OPERATION ALEXANDRA

A 64Kb video game for Amstrad CPC developed by

Code & Engine	JAVIER GARCÍA NAVARRO
Gfx & Game Design	RAFA CASTILLO
Music & Fx	JOHN MCKLAIN
Testing	Blackmores
	Metr81
	JGonza
	Pomez666

- MAKING OF -

v1.0 (24/10/2018)

1. EL ORIGEN

Todo comenzó poco después de terminar la CPCRetroDev de 2017...

JGNavarro estaba “*on fire*” con GameMaker tras terminar la conversión de Profanation 2 de Amstrad CPC a Windows y Android, y pensó que sería buena idea desarrollar una nueva conversión de un videojuego donde todo el material gráfico y sonoro ya estuviera terminado. A través de McKlain, lanzó a Azicuetano la irresistible propuesta de realizar una conversión similar con Baba’s Palace, y en menos de 2 semanas ya tenía lista y publicada la conversión para Windows y Android. Un trabajo intenso pero muy gratificante para todos.

A partir de esta colaboración, los contactos entre los todos fueron cada vez más frecuentes, culminando con la incorporación de Azicuetano al equipo de 4MHz, el grupo donde JGNavarro y McKlain junto a Sad y LordFred desarrollan desde hace tiempo proyectos homebrew para Amstrad CPC.

2. LA IDEA

Ya por noviembre de 2017, Azicuetano llevaba varias semanas detrás de un buen argumento para un videojuego. La idea de base era lo único que tenía suficientemente claro: un videojuego de tiros. También había desarrollado algunos conceptos de personajes realizando algunas acciones, como caminar y disparar... pero era consciente que sin una buena historia no se podía iniciar un desarrollo en condiciones.

En vital contar con una buena historia como punto de partida, porque es precisamente la historia la que permite valorar y contrastar la coherencia de todas las ideas y creatividades que vengan después.

Durante la búsqueda de esa historia que como se llegó a una curiosa noticia de 2016. Una noticia impactante que dio pie a lo que sería la historia sobre la que se desarrollaría el videojuego Operation Alexandra:

"Redescubierta una base militar secreta nazi en el Ártico"

Esta noticia era el punto de partida perfecto para desarrollar una gran historia de ficción en la que se basaría nuestro videojuego:

> *Durante la II Guerra Mundial, los Nazis construyeron esa base en la isla Tierra de Alexandra, dentro del círculo polar ártico, con el objetivo de investigar un OVNI accidentado y conseguir así una tecnología que les otorgara una gran ventaja en la guerra. Levantaron la base sobre él y organizaron una compleja excavación para conseguir acceder al interior del misterioso artefacto. Pero la vía que abrieron para entrar en su interior permitió la liberación de unos temibles seres que en poco tiempo acabaron con todos los habitantes de la base.*
>
> *La base permaneció olvidada durante 30 años hasta que a mediados de los años 70, el protagonista del videojuego - un oficial del ejército soviético - la descubre por casualidad y decide terminar con todos los horrores que campan por la antigua base antes de que algún día lleguen al exterior.*

Azicuetano inició el documento de diseño del videojuego, y allí detalló las múltiples referencias a la noticia en diversos medios, junto a una breve historia de ficción, para presentar el proyecto al resto de compañeros en 4MHz. Finalmente, la idea consiguió ilusionar al resto del grupo, de manera que se organizó un equipo de trabajo compuesto por JGNavarro, Azicuetano y McKlain para abordar el proyecto.

3. EL PROTOTIPO

La tecnología base de este desarrollo sería La Hormigonera, esto es, el *engine* en ASM que JGNavarro tiene en continua evolución para para desarrollar los videojuegos de 4MHz.

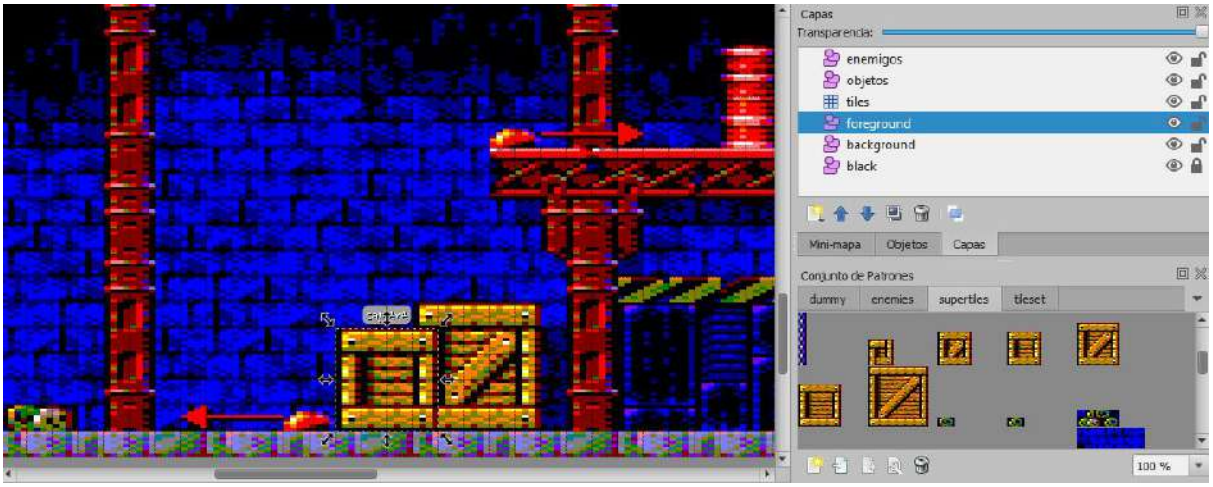
En Profanation 2, el anterior proyecto de 4MHz, JGNavarro y LordFred trabajaron con sistema basado en tiles y supertiles (un grupo de tiles) para crear la definición de los escenarios. Es un sistema complejo, pero que presenta 2 ventajas fundamentales:

1. Ahorrar bytes en la definición de cada pantalla.
2. Una gran riqueza gráfica, ya que permite trabajar con tiles de 4x8.

