemos podido ir cogiendo un ritmo que nos ha permitido terminar a tiempo todo lo que nos habíamos propuesto.

En cuanto a información para desarrollar para CPC, para aquellos que estéis empezando, recomendamos totalmente los videos del "Curso de ensamblador desde cero" que Fran Gallego tiene en youtube:

https://www.youtube.com/watch?v=smwXc3vShZw&list=PLmxqg54iaXrijQi4-J9IkAWDEguKRX9Dh

Una vez terminados al menos los dos primeros niveles, prácticamente todo lo que necesitamos para hacer un juego como el nuestro está en los vídeos correspondientes a los cursos de "Razonamiento Automático" y "Videojuegos I" de la Universidad de Alicante, impartidos también por Fran Gallego y disponibles en YouTube:

https://www.youtube.com/watch?v=Ojac4Y4sxF0&list=PLmxqg54iaXrjhvQy_GbzYdvURV4sKs0CT

https://www.youtube.com/watch?v=13JGNTWcNLA&list=PLmxqg54iaXrjA0wzZmc_HDeZxs-vDwDFS

https://www.youtube.com/watch?v=k5Z_qLLM6hw&list=PLmxqg54iaXrjtcWWbRjv1JEzjFAZqRywi

## 5.1 El motor CPCtelera

Nada de esto hubiera sido posible terminarlo en un mes y medio sin esta potente librería para Amstrad.

CPCtelera permite desarrollar un juego para CPC en lenguaje C o, si lo preferimos, en ensamblador para tener más control a bajo nivel y poder optimizar al máximo.

Posee todo tipo de rutinas de alto rendimiento para dibujar sprites, mapas de tiles, texto, reproducir sonido y música, recibir entrada (teclado y joystick), varias funciones de manejo de la memoria, compresión, carga de cinta, y un largo etcétera, así como un toolchain que nos permite compilar todo nuestro juego con una sola orden, importando automáticamente todos los assets que utilicemos (imagenes, sonidos, tilesets, etc.)

En nuestro caso hemos utilizado la versión 1.5 que aún está en desarrollo, pero que ofrece nuevas funcionalidades y mejora sustancialmente el rendimiento de la versión anterior.

## 5.2 Características especiales

### 5.2.1 Doble buffer

Los gráficos son enviados a la pantalla por medio del chip CRTC del Amstrad. Este chip hace esto cada 1/50 de segundo haciendo un barrido de toda la pantalla desde arriba hasta abajo.

Para mover un personaje por la pantalla, en cada fotograma el primer paso es borrar el gráfico anterior para luego dibujarlo en la nueva posición. Si hacemos esto sin más puede ocurrir que el CRTC pase en un momento inadecuado. Por ejemplo, podemos borrar antes de que llegue el barrido del chip pero al ir a dibujar el CRTC ya ha pasado y aunque volvamos a pintar, el resultado no será visible hasta el siguiente fotograma.

Esto es lo que causa el famoso parpadeo producido por trabajar directamente sobre la memoria de video. La primera solución para esto es no pintar en cualquier momento, sino cuando el chip CRTC nos envía la señal de sincronización vertical, lo que indica que actualmente se encuentra leyendo en la parte superior de la pantalla.

Esto soluciona buena parte del problema pero si tenemos muchos gráficos nos puede pasar otro efecto peor aún y es que por la parte superior los gráficos no se vean nunca, ya que el retrazado gana siempre en velocidad al procesador.

La solución para esto es usar doble buffer. Esta técnica consiste en tener dos buffers de video. Uno será el front buffer (o buffer actualmente visible) y el otro el back buffer (o buffer oculto).

El front buffer será lo que veamos en pantalla pero siempre dibujaremos en el back buffer y, cuando hayamos terminado las tareas de gráficos le diremos al chip CRTC que intercambie los punteros de estos dos buffers para que se muestre lo que acabamos de modificar. En este momento lo que era el front buffer pasa a ser backbuffer, y será en este donde prepararemos el siguiente fotograma mientras se muestra el actual. Esto se repetirá en bucle y, como resultado los parpadeos y sprites que desaparecen ya nunca más serán un problema, a costa, eso si, de una gran cantidad de RAM utilizada.

