Deep, Deeper, Deepest

“En un futuro cercano, ricos y pobres están más separados que nunca. Un programa de TV de dudosa integridad es la última esperanza para muchos. “Deep, Deeper, Deepest” donde el premio es tu vida.

Atrévete a entrar en este laberinto mortal y lucha contra otros concursantes para conseguir todo el dinero que puedas. Hazte con las llaves de color para adentrarte más y salir con vida.”

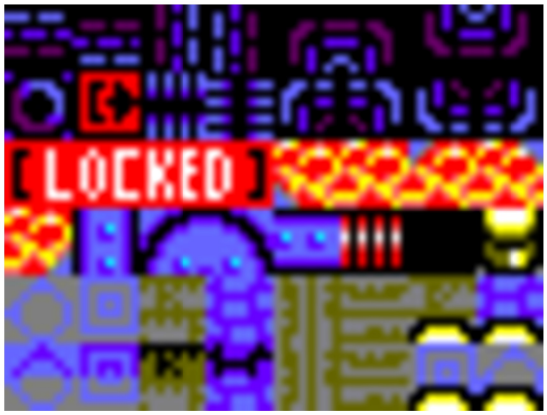
Es un juego de habilidad, cada habitación que entremos será un desafío para no morir antes de lo esperado. Numerosos enemigos impedirán nuestro progreso. Para ayudar a los concursantes la dirección irá dejando ciertos objetos en unos lugares determinados. Podremos recoger monedas para superar nuestra puntuación, relojes para que se acabe antes la habitación, o los más deseados: poder parar los rivales o inmunidad temporal.

El laberinto se compone de 72 habitaciones, repartidas en 3 áreas. Para pasar cada área hay que encontrar primero la llave que dé acceso a la siguiente área. Cada área se compone de 24 habitaciones. La puerta de salida al exterior del laberinto estará custodiada por un jefe final.

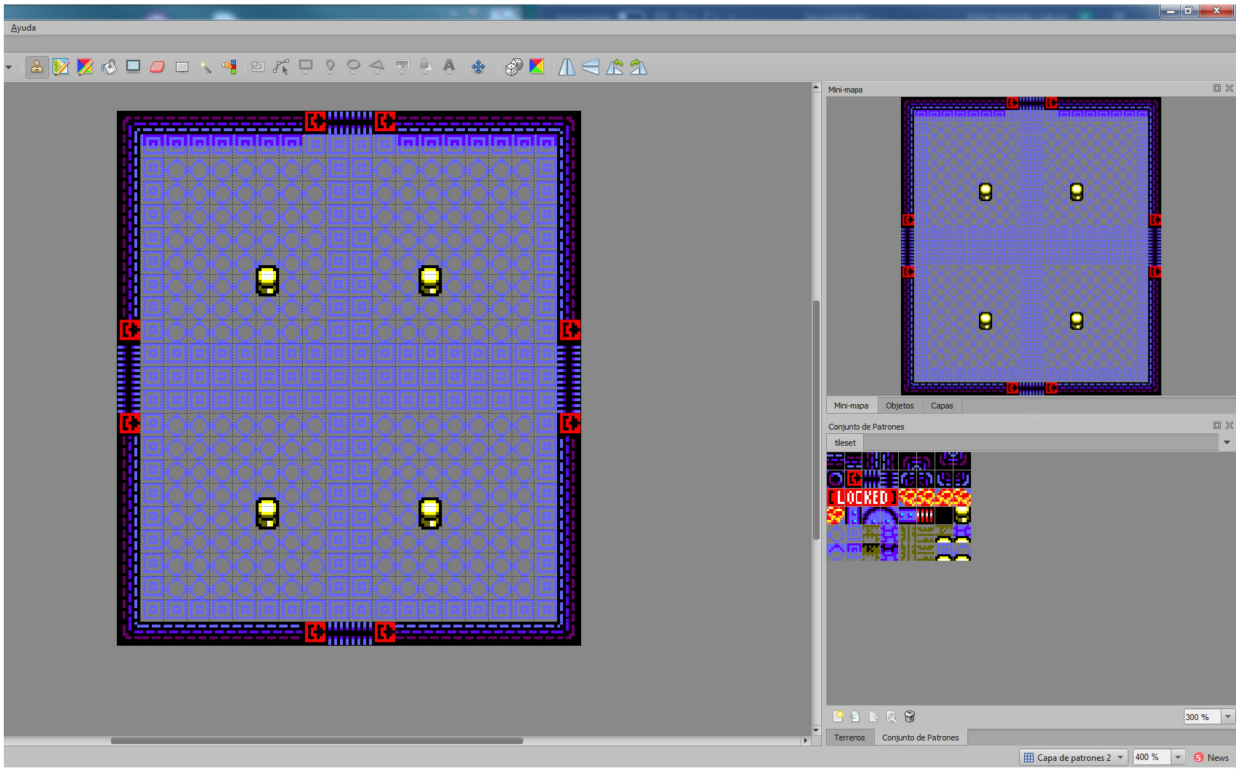
Como características técnicas más resaltables del juego podemos destacar:

- Dado que vamos a tener varias entidades en pantalla y el área de juego es prácticamente la pantalla completa, se hace uso de la técnica del **doble búfer de vídeo** para evitar parpadeos. Con esta técnica podemos preparar el siguiente fotograma en una parte de la memoria que no está leyendo el CRT. Cuando esté listo, existe una instrucción inmediata para cambiar el CRT a que lea la zona de memoria ya preparada.

- Uso de **Tiles** (baldosas o mosaicos). CPCtelera dispone de utilidades muy sencillas para dotar a nuestros juegos de escenarios de manera muy rápida y fácil. Para ello primero necesitaremos de una imagen que contenga todos los posibles mosaicos:



Posteriormente con la herramienta Tiled marcamos la división de cada celdilla 8x8. Y Vamos construyendo la pantalla haciendo uso de estas celdillas:

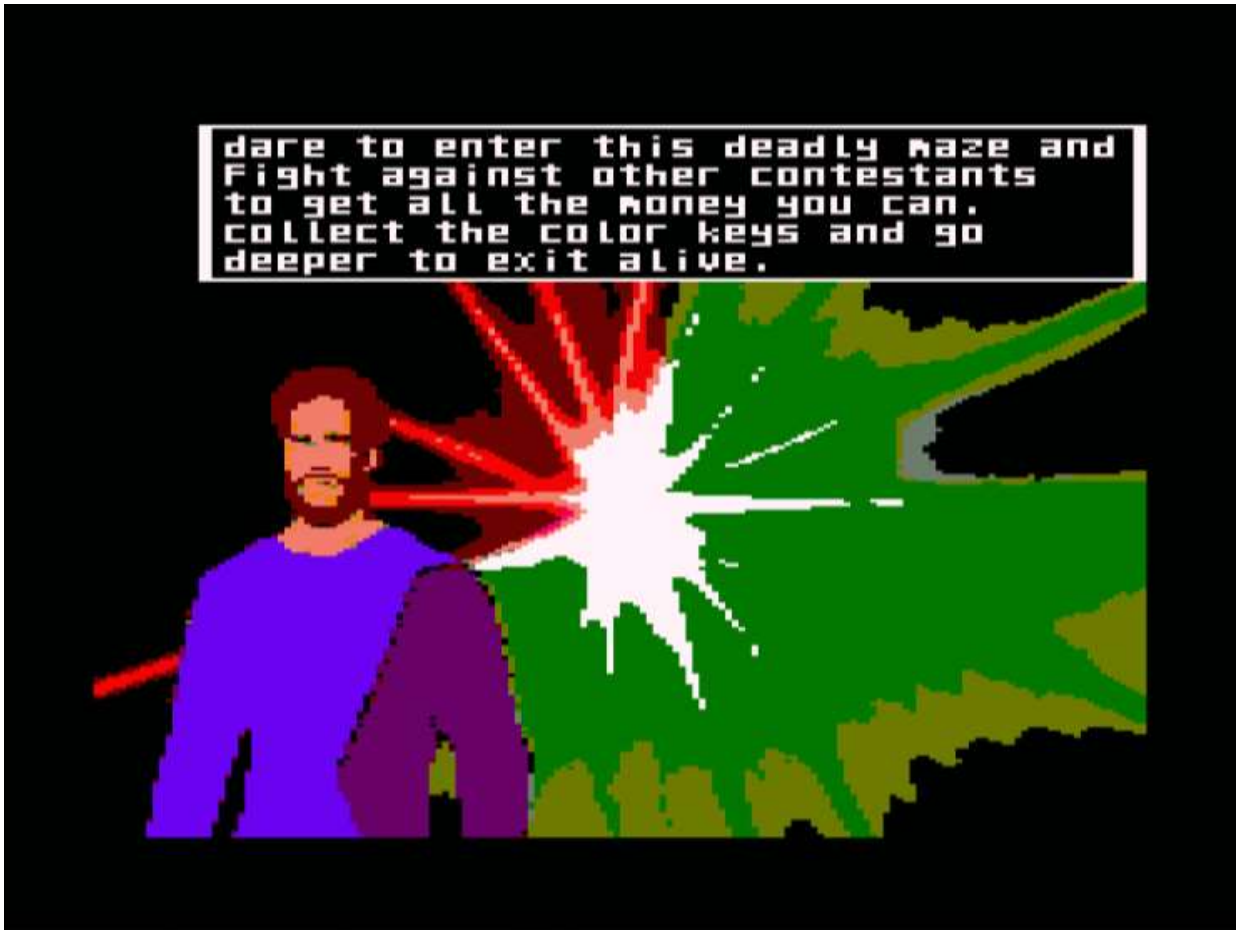


Para el juego hemos realizado 24 diseños (más una para el jefe final). En total el juego tiene 72 pantallas por lo que cada diseño se repite 3 veces, pero la repetición se produce cuando el jugador avanza bastante. Además, se evita la repetición porque la configuración de enemigos siempre es diferente (aunque se repita el diseño).