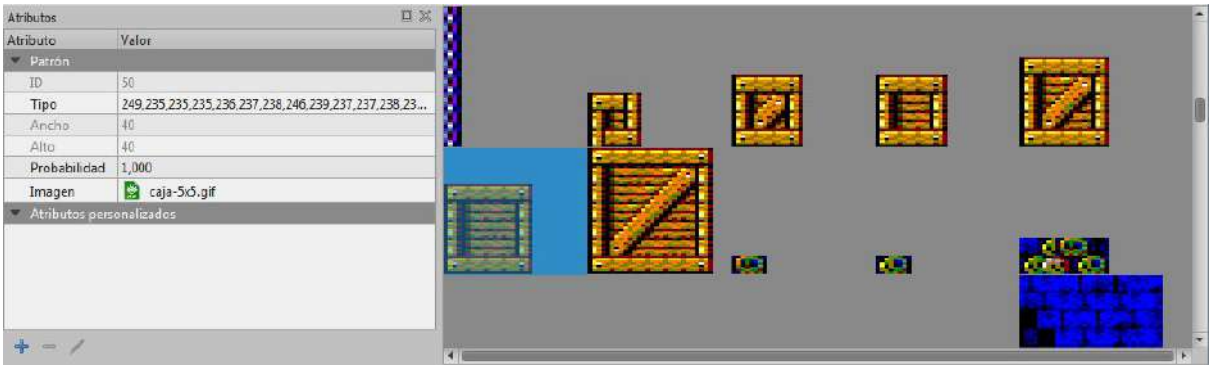
En Profanation 2 este sistema funcionó de manera magistral, y en parte, gracias a él, consiguieron adaptar a 64kb. el desarrollo inicial destinado a 128kb. En ese momento, el trabajo contra-reloj impidió organizar un buen sistema de trabajo, y toda la tarea de codificación de pantallas acabó siendo un trabajo manual realmente duro.

Ahora teníamos la oportunidad de automatizar esta tarea para conseguir reducir errores, ahorrar tiempo y poder destinar esfuerzo a otras partes importantes del desarrollo. También se aprovechó el momento para introducir mejoras que permitieran ahorrar más memoria en las definiciones de pantallas. Así pues, el inicio del proyecto consistió en desarrollar un prototipo con un sistema automatizado para la creación de pantallas basadas en tiles y supertiles.

Como herramienta básica para la creación de pantallas se utilizó Tiled, pero una forma poco convencional. En lugar de crear las pantallas a base de tiles, éstas fueron creadas a base de supertiles. Estos supertiles consisten en “patrones“ tiled. Para crear estos patrones se han usado gráficos *dummy* que recrean todos los supertiles que usamos en el juego.



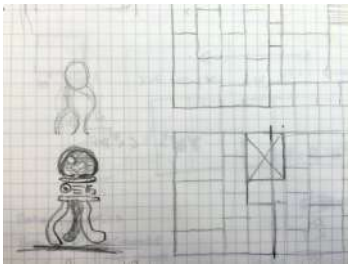
Dentro de cada patrón tiled hemos incluido la definición de cada supertile. Gracias a eso, y a la meta-información que genera Tiled, tenemos en cada archivo .TMX toda la información que necesitamos para componer nuestras pantallas.

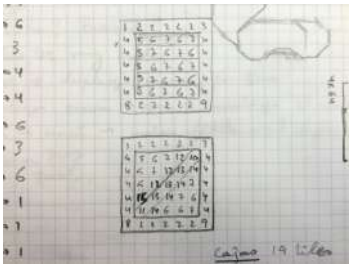


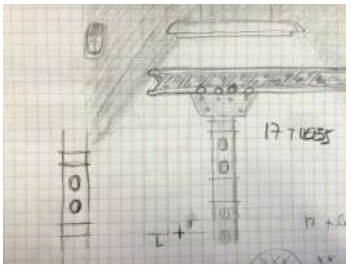
JGNavarro desarrolló una herramienta en Python para parsear los archivos .TMX y realizar una traducción automática de cada pantalla a código ensamblador para Amstrad CPC.

Para diseñar los tiles y supertiles se usó Pro-Motion(Cosmigo). Esta herramienta presenta grandes ventajas, como la posibilidad de trabajar con pixel anamórfico 2:1 (el pixel ancho propio del mode 0 de Amstrad), gestión específica de tiles y posibilidad de exportar estructuras de tiles para generar fácilmente las definiciones de supertiles.

Tras varias semanas de trabajo y pruebas conseguimos dejar establecido y bien ajustado el método de trabajo, y desarrollamos nuestro primer prototipo: 3 pantallas enlazadas entre sí, con el protagonista y un par de enemigos:







4. EL DESARROLLO

Con el prototipo funcionando pudimos realizar las primeras estimaciones. Se creó una hoja de cálculo para realizar todas las estimaciones de memoria, y ahí se estableció cuanta memoria sería asignada a tiles, a la definición de supertiles, a la definición de las pantallas, sprite protagonista, enemigos, juego de caracteres, inteligencia artificial, lógica, etc…

Una vez establecida la previsión de memoria, comenzamos a desarrollar el bestiario: los enemigos y su comportamiento. Este punto era fundamental, ya que gracias a tener bien definido el conjunto de enemigos, con su características, comportamiento, IA, etc. se puede abordar un diseño de niveles inteligente.