### 5.2.2 Scroll por hardware

Como hemos visto en el anterior apartado, el chip CRTC nos permite modificar el puntero que indica la posición de memoria que queremos usar como memoria de video. Esto en Amstrad puede aprovecharse para hacer un scroll por hardware.

Si aumentamos la posición del puntero en dos bytes (por limitaciones del hardware sólo podemos mover de dos en dos), el efecto será que toda la pantalla se moverá 4 pixels (estando en modo 0) hacia la izquierda. Y si lo reducimos su posición en dos bytes, haremos el mismo efecto pero hacia la derecha.

El movimiento en vertical se consigue variando el puntero esta vez +80 bytes para mover hacia arriba o -80 bytes mover abajo.

Por supuesto, no solo basta con mover el puntero. Además de esto deberemos pintar la nueva fila o columna de tiles correspondiente para efectuar el efecto de scroll.

### 5.2.3 Ruptura vertical

Además de cambiar el puntero de vídeo, el chip CRTC nos permite hacer muchas más cosas. En nuestro caso lo hemos utilizado para aplicar un efecto mítico en Amstrad, conocido como ruptura vertical.

El objetivo de este efecto es dividir la pantalla con una línea imaginaria horizontal donde podemos mostrar simultáneamente dos buffers de video, uno en la mitad superior y otro en la mitad inferior. De hecho, se podrían hacer más divisiones e incluso hacer que se pinte en la zona del borde de la pantalla haciendo que haya más zona visible. El problema es que esto también aumentaría la memoria necesaria para el video.

Para nuestro juego simplemente hemos necesitado hacer dos zonas: una grande arriba (de 176 pixels de alto) para la escena del juego y otra más pequeña abajo (de 24 pixels de alto) para el panel.

El principal problema de este efecto es que, para conseguirlo, debemos realizar las operaciones de cambio de buffer en el momento exacto y sin saltarnos en ningún momento el proceso desde que lo iniciamos por primera vez, invocando permanentemente a estas rutinas de manera perfectamente sincronizada en cada fotograma durante todo el juego.

Esto puede parecer demasiado complicado (y de hecho no es sencillo al principio), pero en Amstrad tenemos la ventaja de disponer de un sistema de interrupciones que, preparándolo convenientemente nos permite asignar una rutina que se ejecutará a intervalos regulares y en momentos predecibles. Concretamente, el CPC lanza 6 interrupciones a cada 1/50 de segundo. Esto podemos aprovecharlo para hacer los cambios de buffer de los que hablamos ya que cada una de estas 6 interrupciones corresponderá a un momento en el que el barrido del CRTC está en una posición concreta de la pantalla. Ya es decisión nuestra a cuál esperar para hacer la división.

## 5.3 IA de los enemigos

### 5.3.1 Sensores para detección de proximidad con los muros

Los enemigos caminan por todo el mapeado, pero cuando deben seguir una ruta determinada y tienen una pared delante, no avanzan hasta colisionar con ella, sino que implementan una inteligencia que les hace mantener cierto margen con las paredes. Esto es lo que haría cualquier persona en el mundo real… y es que en la realidad no vamos chocándonos con las paredes cuando caminamos… ¡o al menos no intencionadamente!

Para ello, todos los enemigos tienen un sensor de proximidad. Dicho sensor comprueba antes de realizar cada movimiento si está próximo a algún obstáculo, en dicho caso, se activan los estados correspondientes para modificar la dirección del enemigo según a donde se dirija.