Los tiles pueden ser hasta de tres tipos: tendremos tiles letales que no podrá tocar el protagonista (quitan energía); los sólidos, que no podrán ser atravesados por las entidades; y los transparentes, que forman el suelo (o escenario).

- Para la música y FX hemos decidido no usar Arkos Tracker 1 que viene integrado con CPCTelera. Porque la herramienta para componer las melodías es bastante inestable con los sistemas operativos actuales. En vez de eso, usamos **Arkos Tracker 2**, cuya herramienta de composición es estable y da más facilidades. Para integrarlo con cpctelera, usamos parte del código de Caye y Tonet de BiteStudios (autores de Promotion)

- Aprovechamos la zona de memoria que se usará de doble buffer (de 0x8000 hasta 0xC000) para hacer una pequeña **introducción** al principio, dado que en ese momento no es necesario el doble búfer. Esto hará que esa introducción sólo se podrá ver tras la carga del juego, pues posteriormente esa zona se limpiará y se usará para el doble búfer.



- Para las imágenes completas que se ven en pantalla (hasta 3) usamos **compresión** (la que incluye cpctelera). Así podemos ahorrar memoria, pues en vez de ocupar 16 KB cada uno, ocupan sólo entre 1 y 3 KB. Las imágenes se descomprimirán directamente a la memoria de vídeo por lo que no es necesario reservar ningún espacio en la memoria.

