Baba’s Palace

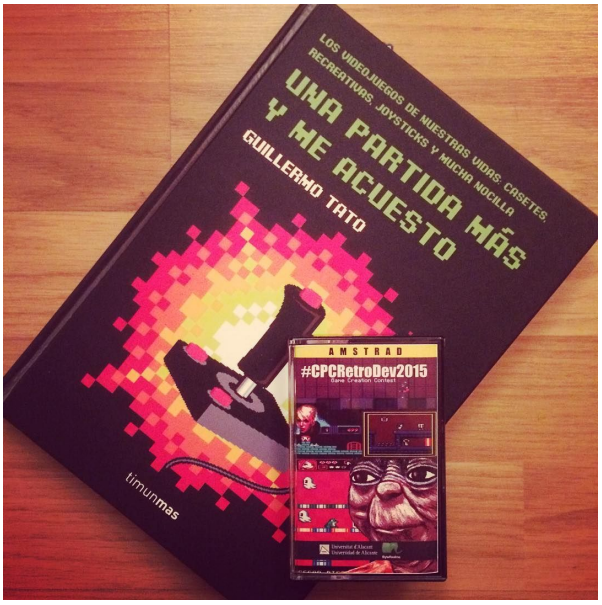
A 64Kb videogame for Amstrad CPC

Code & Gfx by Rafa Castillo · Music & Fx by John McKlain

MAKING OF...

EL ORIGEN

Todo comenzó en octubre de 2016 cuando acudí al evento Retroconsolas Alicante con intención de asistir a la charla de The Oliver Twins. En mi infancia tuve un Amstrad 464 y desde siempre me han cautivado los videojuegos para sistemas de 8 bits. En Retroconsolas me di cuenta que el desarrollo para Amstrad estaba viviendo una segunda juventud, gracias a impresionantes producciones homebrew como Bubble Bobble 4CPC, OutLaws, Magica, Hire Hare, Space Moves, Space Pest Control, Adiós a la casta, Frogalot, Golden Tail, La Guerra de Gamber, Imaginario colectivo, Teodoro no sabe volar, Phantomas Saga Infinity, Hair-Boy, TopTop…



Fue precisamente ese día cuando descubrí el concurso CPCRetroDev, y en ese momento lo vi claro: Le debía un videojuego a mi Amstrad, que tantos buenos ratos me dio cuando era pequeño. ¡Me presentaría a la próxima edición!

Una vez convencido de mi propósito, comencé a indagar sobre cómo desarrollar un videojuego para Amstrad y directamente tropecé con CPCtelera y los vídeos de Fran Gallego. ¡Qué pasada! ¿Se puede pedir más?

Noche tras noche me dediqué a refrescar mis conocimientos en lenguaje C (15 años llevaba sin programar en C) y a estudiar con atención todos los videos sobre programación con CPCtelera publicados por Fran. Con lo que poco a poco aprendía y con ayuda de los ejemplos incluidos en el pack de CPCtelera, en poco tiempo experimenté avances que consiguieron ilusionarme con el proyecto, y en ese momento comencé a darle vueltas al videojuego que me gustaría crear.

LA IDEA

Aunque sí contaba con experiencia previa en desarrollo de videojuegos, tenía muy claro que era un completo novato en el tema de Amstrad, por lo que no quería meterme en un proyecto de complejidad excesiva del que pudiera perder el control. Así que saqué del cajón una idea que me rondaba la cabeza desde hacía muchos años y que encajaba con lo que buscaba.

En 1985, Yutaka Isokawa programó en Hudson's Basic para el Sharp MZ-700 un videojuego al que tituló Pitman. En 1990 se publicó una versión para Game Boy con el mismo nombre (a.k.a. Catrap). Es un juego que me hizo disfrutar muchísimo en mi juventud, con mecánicas que beben de clásicos como Boulder Dash o Sokoban, y desarrollar un juego basado en este concepto me pareció un reto ideal.

Echando la vista atrás, tengo que reconocer que me ha resultado durísimo este desarrollo. El concepto de Isokawa obedece a unas lógicas realmente complejas, y a una serie de acciones que suceden en cascada (como la aplicación de la gravedad en ciertos casos particulares) que me ha exigido un esfuerzo muchísimo más grande que lo que inocentemente suponía en un principio.