En diversos casos se desarrollaron prototipos a modo de animaciones, para estudiar si el comportamiento ideado encajaba con lo que buscábamos y, de paso, servir de referencia a la programación.

Una vez cerrado el bestiario, se inició el diseño de niveles. En este momento, la creación de las pantallas consistía en un procedimiento perfectamente establecido, por lo que la dificultad residía fundamentalmente en la creatividad y en la técnica empleada para conseguir ahorro en bytes y velocidad en el pintado.

El diseño de niveles requería un tiempo considerable, de manera que esta actividad se realizó en paralelo a la implementación funcionalidades requeridas por el juego: efectos especiales, triggers e Interacciones con objetos, sistema de diálogos, apertura de puertas, power-ups, menú principal, menú contextual durante el juego, etc.

Para cuando estuvieron diseñadas la mitad de las pantallas que componen el juego, casi todas las funcionalidades extra del juego ya estaban implementadas, de manera que consideramos era el momento ideal para iniciar la fase de beta-testing.

En la selección de beta-testers se buscó diferentes perfiles con habilidades complementarias. En este caso optamos por un perfil muy técnico, capaz de no sólo encontrar bugs, sino probar y probar hasta conseguir averiguar el modus operandi para reproducirlo. Otro perfil de jugador avanzado que pudiera garantizar que el juego puede completarse de principio a fin. Y por último, un perfil de jugador “habitual”, que pudiera darnos feedback sobre la dificultad que podrá experimentar la mayoría de jugadores.

Se realizaron gran cantidad de ajustes tras la valoración de los informes del equipo de beta-testers, categorizados en ajustes de dificultad y corrección de bugs. Mientras tanto, la tarea de diseño de niveles conseguía llegar a su fin.

Se implementó la lucha contra el enemigo final y el ending, y se suministró una nueva beta al equipo de testing. Los nuevos informes proporcionados, tras las pruebas con esta segunda beta, indicaban que las mejoras en dificultad implementadas habían funcionado

bien, y que la valoración general sobre el juego era muy buena. Así que consideramos cerrado todo lo relacionado con el desarrollo del juego.

Llegados a este punto, apenas contábamos con unos 150 bytes, necesarios para la pila, por lo que no había más memoria para implementar nada nada más. Estábamos muy satisfechos por haber conseguido cumplir la previsión inicial, incluyendo todo lo que teníamos planeado: pantallas, enemigos, melodías, etc.

...aunque echabamos de menos una pequeña intro. Y aquí fue durante un improvisado “brainstorming” cuando tuvimos una idea espectacular para conseguir incluir la intro: compactar TODO el juego y conseguir espacio suficiente para una intro que se vería al cargar el juego, y que tras su visionado sería sobre-escrita al descompactar el juego sobre toda la memoria disponible.

Tras varias pruebas JGNavarro obró el milagro: 17kb a nuestra disposición crear una intro que permitiera meter al jugador en situación. ¡Una verdadera pasada!

Así que nos pusimos manos a la obra con la intro, aunque sin mucho tiempo por delante, ya que estas tareas no las teníamos contempladas.

Tras incorporar la intro, iteramos en un proceso de creación de Release Candidate y testeo, y que llegó a su fin con la Release Candidate 5. ¡Ya teníamos el juego listo para entregar!

5. CREATIVIDAD, DISEÑO, GRÁFICOS Y MÚSICA.

Operation Alexandra cuenta con multitud de obras como fuente de inspiración. Desde películas como “The Thing from Another World” (1951, Dir. Christian Nyby y Howard Hawks) a videojuegos como The Sacred Armour of Antiriad (1986, Palace Software), pasando por melodías como los famosos coros del ejercito ruso.

5.1 Gráficos

La creación de pantallas a base de supertiles implica un trabajo meticuloso en diseño, pero la calidad del resultado, en comparación con la reducida información que se necesita por cada pantalla, hace que bien merezca la pena.

Tomar malas decisiones a la hora de componer una pantalla puede hacer que ocupe el doble o el triple. Hay que ser muy concienzudo en la creación de supertiles, ya que se necesita un repertorio variado para luego poder tener flexibilidad a la hora de diseñar una pantalla.

Esa flexibilidad es fundamental, porque lo más importante del diseño no es la estética, sino la funcionalidad ¡Que sea divertido, vamos! El level design no se puede ver lastrado por no contar con "la pieza" que se necesita…

El sistema desarrollado de supertiles permite solapamiento, lo cual es sensacional para crear pantallas muy atractivas, ya que no es necesario que las distintas estructuras creadas a base de supertiles encajen perfectamente unas con otras, consiguiendo profundidad y rompiendo la sensación de repetición.

Para el desarrollo gráfico se ha contado con un tileset principal compuesto por 256 tiles de 4x8, y un tileset secundario compuesto por 192 tiles de 4x8. El tileset principal es usado para crear todos los escenarios del juego, y el secundario para el resto de gráficos que se necesitan, como la fuente tipográfica, el HUD, menú principal, objetos, el enemigo final,...

Se han desarrollado un total de 230 stiles de muy diversos tamaños, y cuya definición ocupa unos 4kb.

Aseprite ha sido la herramienta utilizada para el desarrollo de todas las animaciones: protagonista, enemigos y explosiones...

5.2 Bestiario

El desarrollo de los enemigos ha sido una de las tareas más divertidas. Idear su comportamiento, ataque, forma de comportarse, etc. condicionaría más tarde el diseño de pantallas.

A grandes rasgos, los detalles más relevantes en el desarrollo de los enemigos han sido:

1. La habilidad de disparar.
2. Los enemigos que disparan deben ser predecibles.
3. Cuando un enemigo es vulnerable o no.