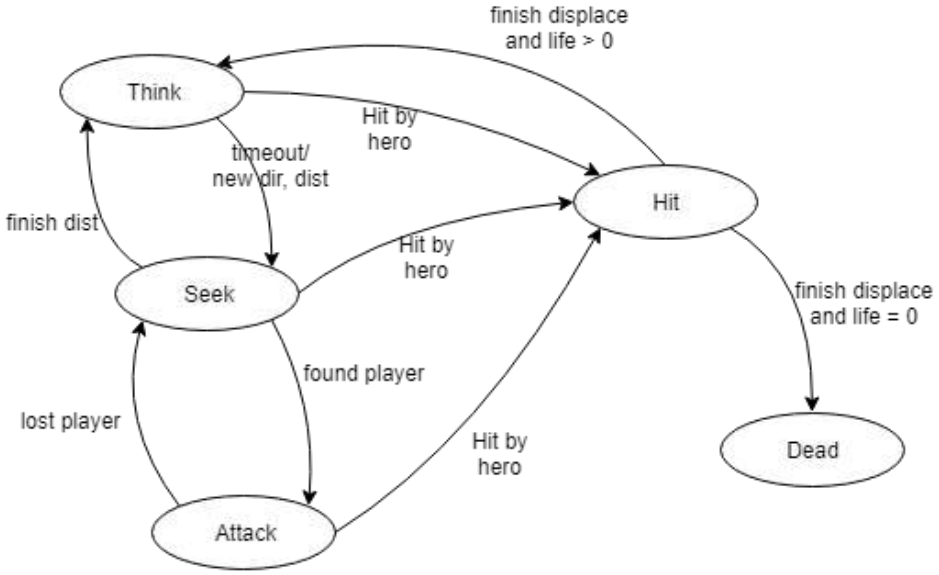
### 5.3.2 Búsqueda de caminos

La rutina básica de un enemigo consistirá en caminar aleatoriamente por el escenario hasta que vea al jugador. Un enemigo no puede ver a través de los muros, por tanto, no vamos a necesitar aquí un complejo algoritmo de búsqueda de caminos, ya que como mucho su objetivo será un punto visible desde su posición.

Una vez que el enemigo ha localizado al jugador irá tras él tal como se explica en el siguiente apartado.

### 5.3.3 Las máquinas de estados

En cuanto a la inteligencia de los enemigos, se ha diseñado como base un comportamiento común a todos ellos del cual se añaden las variaciones necesarias en función del tipo de enemigo. Para dicho comportamiento base, se ha creado una máquina de estados tal y como se muestra en la siguiente imagen:




Como se puede observar, existen 5 estados base, **Think, Seek, Attack, Hit** y **Dead**.

- El estado **Think** escoge una dirección aleatoria teniendo en cuenta que pueda moverse hacia esa dirección, para ello utiliza un sensor que detecta la proximidad a las paredes y que posteriormente se explicará. Además de la dirección también coge una distancia aleatoria que recorrer.

- El estado **Seek**, mueve al enemigo utilizando la dirección y la distancia escogida en el estado **Think**. Si durante el movimiento encuentra con el sensor alguna pared, vuelve al estado **Think** para escoger otra dirección. Con esto se evita que los enemigos estén chocándose continuamente con los muros. Por otro lado, si detecta al héroe a menos de 8 casillas distancia, mediante la distancia de Manhattan cambia al estado **Attack**. Es importante destacar que, si el héroe está a menos de 8 casillas, pero hay algún muro delante, ignora al héroe y mantiene el estado **Seek**. Por último, una vez terminada de recorrer la distancia escogida en el estado **think**, vuelve a dicho estado para elegir otro camino nuevo.

- El estado **Attack** sigue al héroe mediante el camino de Manhattan y, si lo perdiera de vista en algún momento vuelve al estado **Think**. Además, comprueba si ha colisionado con el héroe. En ese caso, pone el estado del héroe en **Hit** y pasa a estado **Think**.

- El estado **Hit** desplaza 3 tiles y decrementa la vida la del enemigo. En caso de quedarse sin vida pasaría al estado **Dead**, en caso contrario pasaría al estado **Think**.

- El estado **Dead** únicamente realiza la animación de muerte y se queda permanentemente en dicho estado.

Todos los enemigos heredan el comportamiento base anteriormente descrito, pero cada uno tiene algunas variaciones para conseguir un comportamiento diferente.