EL APRENDIZAJE

No tardé en darme cuenta de la importancia que tiene un simple byte cuando sólo cuentas con 64kb. Al poco tiempo de iniciar el desarrollo tuve que ponerme a optimizar, optimizar y optimizar: Ahorrar en gfx, reprogramar algoritmos, repensar el cómo incluir 100 niveles, buscar y eliminar gasto de memoria innecesario…

Poco a poco fui acostumbrándome a una nueva forma de pensar: cómo conseguir lo máximo con lo mínimo. Cada nueva idea debía estar sometida a cuánta memoria se necesitaría para llevarla a cabo. Las cuentas de la memoria que necesitaría debían estar muy claras: cuanto se lleva la música, cuanto se lleva la definición de los niveles, cuanto se lleva los gráficos, la lógica, las diferentes opciones…

Y aquí llegó lo más sorprendente para mi: ¡era muy divertido! Disfrutaba pensando como arañar unos pocos bytes de aquí y de allá dándole una vuelta a un algoritmo, o planeando cómo ahorrarme un determinado paso, o cómo mejorar un concepto para optimizar en memoria o en rendimiento según interesara. Era como tener entre manos un rompecabezas gigante y planificar concienzudamente cómo solucionarlo.

LA PROPUESTA

Con la primera imagen del juego que hice pública conseguí atraer la atención de McKlain, uno de los mejores músicos de la escena de Amstrad, con una larga lista de videojuegos musicalizados a sus espaldas. ¡Un verdadero fuera de serie! A partir de ese momento lo tuve claro: debía conseguir ilusionar a McKlain para que participara en este desarrollo. Sabía que al ser este mi primer proyecto para Amstrad, la propuesta de colaboración debía ser algo más que palabras, por lo que me propuse conseguir un prototipo con un mínimo de atractivo para poder lanzar a McKlain una propuesta lo suficientemente seria.

En marzo conseguí un nivel de desarrollo que permitía entrever lo que podría llegar a ser el videojuego. Monté un video teaser de presentación del proyecto y en privado lancé la propuesta a McKlain. En cinco minutos tenía su respuesta en forma de emoticono, una mano con el pulgar en alto. ¡Wow!

A finales de Marzo, cuando presenté en sociedad Baba's Palace con el video teaser, pude anunciar que el mismísimo McKlain se encargaría del apartado sonoro. Para mí, poder contar con McKlain, su talento y su experiencia es todo un sueño hecho realidad.

Su colaboración también supuso para mí una gran responsabilidad. Ahora había otra persona implicada en el proyecto y yo no podía fallarle.

EL DESAFÍO

Para mi, el desarrollo de Baba's Palace ha significado un verdadero reto por mi desconocimiento técnico sobre Amstrad y por lo oxidado de mis conocimientos en C. Pero al margen de estas circunstancias, y en lo relativo específicamente a este videojuego, me he encontrado con un gran desafío en los siguientes aspectos:

Gráficos

Opté por un área de juego que ocupara toda la pantalla, lo que en la práctica significa abandonar el uso doble buffer y gestionar todo el dibujado de sprites controlando la ubicación del raster para evitar parpadeos. Esto se torna complejo de gestionar a la hora de animar hasta 24 enemigos en una sóla pantalla sin que se aprecie pérdida de rendimiento.

Niveles

Crear un videojuego con 100 niveles ha supuesto un enorme reto: desde la necesaria optimización de memoria, pasando por el gran esfuerzo de idear tantos niveles, hasta la creación de un editor de niveles que permitiera su testeo en tiempo real. Sin cuidar todos estos factores conseguir 100 niveles para este juego sido prácticamente imposible.

Lógica

Una de mis mayores sorpresas fue la de cómo un juego con un aspecto relativamente sencillo podía entrañar una complejidad lógica tan alta. Los diferentes casos que se pueden dar durante una partida son realmente complejos, sobre todo si tenemos en cuenta sucesos encadenados que ocurren al empujar piedras o cambiar a una posición diferente con objetos apilados sobre el protagonista… Hay un caso donde un sólo movimiento del protagonista implica hasta una secuencia de 5 sucesos, 2 de ellos con caídas de ítems en cascada. ¡Una locura!