El disparo de los enemigos es dirigido hacia la posición exacta del jugador. Sin duda, un rasgo que denota de inteligencia.

Hemos optado por un sistema en el que los enemigos enemigos no disparen de forma fortuita, de manera que los enemigos que disparan presentan 2 comportamientos que permiten pronosticar su disparo:

1. Gesto característico, lo cual permite al jugador inferir que el enemigo se dispone a disparar.
2. Disparo con una determinada cadencia, cual permite al jugador pronosticar cuando efectuará su próximo disparo.

Por otro lado, ciertos enemigos pueden cambiar su comportamiento al recibir un impacto, por lo que cuando disparas a un enemigo, cuando te metes con él, estos reaccionan y pueden efectuar variaciones en su desplazamiento que pueden coger al jugador desprevenido.

Hay enemigos inmortales, otros que se cierran y se vuelven invulnerables, otros capaces de saltar entre 3 y 11 veces su altura, otro pueden llegar a disparar 2 balas con tiro parabólico, otros al morir dejan en pantalla parte de su mecanismo de desplazamiento y el jugador puede interactuar con él y subirse encima, otros se ponen a girar para anunciar su inminente disparo…

5.3 Jugabilidad y level design

El level design ha constituido un gran reto por muchas razones. Y es que junto al control del protagonista y el comportamiento de los enemigos es uno de los mayores responsables de la jugabilidad del videojuego.

Con Operation Alexandra no buscábamos un juego de tiros donde el protagonista avanza arrasando con todo lo que se encuentra, sino más bien un juego donde el jugador tuviera que poner en práctica cierta estrategia a la hora de abordar cada pantalla. Hay que buscar la forma de disparar a los enemigos, pero también huir en el momento apropiado. Hay zonas desde donde se puede abordar el ataque a los enemigos con mejores garantías, y que que saber encontrarlas. En determinados momentos, el jugador puede escoger entre diferentes caminos y no todos son igual de idóneos dependiendo del punto de desarrollo de la partida.

El juego consta de una serie de efectos visuales para transmitir bien al jugador que está ocurriendo en cada momento. Por ejemplo, el protagonista da un respingo, se tinta de blanco y obtiene unos instantes de inmunidad al recibir un daño. Cuando los enemigos

reciben daño también se tintan de blanco y suena un fx específico. Este detalle es importante ya que no siempre que un enemigo recibe un impacto de bala llega a sufrir daño. Por supuesto, los saltos son un elemento importante, y para ello era fundamental que el control del personaje fuera totalmente preciso, como así se ha conseguido.

Por lo general, en el punto de acceso a una pantalla el jugador suele estar protegido, o protegido durante un tiempo razonable. De esta manera el jugador siempre dispone de un tiempo para pensar y trazar su estrategia.

Este videojuego tiene un pequeño componente de aventura. Para poder avanzar hay que realizar ciertas acciones como coger y usar objetos en ciertos lugares, o eliminar ciertos obstáculos. Para ayudar en estas acciones el jugador cuenta con el monólogo interior del protagonista, que ofrece pistas sobre las acciones a realizar en ciertos puntos del juego.

El mapeado completo está dispuesto de manera que, desde cualquier pantalla el jugador pueda ir a cualquier otra pantalla (una vez el jugador haya liberado los accesos,claro). Es decir, no hay puntos a partir de los cuales no se pueda retroceder. Además, en determinadas ocasiones hay varios caminos para partida tras partida, el jugador aprenderá cual es el mejor recorrido dependiendo del momento, optimizando así su tiempo y disminuyendo el riesgo de ser herido.

Se ha trabajado mucho para ajustar la dificultad. En este punto, el trabajo de los testers es fundamental, ya que los desarrolladores pierden la perspectiva con facilidad debido a la gran cantidad de pruebas que deben realizar. El objetivo desde el primer momento es conseguir un juego donde el jugador sea capaz de mejorar partida tras partida, y que pueda lograr llegar al final con la dedicación oportuna.

Todo los detalles relacionados con la creatividad y conceptualización de Operation Alexandra han sido recogidos en un documento de diseño. Este documento de diseño ha sido la referencia más importante para definir lo que es el juego en sí. En este documento se ha detallado absolutamente todo lo relacionado con el videojuego: historia, textos que componen la intro, valores de paletas gráficas, definición y comportamiento de enemigos, diálogos, efectos especiales, lucha final, ending. Aquí está reflejado desde dónde encontrar un determinado sprite, hasta la sucesión de offsets en Y para componer el salto del protagonista. Está reflejado todo, absolutamente todo. No en vano, el documento de diseño de Operation Alexandra está compuesto por 66 páginas.

5.4 Efectos visuales

Durante el juego, los jugadores podrán experimentar ciertos efectos visuales que no son muy habituales en desarrollos para CPC:

5.4.1 Efecto penumbra

Al inicio de la partida, la base está con la energía de emergencia y el grupo electrógeno se está quedando sin combustible. El jugador experimentará ciertos parpadeos en la luz, fruto de esa falta de energía. Si desciende al sótano, estas pantallas estarán en penumbra.

Este efecto se ha conseguido mediante un cambio de tintas justo tras pintar el HUD. Se ha desarrollado una versión de la paleta oscureciendo un tono todos los colores para dar esta sensación de penumbra.

5.4.2 Efecto nevada

Hay pantallas en la parte superior de la base donde se ve el exterior, y donde se puede apreciar como en el exterior hay una gran tormenta de nieve.

Se han generado unos tiles con unos colores muy particulares para conseguir que, mediante sucesivos cambios de paleta, dar la sensación de ventisca. Además, los cambios de color suceden de tal manera que se consigue transmitir una sensación paralax al trabajar con copos que se desplazan a tres velocidades diferentes.

