Making of

KIMOHIYOKO

Al igual que la astrofísica, KimoHiyoko nació una cálida noche de verano en una ciudad del Mediterráneo. Andaba yo perezoso y aburrido deambulando por Twitter después de volver de mis vacaciones cuando di con este tuit:

Siempre estoy buscando gamejams en las que participar, especialmente la Ludum Dare y las de itch.io, y el CPCRetroDev es esencialmente una gamejam con su deadline y sus reglas. Ya la conocía de ediciones anteriores y por una vez me había enterado a tiempo para poder participar.

Además encajaba con otra de las cosas que había estado investigando últimamente como es el desarrollo para sistemas retro. Estos suelen tener una barrera de acceso bastante alta. Cada máquina cuenta con un hardware y una arquitectura diferentes y específicas. Montar un toolchain y un entorno adecuados para su desarrollo no es tarea sencilla por no hablar del reto de hacer un juego completo en ensamblador.

Aunque conocía de la existencia de CPCtelera nunca había probado a experimentar con ella. El vídeo de Fran Gallego explicando cómo funciona y creando un ejemplo sobre la marcha me animó a lanzarme a empezar mi propio proyecto. No imaginaba que la instalación fuera tan sencilla como una línea de comando y que en cuestión de minutos pudieras tener listo todo el entorno para desarrollar en Amstrad.

Así es como decidí a lanzarme a expermientar con CPCtelera, con las dudas sobre el rendimiento de lo que podría crear al usar por primera vez una nueva tecnología (si es que entendemos el Amstrad CPC como “nueva tecnología”) y esperando atascarme en algún momento y abandonar. Esto no sucedió y a los pocos días ya tenía mi primer protoengine montado sobre el que empezar a montar un juego.

MAKING OF KIMOHIYOKO
2018 Álvaro Amador Ruiz
@sararyCow

Llegaba el momento de decidir el juego que quería hacer. Desde que empecé el proyecto empecé a analizar el catálogo de Amstrad, tanto juegos clásicos como modernos para conocer las posibilidades técnicas de la máquina y las tendencias en géneros y temáticas de su software. Fueron horas probando juegos, visionando gameplays en Youtube y leyendo el libro de Génesis para conocer los mitos fundacionales y los rasgos característicos de la Edad de Oro del Software español; una enorme biblioteca lúdica que me perdí por edad y en la que tenía intención de profundizar en algún momento de mi vida tal como había hecho con los desarrollos japoneses y norteamericanos de la época. Esta fue la ocasión perfecta para ponerme a ello, y espero seguir aprendiendo más de este renglón de la historia de los videojuegos en el futuro.

Una de las conclusiones de mi análisis fue que los juegos de Amstrad tienden a ser muy cortos. Una vez sabes lo que hay que hacer pocos duran más de una hora, la mayoría mucho menos (hay gameplays de juegos completos de 10 minutos). Mirando los mapas de los juegos en CPC-POWER (otra enorme y fantástica recopilación) vi que pocos sidescrollers superan las 60 pantallas, teniendo la mayoría unas 40 o menos, y eso que en la mayoría la pantalla de juego solo ocupa una fracción del total. Esto es sin duda para que quepan en los 64Kb de memoria, limitación a la que me costó acostumbrarme en un primer momento.

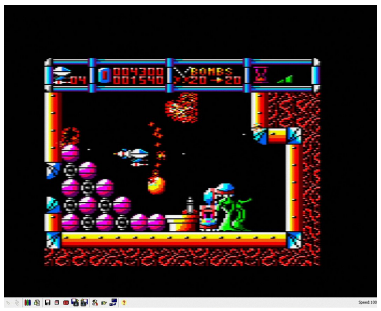
Que quepa tan poco contenido en los juegos hace que estos no puedan desarrollar profundidad en sus mecánicas, y basan su duración en incrementar la dificultad para que el progreso se consiga a base de ensayo y error.

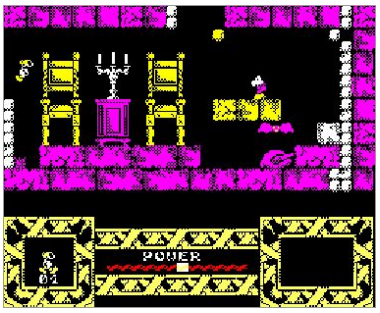
Yo que crecí con la Game Boy y la SNES en adelante estoy acostumbrado a juegos que sin dejar de lado la acción aportan algo más al jugador para motivarle a seguir la aventura. Llevaba tiempo queriendo hacer algo como Metroid (NES, 1986) que basa sus mecánicas más en la exploración que en la acción, y cuenta con un mundo abierto en el que el progreso se obtiene obteniendo mejoras con las que poder superar obstáculos antes infranqueables.