Rage against the memory machine

La necesidad de optimizar al máximo hasta el mínimo detalle ha sido la tónica general durante el desarrollo. Me atrevo a decir que por cada hora destinada a programar, he destinado 2 horas a reprogramar algoritmos y funciones, todo para ahorrar el máximo en bytes.

EL VIDEOJUEGO

Hay dos ideas fundamentales donde reside la concepción artística de este videojuego:

1. **BABA YAGA**
   Baba Yaga es el nombre de una famosa bruja del folclore y la mitología eslava. Va montada en un almirez mágico que vuela. Siempre me ha fascinado este personaje…

2. **KUNG FU**
   Mítica serie de mediados de los ‘70 protagonizada por David Carradine que narra las aventuras de un solitario monje shaolín que viajaba a través del lejano oeste usando como únicas armas su destreza en artes marciales y la fuerza interior de su filosofía de vida, el budismo.

Fruto de estos conceptos surge la historia un poco anacrónica que se da en este videojuego, donde Little Indian y Shaolin Kid se enfrentan a los secuaces de Baba Yaga en busca del elixir de la vida que permita resucitar al padre del pequeño indio.

Baba’s Palace es un juego de lógica con estética arcade y cuenta con hasta 100 niveles. El jugador puede manejar a Shaolin Kid y a Little Indian para resolver los puzzles, eliminando a todos los enemigos presentes en cada nivel, mientras descienden a las profundidades del palacio de la bruja Baba Yaga en busca del elixir de la vida.

Experiencia de juego

Para mi siempre ha sido muy importante que el jugador se sienta muy cómodo jugando. Por ello en el menú principal se ofrece la opción de jugar con Joystick, teclado e incluso poder redefinir teclas.

En cuanto a los movimientos del personaje principal, se ha buscado por todos los medios que la acción tenga mucho ritmo. Sería desesperante visualizar mentalmente el recorrido que deseas trazar con el protagonista y que el trayecto sea insufrible por lentitud en las acciones.

Los primeros niveles sirve para introducir al jugador en las mecánicas del juego. A modo de tutorial se van exponiendo las reglas principales que rigen la lógica, de manera que el jugador aprende, casi sin darse cuenta, que las piedras se pueden empujar, que los enemigos se eliminan al tocarlos, que hay muros que se pueden destruir, etc.

La creación de los 100 niveles ha sido bastante compleja, sobre todo porque para idear un nivel que represente un reto intelectual hay que trabajarlo muy bien y pasarle un riguroso control de calidad. Cinco de los niveles finales son adaptaciones de niveles que podemos encontrar en la versión Game Boy. Son niveles míticos y merecen ser conocidos y disfrutados.

Los fallos más habituales en la creación de niveles son:

1. Crear un nivel que es imposible resolver. Para evitar esto es imprescindible probar todos y cada uno de los niveles.
2. Crear un nivel que se puede resolver de una forma más sencilla que la idea bajo la que se desarrolló inicialmente. Para no caer en este error hay que realizar muchas pruebas que aseguren que la única forma de resolver el nivel es siguiendo la estrategia con la que se diseñó inicialmente.

Al superar cada nivel suena un pequeña fanfarria o jingle de éxito, reforzando el mensaje de LEVEL CLEAR y recompensando al jugador por su buen hacer.

Tecnología

Para conseguir el ritmo de juego deseado era muy importante conseguir unas rutinas de pintado de sprites que fueran muy eficientes. Al prescindir del doble buffer hay que gestionar muy bien el pintado de sprites para evitar parpadeos. Aquí he aplicado una técnica que me ha permitido una mejora muy importante del rendimiento.

Frente al clásico *waitVSYNC* (que tampoco te libra de los parpadeos por la zona superior de pantalla) he implementado una función denominada *waitRaster*. Esta función me permite conocer la zona de pantalla por la que va el raster. Si el *raster* no está por la zona de

pintado, pinto el sprite directamente (por estadística debe ocurrir 5 de cada 6 veces). Si detecto colisión, se realiza una breve espera hasta asegurarme que el raster ha pasado por la zona problemática y pinto mi sprite. ¡La mejora en rendimiento frente a un *waitVSYNC* es espectacular!