Al tratarse de un efecto logrado mediante cambio de tintas, esta animación puede cubrir una gran parte de la pantalla sin que por ello se resienta el rendimiento.

5.4.3 Efecto beam

En los pantallas más avanzadas, en la zona del OVNI,el jugador está en contacto con tecnología alienígena. Hay unos rayos capaces de dañar al protagonista si entra en contacto con ellos. Estos rayos emiten un resplandor a modo de vibración.

Este resplandor se consigue sacrificando una tinta de la paleta. Esta tinta en los frames pares es naranja y en los impares amarilla, consiguiendo así el efecto de vibración en el rayo.

5.4.1 Efecto flash

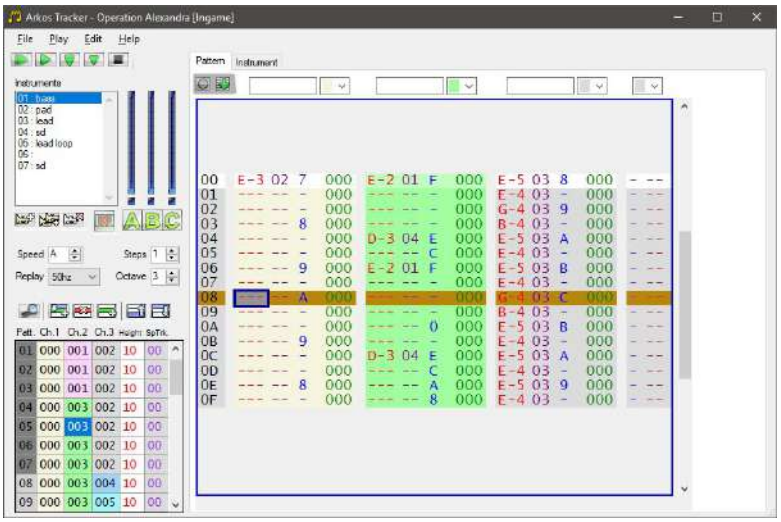
Durante la lucha final, al recibir daño el boss, un resplandor inunda la pantalla.

Del mismo modo que el efecto penumbra, este efecto flash consiste en un cambio a una versión de la paleta más clara durante 2 frames.

5.5 Música y efectos sonoros

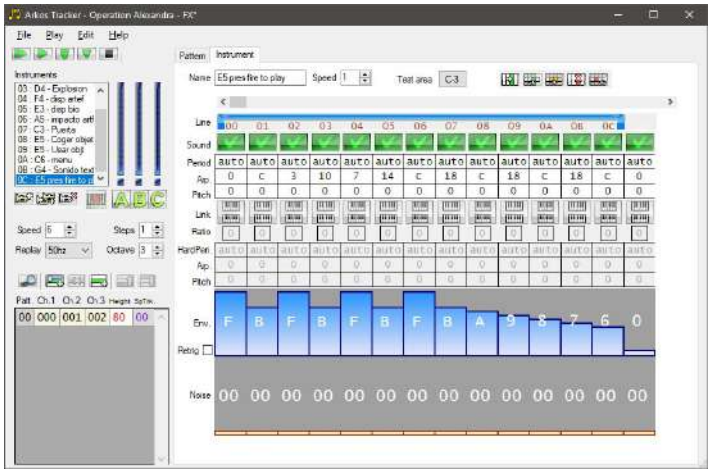
La inspiración para la música ingame viene de las bandas sonoras hechas con sintetizador en los años ochenta en películas de terror y misterio, principalmente pensando en la música de John Carpenter. Se trataba de acompañar al jugador durante la aventura y sumergirlo en

la atmósfera de misterio y peligro que rodea al juego, pero sin distraerlo de la acción principal. Esta melodía dura 1m 55s y ocupa 699 bytes. La música del menú va por otros derroteros y está inspirada por las clásicas melodías interpretadas por los coros del ejército ruso, en un intento de evocar los “días gloriosos" de la URSS durante los que transcurre el juego. Esta música dura 23s y ocupa 717 bytes.



La melodía de la intro va en la línea de la música ingame, un tema sintético para ambientar el diálogo entre el protagonista y sus compañeros de la base. Esta melodía solo usa dos canales para dejar un canal libre para el FX de los diálogos, que suena casi de continuo. La idea del FX es simular una conversación entre los personajes variando aleatoriamente el tono del FX al imprimir cada letra del diálogo y dejando una pausa entre palabras. La melodía dura 38s y ocupa 425 bytes y el FX de los diálogos ocupa 62 bytes.

Entrando de lleno en el apartado de los efectos de sonido, partiendo de un juego de 12 efectos de sonido únicos logramos tener 19 efectos diferentes reutilizando algunos efectos de sonido base que se lanzan con diferentes parámetros. Por ejemplo, los efectos de salto, aterrizaje e impacto de un enemigo con nuestro protagonista comparten el mismo efecto base, pero se lanzan con diferentes valores de nota, con el consiguiente ahorro de memoria.



En la lista de efectos de sonido tenemos efectos de salto y caída, disparos, impactos, explosiones, melodías cortas para indicar la recogida y uso de objetos y un fx para los diálogos del protagonista.