- El **esqueleto** refina el estado **Dead** añadiendo un contador de 4 segundos que, cuando finaliza vuelve al estado **Think**, es decir, que "resucita" ya que es un esqueleto y no puede morir.
- El **caballero** no tiene refinamientos en cuanto a estados, pero, para conseguir reducirle la velocidad actualizamos su lógica de estados cada 2 veces, esto genera una reducción de la velocidad a la mitad respecto al resto de enemigos.
- El **mago** tiene un refinamiento del estado **Attack** que consiste en una nueva máquina de estados dentro de dicho estado. Los estados de ésta son **Escape, Align** y **Throw**.
    - El estado **escape** se activa si el héroe se encuentra a menos de 4 casillas de distancia y hace que el mago escape del héroe por el eje que menos alineado esté de él.
    - El estado **align** se activa si el héroe se encuentre entre 4 y 8 casillas de distancia y lo que hace es mover al mago en el eje que más alineado esté para alinearlo con el héroe ya sea vertical u horizontalmente. Esto es para buscar una posición de tiro hacia el héroe (ya que este enemigo lanza bolas de fuego).
    - El estado **throw** se activa si el héroe se encuentra entre 4 y 8 casillas de distancia y se encuentra alineado en alguno de los ejes y lo que hace es lanzar una bola en la dirección del héroe.
    - Por último, si se encuentra a menos de 2 casillas de distancia del héroe se activa el estado **Attack** base que anteriormente se ha explicado.
- La **babosa** tiene los siguientes cambios:
    - En el estado **Dead**, en lugar de morir, crea 3 babosas más pequeñas, que a su vez al pasar al estado **Dead** mueren definitivamente.
    - Por otro lado, tiene un estado **Sticked**. Dicho estado se activa si colisiona con el héroe reduciéndole la velocidad (siguiendo la misma técnica que en el caballero) y tiene un contador en el cuál cada segundo quita vida al héroe. Para salir del estado **Sticked** el jugador debe moverse a izquierda y derecha rápidamente.

## 5.4 Trabajar con escasa memoria

El Amstrad CPC464 para el que se ha diseñado este juego sólo tiene 64Kb de RAM distribuidos por defecto de la siguiente forma:

Esto es un verdadero problema si se pretende hacer un juego de estas características con gráficos bien variados y con múltiples tipos enemigos con lógicas diferenciadas. En nuestro caso, como ya se ha visto, tenemos 3 grandes tilesets de los cuales se genera un buen número de mapas, el juego tiene 3 fases, 5 tipos de enemigos + el personaje principal, y 6 tipos de ítems.

A todo esto, hay que añadir un menú rico en gráficos, incluyendo un vistoso tutorial antes del juego y un tema principal de un minuto y medio de duración.

Además, debemos tener en cuenta el código que mueve todo esto y las funciones necesarias de la librería de CPCtelera que lo hace posible. A priori puede parecer inviable que todo esto vaya a caber en 64Kb, y es que, dado el nivel de detalle de los gráficos, ni siquiera las rutinas de compresión lograban reducir el espacio ocupado de forma sensible.

Pero es que, si todo esto aún nos pareciera poco problema, aún tenemos que tener en cuenta que el juego utiliza doble buffer para el renderizado fluido y sin parpadeos de los gráficos.

Para quienes no lo sepan, la memoria de video del Amstrad forma parte de la RAM principal. Para mantener los gráficos en pantalla, debemos tener 16Kb reservados de los 64Kb disponibles, con lo cual nos quedarían 48Kb, pero utilizando doble buffer necesitamos 32Kb para video y sorpresa… ¡Sólo nos quedan 32Kb para el juego!

Si hablamos de números reales, nuestro juego en realidad ocupa más de 45Kb, con lo cual a priori no cabe entero.

### 5.4.1 Primera solución

La solución ha sido, cargar el menú en la zona del doble buffer y, durante esta parte del programa, sólo se dibujará directo a la memoria de video (sin doble buffer). Una vez comienza el juego, se sobreescribirá el código y los gráficos del menú con el back buffer. Ya no vamos a volver al menú principal una vez iniciado el juego, ya que, en caso de que quisiéramos reiniciar la partida o continuar desde un punto de guardado, se muestra una opción para ello al finalizar.

Esto nos ha dado mucho más espacio disponible pero, aun así no ha sido suficiente, porque al final, sólo el bloque del juego sobrepasa por muy poco los 32Kb de los que disponíamos.

### 5.4.2 Recuperar aún más espacio (y sin recortar contenido)

Los mapas de tiles que se muestran están comprimidos y lo que se hace en cada momento es descomprimir en una zona reservada aquel que se va a mostrar.