Otro punto muy delicado es la cantidad de enemigos que pueden estar a la vez en pantalla. El sistema implementado permite hasta 24 enemigos en pantalla, cambiando de estado cada 0.5 segundos (movimiento idle). Nuevamente, gracias al poder pintar sprites sin tener que hacer un *waitVSYNC* esto ha sido posible a la velocidad deseada.

La transición entre pantallas se realiza mediante *fades* a/desde negro. Es un efecto muy rápido, pero le da un toque elegante muy interesante. La paleta pasa por 6 estados diferentes al realizar cada *fade*, y los colores han sido precalculados a mano para conseguir el efecto deseado.

La función para pintar texto en pantalla permite pintar texto en diferentes colores, y maneja mayúsculas, minúsculas y espacio variable entre caracteres como en el caso de las letras i, l…

En la intro se ha usado *Exomizer 2 Z80 decoder* con el objetivo de no sobrepasar un límite en Kb aceptable para una intro (en este caso 8kb).

Uno de los mayores retos técnicos a los que me he enfrentado ha sido el cómo codificar 100 niveles para que sean visualmente atractivos y con el mínimo coste en Kb por nivel. Para conseguir superar este reto he desarrollado un sistema de niveles codificados con un bajo número de elementos para, posteriormente, de manera procedural, crear un nivel final atractivo.

- Cada nivel consta de 11x9 tiles, pero la última fila siempre es un bloque sólido, por lo que tan solo necesito codificar 11x8 tiles.
- Cada nivel tiene hasta 8 tipos diferentes de tiles: bloque sólido, bloque destruible, escaleras, enemigo terrestre, enemigo volador, protagonista, piedra y hueco. De manera que uso 3 bits para codificar cada uno de estos elementos.
  En total, 11x8x3 bits = 264 bits = 33 bytes por nivel → 100 niveles = 3.300 bytes.
- Para cargar un nivel, leo el tipo de elemento que hay en una determinada posición y, en función de los elementos que hay alrededor, selecciono el gráfico que se debo asignar a esa posición.
- En el caso de pintar un tile de fondo, según estemos en una fila par o impar se selecciona un tipo u otro de gráfico, y en función del azar si ese gráfico es en su versión clara u oscura. Todo destinado a romper sensación de trama o repetición.
- Por último se ejecuta una función *Beautify* que mejora el aspecto gráfico del nivel con columnas y rostros de antiguos ídolos.

Al poco tiempo de trabajar en la creación de niveles me di cuenta que no era viable el sistema que estaba siguiendo de crear el nivel con un editor de niveles clásico, compilar el proyecto y probarlo en el emulador para, seguidamente, detectar errores o mejoras y volver

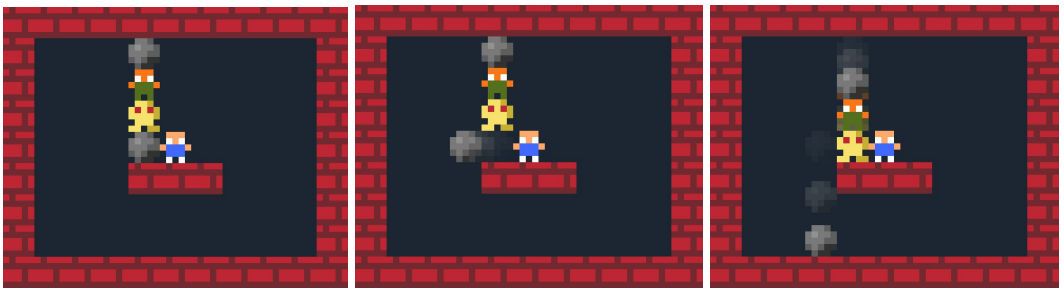
a empezar... Finalmente acabé desarrollando un sistema propio de edición de niveles en PuzzleScript, lo que me permitía dentro del mismo editor de niveles jugar al nivel y detectar problemas y mejoras en tiempo real. Esto ha sido clave para poder acometer la producción de mis tan ansiados 100 niveles.

Lógica e Inteligencia Artificial

Este proyecto no tiene enemigos que presenten un comportamiento inteligente basado en algoritmos propios de inteligencia artificial. En cambio, sí cuenta con una programación lógica muy avanzada para poder resolver la gran cantidad de situaciones que se pueden dar al interactuar con los elementos de un nivel, así como para generar de forma procedural cada uno de los niveles.