Orden	FX Nº	Nombre	Valor	Canal	Nota
00	**1**	disparo prota	**36**	1	C3
01	**2**	salto	**35**	0	B2
02	**2**	aterrizaje	**26**	0	D2
03	**3**	exp. pequeña	**33**	1	A2
04	**3**	exp. grande	**50**	1	D4
05	**4**	disp. artefacto	**53**	1	F4
06	**5**	disp. biológico	**41**	1	F3
07	**6**	imp. disp. con. artef. (cerrado)	**69**	1	A5
08	**6**	imp. disp. con. artef. (abierto)	**64**	1	E5
09	**5**	imp. disp. con. bio.	**29**	1	F2
10	**4**	imp. disp. ene. prota	**33**	1	A2
11	**2**	imp. ene. con prota	**45**	0	A3
12	**7**	abre puerta	**36**	1	C3
13	**8**	coger objeto	**64**	2	E5
14	**9**	usar objeto	**64**	2	E5
15	**10**	menu - esc	**72**	0	C6
16	**10**	menú - opción	**65**	0	F5
17	**11**	sonido textos	**55 (55-58)**	0	G4
18	**12**	press fire to play	**64**	0	E5
19	**1**	disparo prota largo	**41**	1	F3

Los efectos que usan el canal de ruido se envían al canal central para evitar conflictos en el uso del generador de ruido (la música ingame usa ruido para los sonidos percusivos del canal central), los fx relacionados con objetos se envían al canal derecho para cortar la melodía principal al sonar y el resto de fx se envían al canal izquierdo. En total este juego de FX ocupa 329 bytes en memoria.

La música y efectos se han hecho con Arkos Tracker 1.0, ya todo un estándar en las nuevas producciones para Amstrad CPC.

6. TECNOLOGÍA

La programación se ha desarrollado al completo en lenguaje ensamblador. Si bien al inicio del proyecto muchas rutinas parten de la base de La Hormigonera, durante el desarrollo muchas de esas rutinas son mejoradas y optimizadas para resolver de manera más eficiente los casos particulares que se dan en este videojuego.

En particular, hay varios aspectos que merecen una explicación un poco más detallada:

6.1 Inteligencia Artificial

Durante la fase de diseño se establecieron comportamientos muy particulares y específicos para cada enemigo.

6.1.1 Disparos a una coordenada específica

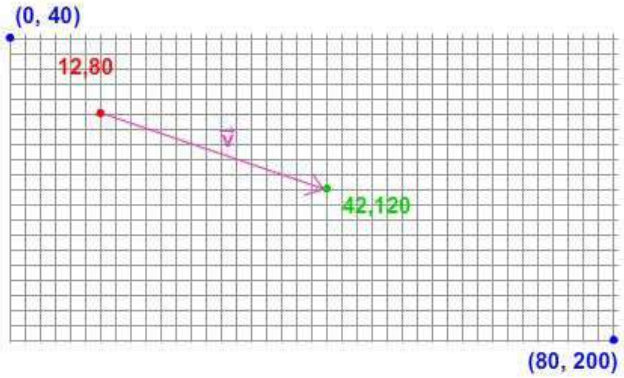
De todos los comportamientos de los enemigos, el más relevante es la habilidad para disparar un proyectil a un punto específico de la pantalla: donde se encuentra el protagonista. Una tarea trivial en la actualidad, pero ciertamente compleja en 8bits.

Para implementar esta acción desarrollamos una versión del algoritmo Bresenham, y realmente funcionar funcionaba, pero no estábamos contentos con el resultado. La razón de era que este algoritmo está pensado para pintar líneas entre dos coordenadas por lo que obligaba a pintar la bala siempre en píxeles contiguos. Esto provocaba que el proyectil se desplazaba 4 veces más rápido en el eje X que en el eje Y. No estaba mal, pero queríamos hacerlo mejor.

De manera que optamos por una variación de uno de los algoritmos tradicionales para desplazar objetos entre dos coordenadas. Realizamos una mejora del algoritmo base bautizada como “Evaristo” para corregir el pixel anamórfico del amstrad 2:1, y el hecho de que de forma natural se pinta en bytes exactos (de 2 en 2 pixeles en horizontal).

Es decir, debíamos realizar una compensación de 4:1 en horizontal:

PIXEL CUADRADO 1:1



$\vec{v}$=(dX, dY)
dX = x2-x1 = 42-12 = 30;
dY = y2-y1 = 120-80 = 40;

$\vec{v}$=(30, 40)

$|\vec{v}| = \sqrt{30^2 + 40^2} = 50$

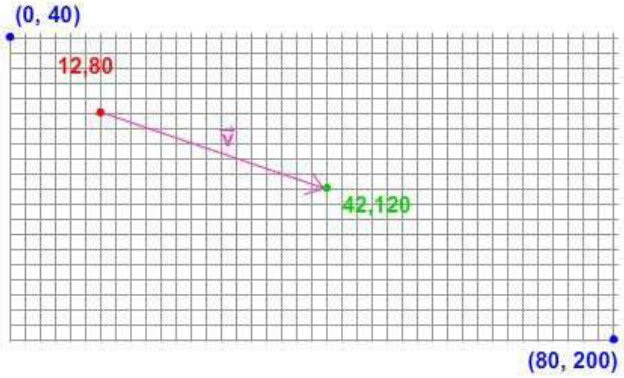
$\vec{u} = \frac{1}{50} \cdot (30, 40)$ --> $\vec{u} = (\frac{30}{50}, \frac{40}{50})$

$\vec{u}$ = (0.6 , 0.8)

IncX = velocidad * 0.6
IncY = velocidad * 0.8

IncX = 0.6
IncY = 0.8

CON COMPESACIÓN ANAMÓRFICA 4:1



$\vec{v}$=(dX, dY)
dX = x2-x1 = 42-12 = 30 * 4 = 120;
dY = y2-y1 = 120-80 = 40;

$\vec{v}$=(120, 40)

$|\vec{v}| = \sqrt{120^2 + 40^2} = 126.49$

$\vec{u} = \frac{1}{126.49} \cdot (120, 40)$ --> $\vec{u} = (\frac{120}{126.49}, \frac{40}{126.49})$

$\vec{u}$ = (0.9487 , 0.3162)

IncX = velocidadX * 0.9487
IncY = velocidadY * 0.3162

IncX = 0.9487
IncY = 1.2648

COMPENSACIÓN ANAMÓRFICA
velocidadX = 1
velocidadY = 4

La implementación de este algoritmo tampoco es trivial al tener involucradas raíces cuadradas y operaciones en coma flotante. Pero tras un día de ensayo y error conseguimos resolver este problema. ¡Fue muy gratificante poder ver por pantalla los proyectiles desplazándose a velocidad constante sin importar su dirección!