Pues bien, para ahorrar el espacio que nos faltaba para entrar en los 32Kb lo que se ha hecho es utilizar esta zona para las rutinas y funciones de CPCtelera que sólo se utilizaban durante la inicialización del sistema y del juego (todas las que sólo se usan una vez). Así, cuando el juego empieza, reaprovecha esa zona para descomprimir los mapas, con lo cual esto queda implementado sin coste de memoria.

Finalmente el juego ha cabido, ¡pero sólo nos han sobrado 14 bytes! … de modo que la memoria ha quedado más que bien aprovechada ;-)

## 5.5 Trabajar con capacidad de proceso limitada

El CPC dispone de un procesador Z80 que funciona aproximadamente a 4Mhz esto limita lo que se puede hacer pero, aun así, como ya sabemos todavía se pueden hacer muchas cosas.

Nuestro juego logra funcionar a 50 fps con hasta 9 entidades moviéndose en pantalla. Dicho framerate tampoco se ve reducido al hacer scroll, ya que éste ha sido implementado por hardware (tal como se ha explicado anteriormente).

En ocasiones, el número de entidades en la escena puede hacer que nos saltemos un fotograma. En este caso, al estar sincronizados con el retrazado del monitor, el resultado es que bajamos directamente a 25fps. Sin embargo, esto no provoca que los movimientos se vuelvan lentos en el juego, sino que esta situación se detecta y los incrementos en las posiciones se duplican automáticamente para ajustar la velocidad del juego a los fps actuales.

Con esto se consigue que la velocidad del juego sea independiente de la carga que tenga que ejecutar.

## 5.6 Limitaciones del sistema de video

El modo de video elegido para los gráficos ha sido el modo 0 ya que, aunque tiene menos resolución nos proporciona más colores (en total 16). En dicho modo, cada byte de la memoria de video representa dos píxels. Esto es un problema, ya que cuando movemos un sprite una posición en horizontal, no se mueve un píxel sino dos.

Hemos querido pulir este detalle consiguiendo que se mueva pixel a pixel a costa de consumir más memoria.

Para solucionarlo, lo que hacemos es, para las animaciones de caminar en horizontal de los personajes, en lugar de usar 4 fotogramas como las demás, usamos 8. Los 4 fotogramas extra son copias exactas de los anteriores, solo que movidos un pixel a la derecha. De esta forma, cuando movemos un personaje, vamos alternando estos frames duplicados de forma que, aunque el grafico se mueve de dos en dos en memoria, nosotros lo percibimos como si el movimiento fuera de uno en uno.

## 5.7 Ampliaciones en la funcionalidad de CPCTelera

### 5.7.1 Dibujado de mapas de 4x8

La versión 1.5 de CPCtelera dispone por primera vez de una rutina que dibuja tilemaps con tiles de 4x8 bytes (la anterior estaba limitada a 2x4). Con esta función se puede pintar mucho más rápido el mapeado ya que necesita muchos menos cálculos.

Sin embargo, tiene una limitación y es que, durante el pintado de cada fila, deshabilita las interrupciones. Esto es un problema importante para nosotros, ya que nuestro juego utiliza una ruptura vertical (explicada anteriormente) que requiere una actualización constante para mantenerla, o de lo contrario, se desincronizará la ruptura y los gráficos se verán dando vueltas en la pantalla.

Para solucionar esto, hemos implementado nuestra propia rutina para pintar tiles de 4x8 que, aunque es un 25% más lenta que la de CPCtelera no interfiere en las interrupciones.

### 5.7.2 Dibujado de tiles parciales

Nuestro scroll avanza en horizontal a una velocidad de 4 en 4 pixeles. Sin embargo, los tiles tienen un ancho de 8.

Al mover un paso de scroll (4 pixels), necesitamos pintar una nueva columna de tiles en el lateral de la pantalla, pero al ser de 8, sobresalen de la pantalla y aparecen por el lado opuesto, lo cual es un efecto no deseado.