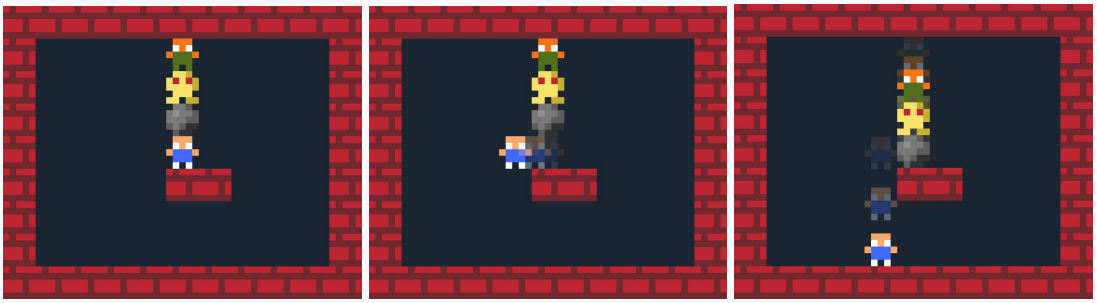
El protagonista puede interactuar con los elementos del nivel:

- **Empujando piedras**
  Al empujar una piedra, ésta pasa a ocupar el hueco adyacente según la dirección en la que sea empujada. Todos los elementos que pueden estar afectados por la gravedad inician su caída en cascada:
  - La misma piedra que acabamos de empujar si en la casilla adyacente no hay suelo.
  - Los elementos que estuvieran encima de la piedra (otras piedras, enemigos terrestres, protagonista compañero).



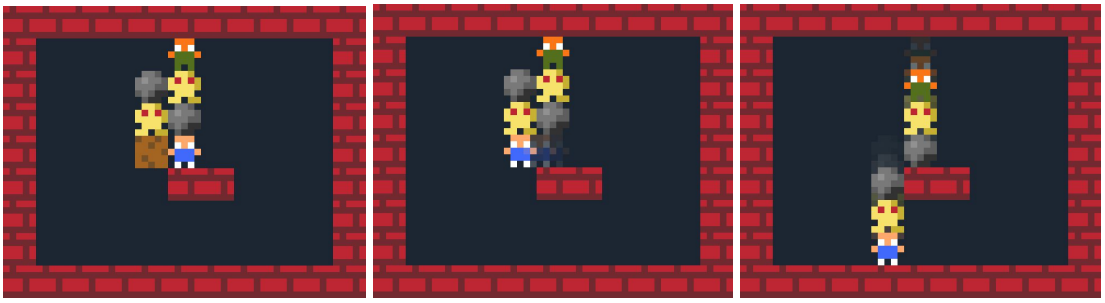
- **Caminando a una posición adyacente o bajando por escaleras**
  - Se puede dar la situación donde el protagonista camina a una casilla adyacente, pero sobre el protagonista pueden existir elementos que se ven afectados por la gravedad.
  - De la misma manera puede ocurrir que el protagonista baje por una escalera teniendo elementos afectados por la gravedad sobre él mismo.

  Al realizar cualquiera de estas acciones, los elementos afectados por la gravedad caerían en cascada a la posición que ocupaba el protagonista.

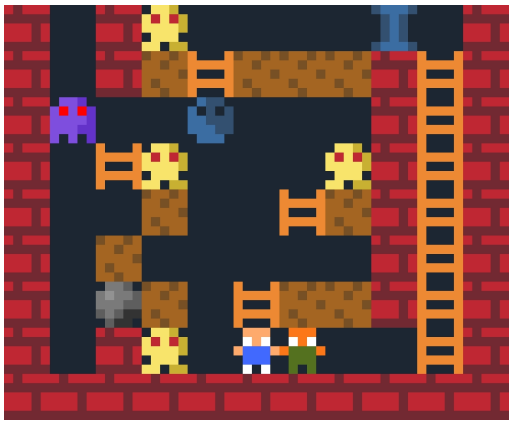


El caso más complicado se da cuando el protagonista camina a una posición adyacente, rompiendo un muro destruible o eliminando a un enemigo volador, a la vez que en esa posición adyacente no hay suelo, lo que provocaría:

A. La caída del protagonista.
B. La caída de todos los objetos afectados por la gravedad que estuvieran sobre el muro destruible/enemigo volador.
C. La caída de todos los objetos afectados por la gravedad que estuvieran sobre el protagonista antes de iniciar su paso a la posición adyacente.
D.