6.1.2 Reacciones ante el ataque.

Se ha implementado una especie de reacción de los enemigos ante el ataque.

En algunos casos, no siempre, un enemigo reacciona ante un impacto de bala cambiando su trayectoria de desplazamiento. Esto no ocurre siempre, de hecho ocurre pocas veces, pero con eso conseguimos nuestro objetivo, que es que no te puedas fiar de la trayectoria que describe un enemigo, ya que ante un impacto podría variar y cazar desprevenido al jugador.

6.1.3 Elección del momento del disparo

Hay un enemigo que dispara y otro que salta con una cadencia fija y que, como se ha indicado anteriormente, es para que su ataque pueda ser predecible. Pero hay otros enemigos que toman su propia decisión sobre cuándo disparar, como son la *Almeja*, la *Torreta* y la *Planta*.

Estos enemigos, avisan que van a disparar con un gesto característico, pero la elección sobre cuándo disparar la toma el enemigo, y no están sujetos a una cadencia prefijada.

6.1.4 Cambios de dirección

Contamos con un enemigo al que hemos denominado *Rana*, que tiene recorridos prefijados. Este enemigo puede decidir sobre el momento donde inicia un cambio de dirección en función de las acciones que realice el jugador y de cierto componente de azar. La forma más habitual de evitar a este enemigo es saltandolo, por lo que un cambio de dirección en este momento puede arruinar las intenciones del jugador.

6.2 Supertiles

Como ya hemos comentado, las pantallas están generadas a base de supertiles con el objetivo primordial de ahorrar en las definiciones de las pantallas y conseguir riqueza gráfica.

Se han desarrollado diversas optimizaciones que permiten mayor compresión en casos particulares gracias a la repetición de supertiles en horizontal, vertical y a modo de trama. Este sistema hace que la definición de cada pantallas sea variable. Depende de los tiles y supertiles usados, de en qué capa se encuentran, de cuantas veces se repiten en esa pantalla, de cómo se definen repeticiones tipo trama… Para aprovechar bien las optimizaciones es importante tener buen ojo en el level design.

La media de lo que ocupan las pantallas en Operation Alexandra son 126 bytes. Importante destacar que una pantalla está compuesta por 40 tiles de ancho por 20 de alto, de manera que se consigue codificar en 126 bytes una pantalla formada por 800 tiles independientes.

El solapamiento de tiles/supertiles es posible pero peligroso, ya que aumenta el tiempo de generación de la pantalla. Podría definirse una pantalla con tan solo 32 bytes, usando 20 supertiles de 8x10 tiles, lo que implicaría dibujar 1600 tiles para componerla. La definición de la pantalla sería pequeña, tan sólo 32 bytes, pero el tiempo de pintado sería inasumible.

Se desarrolló una herramienta específica en Python para traducir las pantallas creadas desde Tiled a código ASM, con el consiguiente ahorro en tiempos, reducción de errores y flexibilidad. Gracias a esta herramienta, incorporar al juego una modificación en una pantalla consistía en una tarea de segundos.

6.3 Gestión de sprites por interrupciones y buffers dinámicos.

La complejidad mayor de un juego desarrollado en 64Kb. es la falta de espacio y el gran sacrificio de memoria que supone de usar doble buffer para realizar los pintados de sprites sin parpadeos incómodos. Para ello se ha aprovechado para realizar cada cosa en su momento, de manera que esté todo controlado. Pero cómo saber qué momento es el actual si el CRTC de Amstrad no indica dónde está el haz de electrones en cada momento. Pues fácil, ejecutando los procesos en las interrupciones que, previamente, se han definido para empezar en el mismo momento siempre: cuando el haz de electrones está en la parte superior izquierda de la pantalla (único momento en el que CRTC informa).

De esta manera la primera interrupción se produce arriba a la derecha de la pantalla, y las siguientes cada 10Hz aproximadamente. Esto quiere decir que hay que aprovechar para pintar en pantalla en los momentos en los que sabemos que no está el haz de electrones “encima”.

Además, visto que era innecesario mover los objetos a 50fps si no que con 25fps se conseguía la velocidad idonea, se optó por pintar cada dos refrescos de pantalla los sprites (lo mismo que decir 25fps).

Por ello se han definido las siguientes acciones por cada interrupción:

Interrupción/Zona Pantalla	**Paso 0**	**Paso 1**
0 Zona superior de la pantalla	**SPRITE PROTAGONISTA** Captura Fondo de Pantalla	**ENEMIGO 1** Imprime Fondo Pantalla Mueve sprite Imprime Sprite en Pantalla
1 Marcador	**SPRITE PROTAGONISTA** Imprime Sprite en Pantalla	**DISPAROS** Captura Fondo Pantalla Imprime Sprite Pantalla
2 Zona superior de Juego	Efectos de Tintas Lectura del Teclado Arkos Player	
3 Zona media de Juego	**ENEMIGO 5** Imprime Fondo Pantalla Mueve sprite Imprime Sprite en Pantalla	**ENEMIGO 3** Imprime Fondo Pantalla Mueve sprite Imprime Sprite en Pantalla
4 Zona inferior de Juego	**ENEMIGO 2** Imprime Fondo Pantalla Mueve sprite Imprime Sprite en Pantalla	**ENEMIGO 4** Imprime Fondo Pantalla Mueve sprite Imprime Sprite en Pantalla
5 Zona inferior de la pantalla	**SPRITE PROTAGONISTA** Imprime Fondo Pantalla	**DISPAROS** Imprime Fondo Pantalla