La solución ha sido crear dos nuevas rutinas, las cuales, a partir de un tile de 8x8 pixeles, una pinta la mitad izquierda y otro en la mitad derecha. De esta forma, en cada paso del scroll, vamos alternando llamadas a estas dos funciones para formar el tilemap en orden el cual aparecerá de 4 en 4 pixeles.

### 5.7.3 Carga de disco desde ensamblador

CPCtelera tiene una función para cargar datos de cinta desde ensamblador, pero esto no sirve para disco.

Debido a la disposición de memoria explicada anteriormente, necesitamos cargar el bloque del menú por código y esto sólo es posible hacerlo una vez el programa se ha iniciado ya que es necesario deshabilitar previamente el firmware. Por tanto, hemos implementado también una función para cargar desde disco y así, nuestro juego es compatible con toda la gama de CPC

### 5.7.4 Dibujar texto con sprites

La función para dibujar texto de CPCtelera utiliza para ello la tipografía original de la ROM. Sin embargo, en modo 0 estas letras quedan demasiado grandes para nuestro juego y además queríamos una tipografía distinta con posibilidad de tener varios colores simultáneos.

Por este motivo hemos realizado también una rutina que, a partir de una cadena de texto y un tileset con los caracteres ASCII, escribe en pantalla el texto correspondiente. La siguiente imagen corresponde al tileset de texto usado en el juego:

### 5.7.5 Reproducción de efectos de sonido con Arkos Tracker 2

Debido a las restricciones de memoria, el juego incluye el tema principal de la música en el menú principal, pero no ha podido entrar una música in-game. Además, sólo el player ocupa 2Kb.

Dicho player no puede separarse del de los efectos de sonido en la versión de Arkos Tracker 1, con lo que a pesar de que no lo íbamos a usar, seguiría ocupando 2Kb sólo para los efectos.

Sin embargo, Arkos Tracker 2 sí que permite separar estos dos bloques, siendo solamente 500bytes lo que ocupa el player de efectos de sonido.

Por esto hemos hecho una función que, siguiendo la misma sintaxis de CPCtelera reproduce sonidos utilizando el player de Arkos Tracker 2. Y de esta manera, tenemos música y efectos durante el menú, y sólo efectos durante el juego con 2Kb de ahorro en memoria.

No obstante, el juego sí que tiene melodías cortas hechas editando los instrumentos, variando el pitch para crear las notas. Esto nos ha permitido incluir la melodía de game over, y otras que aparecen en determinadas situaciones.

# 6 Herramientas utilizadas

- CPCtelera (Motor de juegos)
- Gimp (Diseño de gráficos)
- Arkos Traker 1 (música)
- Arkos Traker 2 (efectos sonido)
- Ubuntu 18.10
- Tiled (Diseño de mapas)
- Trello (organización de trabajo)
- Github (control de versiones)

# 7 Conclusión

Una vez concluido el trabajo y haciendo una reflexión sobre el resultado, la satisfacción es muy grande. Vale la pena aclarar que cuando hablamos de resultado no nos referimos exclusivamente a lo que se puede ver.

Para nosotros el reto fué un gran desafío, era un proyecto ambicioso y con una cota de tiempo muy ajustada. Hemos tenido que sacar horas de cualquier sitio para conseguir "terminar" a tiempo. Entrecomillar la palabra terminar puesto que hemos tenido que ir dejando muchas ideas en el cajón a medida que avanzaba el proyecto, pero había que ser realistas. Teníamos una fecha muy próxima en la que necesitábamos un producto listo para producción y no podíamos permitirnos que no fuese así. También es cierto que ninguna de las funciones que no hemos podido implementar era de las consideradas como indispensables, las cuales se han completado al 100%.

Aparte de la experiencia a nivel de desarrollo de proyecto y trabajo en equipo, todo lo aprendido a nivel técnico ha sido de lo más interesante. Trabajar en un hardware tan limitado nos ha llevado a optimizar nuestro código al byte e inventar todo tipo de "triquiñuelas" con tal de arañar unos bits o unos ciclos de reloj en las rutinas de peso.

No sabemos qué nos deparará el destino, pero estaría feo negar que la experiencia nos ha hecho pensar en futuros proyectos de cara a la CPCRetrodev'19 y creo que en resumen esto lo dice todo :)