Para la generación procedural de niveles se combina algoritmos lógicos para seleccionar el tile adecuado en función del tipo de tiles que lo rodean, y funciones random condicionadas a cierta lógica para conseguir variedad en los fondos mediante columnas y rostros de ídolos.



Nivel en el editor de niveles

Nivel real dentro del juego

Gráficos y diseño

Los gráficos y el diseño de personajes están inspirados en series de anime de finales de los ‘80 y principios de los ‘90.

Cada uno de los protagonistas cuenta con hasta 20 frames diferentes para acometer todas las acciones que realizan en el juego, como caminar, empujar piedras, destruir enemigos, subir y bajar escaleras, saltar desde plataformas a escaleras y viceversa, saltar de una escalera a otra, empujar piedras desde escaleras…

Se ha tenido especial cuidado en el tratamiento de aceleraciones y desaceleraciones en las animaciones. Esto se aprecia, por ejemplo, al empujar una piedra, golpear enemigos o bajar por escaleras. Son acciones sin movimiento uniforme, sino con desaceleración, un detalle que ayuda a darle atractivo a las acciones del protagonista.

También se ha cuidado la anticipación al movimiento. Por ejemplo, antes de golpear a un enemigo, de empujar una piedra o de subir escaleras el protagonista destina unos frames a preparar su acción, lo que revierte en unas animaciones más “realistas” y menos “robóticas”.

Se ha optado por una fuente tipográfica con mayúsculas y minúsculas, posibilidad de elegir varios colores y con diferentes anchos de letra (como es el caso de la ‘i’ o la ‘l’) para presentar textos más atractivos.

Por último, se realiza un tratamiento especial de los fondos de pantalla para disminuir la sensación repetitiva (trama) y darle más variedad a áreas que pueden llegar a ocupar un gran tamaño. Para llevar a cabo esto contamos con tiles de fondo diferentes para las filas pares y las impares, además de 2 versiones por cada tile: la versión iluminada y la versión oscura. Además, se incorporan columnas y rostros de ídolos con el mismo objetivo: conseguir mayor variedad y menor sensación de repetición.

Música y FX

El juego consta de 4 melodías:

- Intro
- Menú principal
- In-game
- Fanfarria al completar un nivel.

Todas las melodías se han trabajado con especial atención para lograr coherencia entre todas ellas, y que juntas transmitan un estilo único y particular.

La música in-game, se ha tratado de una forma muy particular para permitir dejar un canal lo suficientemente libre como para no interferir con los efectos sonoros.

El juego cuenta con hasta 10 efectos sonoros diferentes que se ejecutan al empujar piedras, destruir bloques de arena, al caer elementos, saltar, etc. en pro de conseguir una experiencia de juego plena.

Por supuesto, el ahorro en memoria también ha condicionado este desarrollo, donde en sucesivas iteraciones se ha trabajado la definición de los instrumentos y la estructura de patterns para conseguir ahorrar el máximo de bytes con el mínimo perjuicio en calidad.

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:

- CPCtelera 1.4 - Astonishingly fast Amstrad CPC game engine.
- WinAPE 2.0b2 - Emulador de Amstrad.
- Sublime Text - Editor de código.
- Aseprite - Creación de sprites y pantalla de carga.
- Arkos tracker - Creación de Música y FX.
- PuzzleScript - creación de niveles y control de calidad.
- Trello - Organización de tareas.
- Google Drive - Almacenamiento de fuentes y materiales.
- BitBucket - Repositorio de código remoto Git.
- SourceTree - Gestor de repositorio de código Git en BitBucket.
- BabasLevelParser - Herramienta propia para codificar los niveles en función del código generado con puzzlescript.
- MemoryMapCalculator - Herramienta propia para calcular la ubicación de cada asset gráfico o de datos en la zona baja de memoria.
- Google Docs - Generación de la documentación.

Créditos

Code & Graphics: Rafa Castillo (@azicuetano)
Music & Sound Effects: John McKlain (@elmcklain)