Con esta estrategia da tiempo justo para imprimir cada dos refrescos de pantalla:

- 1 Sprite protagonista de 12px*24px a 50fps
- 5 Sprites enemigos de hasta 16px*16x a 25fps
- 2 disparos amigos de 4px*6px a 50fps
- 3 disparos enemigos de 4px*6px a 50fps

Se puede observar en la tabla anterior que los enemigos no realizan captura del fondo de pantalla, lo cuál es muy extraño. El motivo es que los enemigos tienen un área de movimientos previsibles, por lo que hemos optado por captura el fondo por donde se mueven para ahorrar tiempo. El buffer donde se almacena esta información es de 3Kb. solo, frente a las 13Kb, que serían necesarias si se capturara todo el fondo.

El protagonista y los disparos sí que capturan el fondo por lo mismo:

- porque se necesitarían 13Kb. y no las tenemos.
- porque el movimiento no es previsible y puede ser en cualquier zona de la pantalla.

Para ahorrar más espacio de memoria, los sprites se guardan mirando a la derecha y se rotan al vuelo. Afortunadamente, el diseño de los enemigos por parte de Azicuetano ha sido simétrico en movimientos y no hay que rotarlos, pero el protagonista no podía ser simétrico, por lo que hay que rotarlo. La complejidad no sería mucha si no fuera porque las interrupciones sólo permiten ejecutar unos pocos ciclos de reloj que dan para imprimir un sprite de algo más de 12px*24px como mucho. Si hiciéramos la rotación al vuelo tardaría unas veces mucho y otras menos, por lo que el tiempo no sería previsible y podríamos perder el control y llegar a tener desincronización de las interrupciones y perder el control de manera absoluta. ¿Cómo se resolvió esto? Muy fácil. En una charla con Dr. Fran Gallego sugirió la posibilidad de guardar el protagonista mitad rotado y mitad no. Usando esta técnica conseguimos que siempre se tardara lo mismo en dibujar al protagonista mirando a izquierda o a derecha y, además, que diera tiempo en una interrupción.

6.4 Triggers y acciones

El juego consta de ciertos componentes de aventura:

- Objetos que pueden ser recolectados y pasan al inventario.
- Lugares donde los objetos del inventario pueden ser usados.
- Objetos y posiciones capaces de desencadenar monólogos del protagonista para ofrecer pistas y meter al jugador en la historia.

Toda esta gestión se realiza mediante un tipo de objetos especiales donde la colisión del protagonista con ellos es capaz de desencadenar las acciones comentadas.

Esta acciones obedecen a una lógica muy concreta, y a un trabajo muy específico en programación, ya que no es posible desarrollar un método general para resolver acciones tan especiales sin lastrar el rendimiento.

6.5 Sistema de compresión

Una de las ideas de la que estamos especialmente orgullosos es del sistema empleado para incluir la intro dentro de los 64kb del juego.

Una vez terminado el juego, con apenas 150 bytes libres, ideamos un sistema para comprimir todo el juego con Exomizer y así disponer de espacio para poder desarrollar una introducción la juego. Esta intro sólo será posible visualizarla nada más cargar el juego ya que, una vez presentada la intro, el juego se descomprime sobre toda la memoria, sobreescribiendo el juego comprimido (con el margen de seguridad apropiado) y la intro.

Esta implementación no es tan sencilla como pueda parecer. Ha costado cierto esfuerzo en pruebas y ha obligado a desarrollar un loader específico, pero el beneficio de esta técnica es evidente, y gracias a ella conseguimos nuestra ansiada intro sin cargas extra.

7. HERRAMIENTAS UTILIZADAS

Este proyecto ha sido desarrollado bajo los S.O. Windows y MacOS de manera simultánea. Las principales herramientas utilizadas para el desarrollo del proyecto han sido:

- Propias desarrolladas en Python:
    - bitmap2CPC - Sprites.py: convierte cualquier bitmap en formato raw de CPC (ordenado por scanlines o en 8 tramos)
    - bitmap2CPC - Tiles.py: convierte cualquier bitmap en formato raw de CPC (ordenador por scanlines o desordenado para optimización)
    - pantalla2CPC.py: convierte TMX en formato raw CPC utilizable por el motor de 4MHz.es
    - Stiles2cpc.py: convierte el XML con los STILES en formato ASM
- Generación de discos: ManageDSK, CPCDiskXP y WinAPE 2.0b2
- Generación de CDTs: Herramientas DINAMIC y CSW2CDT
- CPCtelera 1.4 - Prototipado de efectos.
- WinAPE 2.0b2 & RMV2 beta - Emulación de Amstrad.
- Sublime Text - Editor de código.
- Aseprite - Creación de sprites, animaciones y pantalla de carga.
- Pro-Motion - Creación y gestión de tiles y supertiles.
- PaintBrush - Ajustes rápidos en assets
- Arkos tracker - Creación de Música y FX.
- Tilemap - creación de niveles.
- Telegram y teléfono - Comunicación entre el equipo.
- Trello - Organización de tareas.
- Documentación: Manual Zilog y http://clrhome.org/table/
- Google Drive y Dropbox - Almacenamiento de materiales, fuentes y assets.
- Google Docs - Generación de la documentación.

8. CRÉDITOS

Code & Engine	JAVIER GARCÍA NAVARRO
Gfx & Game Design	RAFA CASTILLO
Music & Fx	JOHN MCKLAIN
Testing	Blackmores
	Metr81
	JGonza
	Pomez666