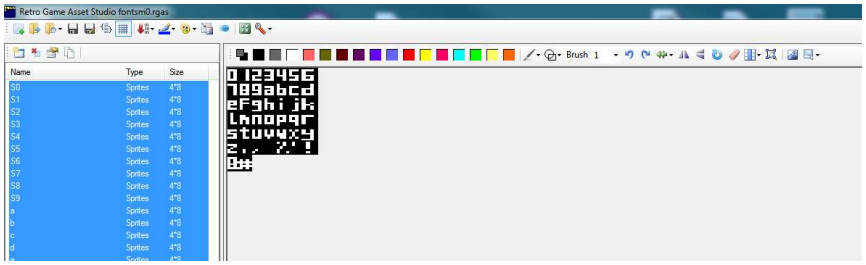
  También usamos la herramienta de compresión para los diseños de escenarios comentados más arriba; pasando de 460B a unos 130B. No muy significativo para 1, pero sí cuando hacemos uso de hasta 25 diseños.

- Para añadir dificultad en ciertas habitaciones tienen un cambio de paleta dinámico, para simular como un parpadeo de luces. Esto desconcentra algo al jugador y hace menos visibles los enemigos y disparos. Este efecto se usa en las habitaciones que sirven de cambio de área (las que hay que llegar con la llave correspondiente)

- Algo que queríamos que fuese una característica del juego es la variada Inteligencia Artificial de los enemigos. O lo que es lo mismo que fuesen muy variados en su comportamiento. Hemos programado varios patrones, siendo algunos de ellos parametrizables para que así sean más difíciles de evitar conforme más nos adentremos en el laberinto

    - Movimiento básico por eje X,Y o ambos (con o sin disparo)
    - Movimiento Aleatorio (cada cierto tiempo cambia)
    - Perseguidor (cada cierto tiempo se acerca dónde está el protagonista). Con este comportamiento hay varios diferentes, los que parten de una posición estática, los que persiguen sin descanso o los que están patrullando un área hasta que detecten al protagonista
    - Cañón en laterales del muro que lanza un láser
    - Pinchos que surgen del suelo cerca de la posición del protagonista
    - Postes que cada cierto tiempo crear una barrera letal
    - Para darle algo de atractivo gráfico también hemos añadido unos rivales de aspecto similar al protagonista, como si fueran otros concursantes de este espectáculo. Uno de ellos irá a por los ítems en pantalla; mientras que otros nos perseguirán a la misma velocidad que nosotros. Por suerte se cansa pronto y tendrá que pararse.

- El texto por defecto para modo 0 ocupa bastante y sería muy complicado añadir frases en pantalla. Por tanto usamos unos procesos propios para escribir texto en pantalla que hace uso de caracteres de 4 pixeles de ancho. Es menos eficiente (porque ocupa código) pero podemos escribir textos legibles sin problema.



- Existe una serie de vídeos en los que se explican algunas de estas técnicas y la estructura principal que tiene el juego. Por si es de su interés visualizarlos: https://www.youtube.com/watch?v=heV62RyM0dM

- Rejugabilidad: Son 3 laberintos aunque al principio parecerá que es uno distinto a cada nueva partida, hasta que el jugador se dé cuenta. También el hecho que explorar el 100% te da puntos extra para competiciones de puntuación.

A lo largo del desarrollo la mayor problemática han sido 2 que son las habituales al tratar con máquinas de 8 bits:

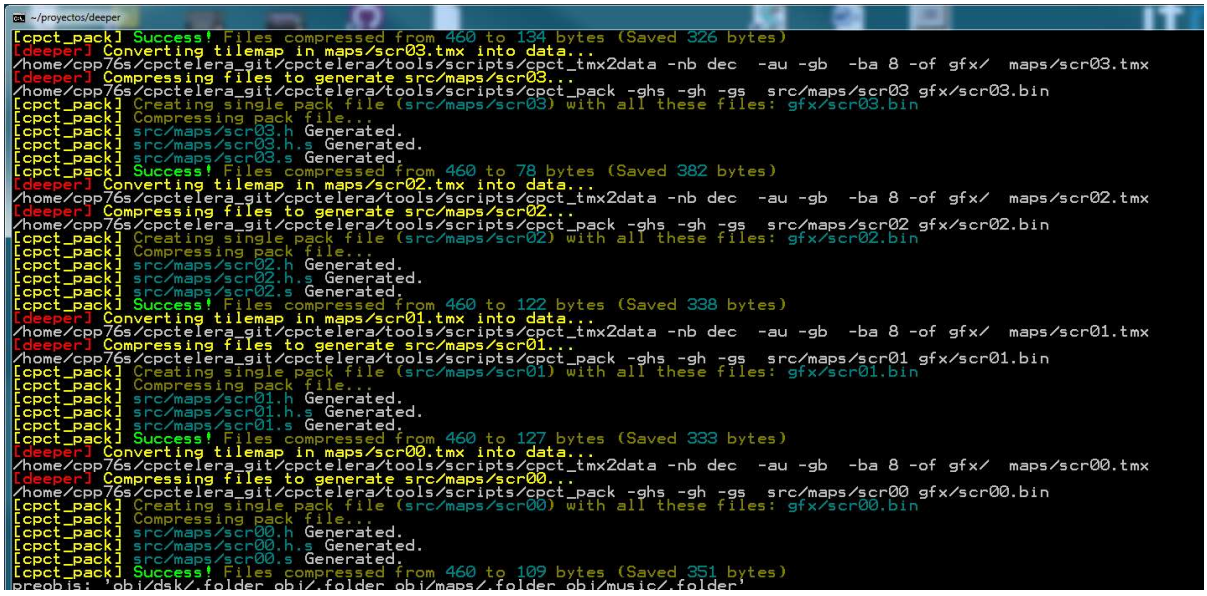
- Velocidad de CPU, el Hardware del Amstrad CPC no se diseñó específicamente para videojuegos, pues tiene pocas facilidades en ese aspecto, al menos de fácil uso. A diferencia de otros sistemas, no dispone de sprites por hardware ni ninguna ayuda para tratar fondos (y luego restaurarlos). Así como tampoco mezclar modos gráficos en la misma horizontal; sólo es posible en distintas áreas de la pantalla.

  En el sistema Amstrad todo el dibujado de gráficos debe pasar por la memoria, es decir, de los 64 KB se reserva una parte (normalmente 16KB) donde se escribe lo que se quiere pintar. El CRTC irá periódicamente leyendo de ahí y mostrando en pantalla. Con esta manera de trabajar puede ocurrir que el CRTC esté pintando en pantalla una zona de memoria que estemos actualizando o borrando, pues CRTC y CPU trabajan de manera independiente (no se esperan el uno al otro). Si ocurriese esto (que suele ocurrir cuando hay muchos gráficos a pintar) se producen los efectos de flickering (parpadeo) o tearing (parte del gráfico está en un fotograma y otro desplazado para el siguiente fotograma).

  Para evitar este problema hacemos uso del doble búfer que hemos comentado antes. Sacrificamos otros 16KB de memoria para tener 2 zonas de vídeo. Y mientras se está pintando una, vamos preparando en la otra el siguiente fotograma.

- La otra limitación suele producirse a la hora de estar terminando el proyecto, que es la memoria. Tras leer el párrafo anterior se deduce que sólo dispondremos de 32KB para nuestro juego (pues los otros 32KB estarán reservados para las memorias de vídeo). Para intentar que entre lo máximo posible hacemos uso de varias técnicas:

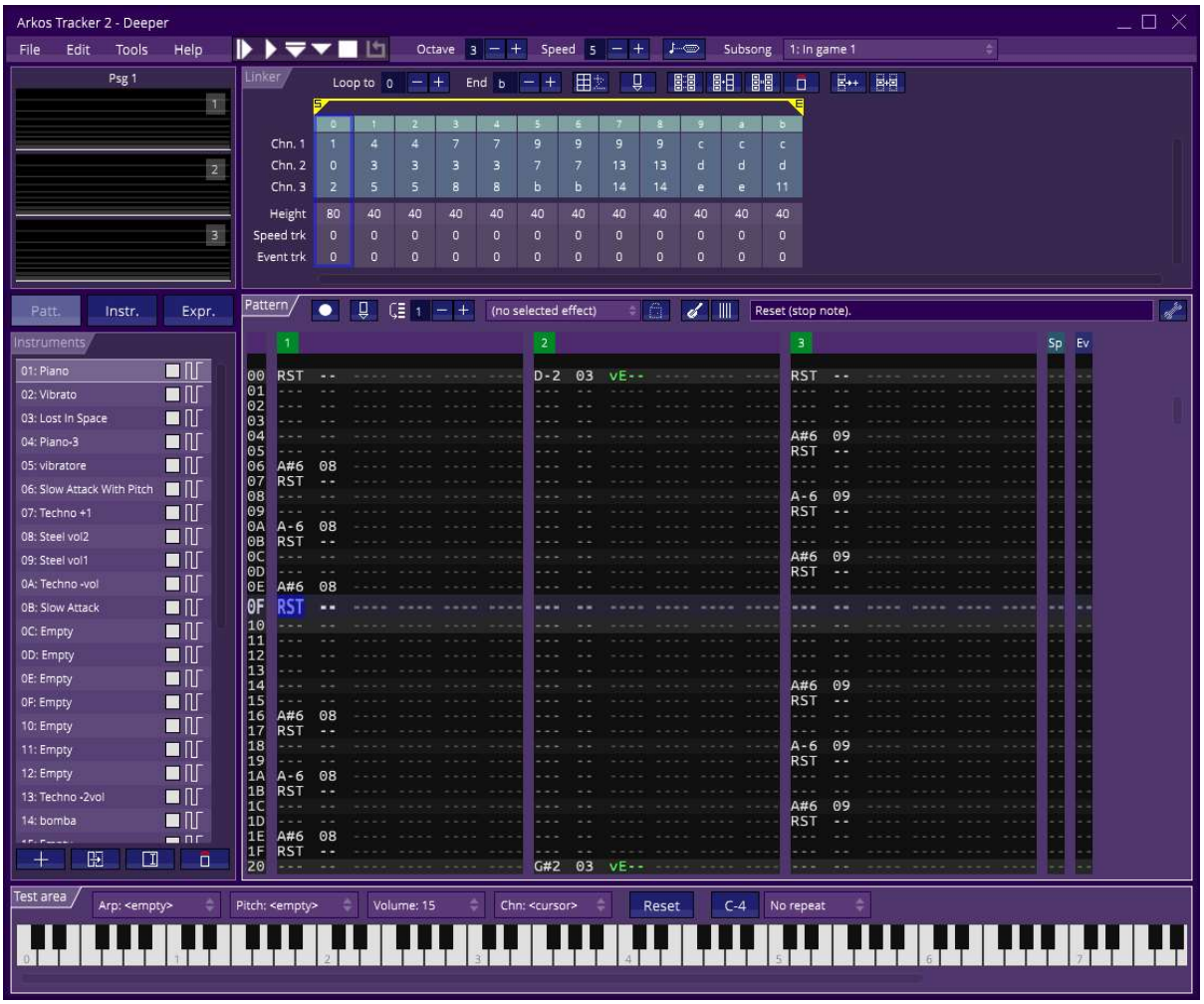
  Compresión, algunas imágenes (la del menú) como los diseños de los escenarios tienen elementos que se van repitiendo (color de pixeles o tiles usados). Esto lo hace propicio para que pasen por un proceso de compresión y así ahorrar bastantes bytes. El único inconveniente de esta compresión es que luego necesita de una descompresión, que es más lenta que leer los bytes sin comprimir. Pero esto no afecta en nuestro juego pues esa descompresión la hacemos siempre que el juego no está en marcha como al inicio de la partida o al cambiar de pantalla.



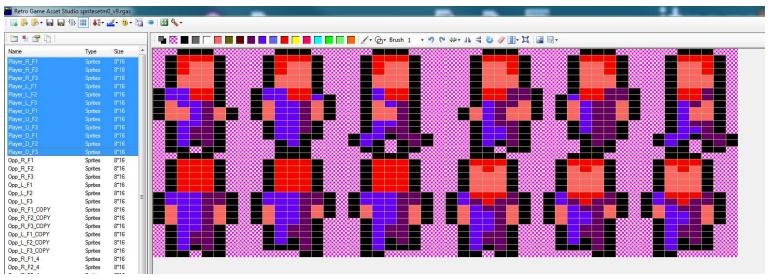
Factorización de código, hay ciertas partes del código que se repiten mucho, como inicializar la posición de una entidad o resetear las direcciones de la misma. Como este código se repite en varias ocasiones, hemos creado una función que realiza estos pasos. Así en vez de tener el código repetido varias veces (con su consiguiente gato de memoria), sólo lo tenemos una vez y cada vez que lo necesitamos hacemos una llamada a la función, que ahorra muchos más bytes que tener los pasos repetidos.

- Otro problema que ya venía de proyectos anteriores ha sido en el apartado de componer la música. Arkos Tracker 1 es la herramienta que usa cpctelera y está perfectamente integrada en la misma, pero la utilidad que se usa para componer está compilada hace ya bastante años y da problemas en ordenadores actuales, además cerrándose de manera inesperada perdiendo el progreso. Para este proyecto nos decidimos por usar Arkos Tracker 2, que es estable y funciona perfectamente en máquinas actuales. Sin embargo, las canciones compuestas con esta utilidad no son compatibles con cpctelera directamente, para ello hay que hacer uso de un reproductor nuevo. Para este problema contamos con la ayuda de Tonet y Caye de BiteStudios, pues ya habían realizado una serie de mejoras a cpctelera para que se pudiese incluir canciones de Arkos2 de una manera similar a la que hay para Arkos 1. Nuevamente, nuestro más sincero agradecimiento a ellos.

  Además, con Arkos 2 tenemos la posibilidad de crear varias melodías dentro de un mismo fichero. Así conseguimos ahorrar también algo de memoria pues las distintas melodías reutilizan los mismos instrumentos y no tiene que duplicarse esa información en el código resultante.



- Para el desarrollo usamos la misma estrategia que en años anteriores, dada la facilidad que ofrece cpctelera. El proyecto puede iniciarse en 3 estados a la vez (código, gráficos y música) sin que se obtengan bloqueos entre ellos. A la programación se le añaden gráficos y música de prueba para que luego sean sustituidos en el proceso final por los definitivos. Así se puede ir probando cómo va funcionando la respuesta de la CPU y teclado, así como reservar el espacio necesario para la versión final.



MEMORIA

Inicialmente se cuenta con 32KB para todo el contenido del juego: gráficos, música y código, pues usará doble buffer

0x0000	RESERVADO (Vector de interrupciones)
0x0040	Tileset (Los tiles que se usarán)
0x0640	Imagen del Menú (comprimida)
0x093D	Escenarios (comprimidos)
0x1810	FXs y Melodías
0x21F2	Arkos Player
0x2916	Procesos de pintado y borrado de entidades
0x322E	Gráficos
0x4B44	Datos (textos, variables, …)
0x5312	Código
0x7277	Motor para escribir texto
0x7674	Funciones de cpctelera
0x7BE5	Variables globales
0x7F13	Fin
0x8000	2ª Memoria de búfer (y datos + código para introducción)
0xC000	1ª Memoria de búfer

GUIÑO a LARCENA'S LEGACY

Las monedas que aparecen en el juego (en especial la dorada) están basadas en la que aparece en este juego.