El otro rasgo característico de los metroidvanias es que al empezar la partida el jugador no tiene absolutamente nada y es vulnerable al ataque más débil, mientras que al final del juego cuando ha obtenido todo lo necesario para enfrentarse al jefe final es prácticamente invencible. Quería transmitir esa sensación de “nada a todo” en mi juego. Además parece que es un género en el que Amstrad escasea (aunque hay quien dice que es donde se inventó el género).

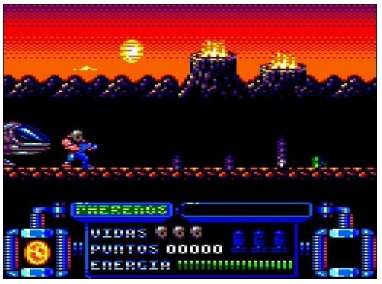
Por último estaba la temática. Desde hace tiempo tenía en mi baúl mental de ideas ese concepto de algo como un robot o un humano miniaturizado explorando el interior de un cuerpo y sanando al paciente desde el interior. No tenía mucho tiempo para pensar una historia y una temática así que recurrí a eso como concepto inicial.
Así pues ya tenía todos los ingredientes del juego que quería hacer:

Referentes CPC clásicos:



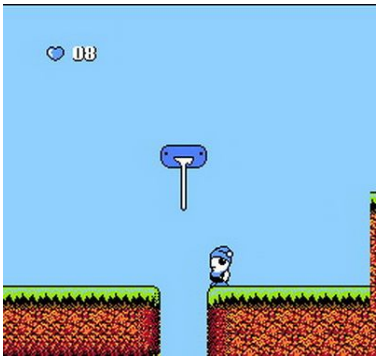


Referentes CPC modernos:

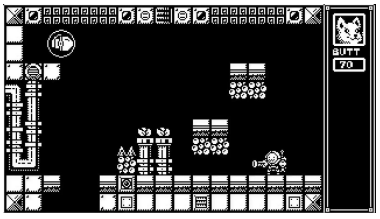
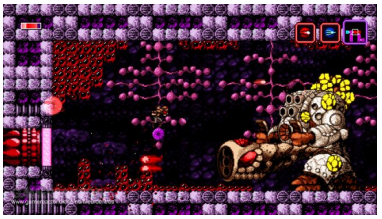
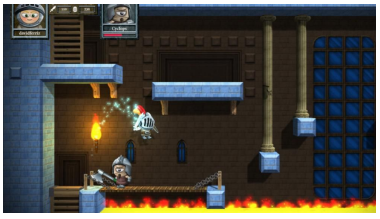




Referentes 8-bits:





Referentes modernos:

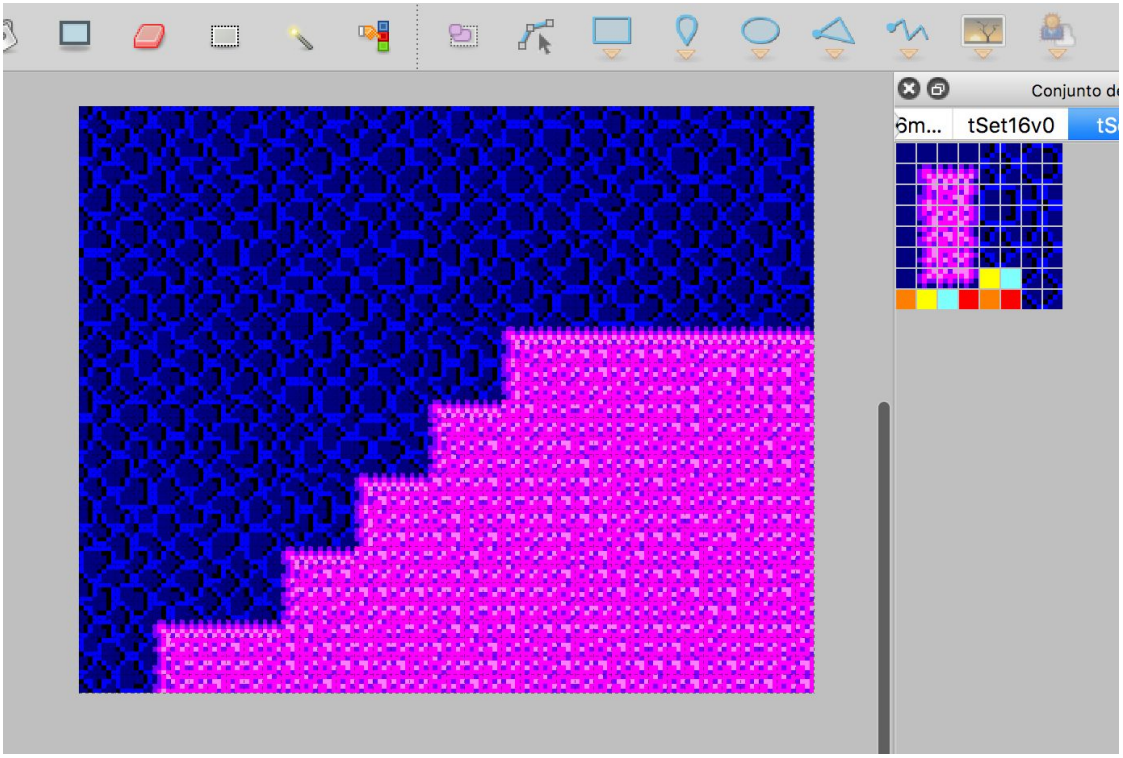
Referentes culturales:

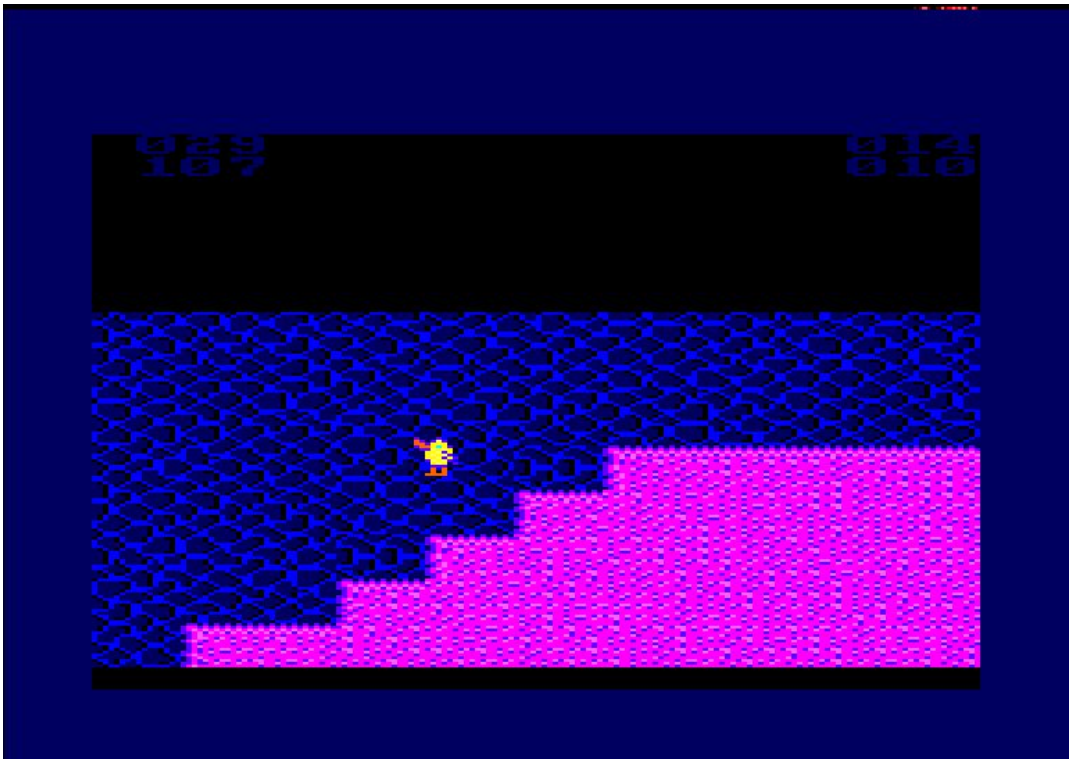


MAKING OF KIMOHIYOKO
2018 Álvaro Amador Ruiz
@sararyCow

Así ya solo quedaba ponerse manos a la obra.

Desde el primer momento sabía que el espacio en memoria iba a ser un problema así que opté por usar tiles mínimos. Esto es de 2*4 bytes (4*4 píxeles). Este fue el resultado:





Primer tileset y primer prototipo de KimoHiyoko

En seguida me di cuenta de varios problemas:

- Con pocos tiles no iba a poder dar variedad gráfica al juego.
- En ese momento la zona de juego era de 40*32 tiles, esto es 1280 bytes por pantalla. Así no cabrían muchas.
- Aunque los tiles estuvieran diseñados para interconectar entre ellos había que colocarlos uno a uno, para dar variedad a los escenarios o crear reglas específicas de AutoMap en Tiled para cada uno (lo cual es un engorro).
- Así solo se almacenaría el escenario en sí, teniendo que crear estructuras para guardar los datos de los enemigos e ítems que podrías encontrar en cada una con el consecuente gasto de tiempo, esfuerzo y memoria en ello.
- El formato. Los píxeles-ladrillo del modo 0 de Amstrad hacen que todo se vea más aplastado de lo que debería. El formato cuadrado se convierte en panorámico y el panorámico en Panavisión. No me convencía esa deformación.

Aunque los tiles eran diferentes la funcionalidad a nivel interno solo son 2 estados (transitable o obstáculo), por lo que almacenar la ID de cada uno individualmente es un desperdicio. En lugar de almacenar todas las IDs se podrían guardar metaIDs que representaran los posibles tiles que puede pintar en esa posición y luego calcularlos algorítmicamente durante la carga del nivel.

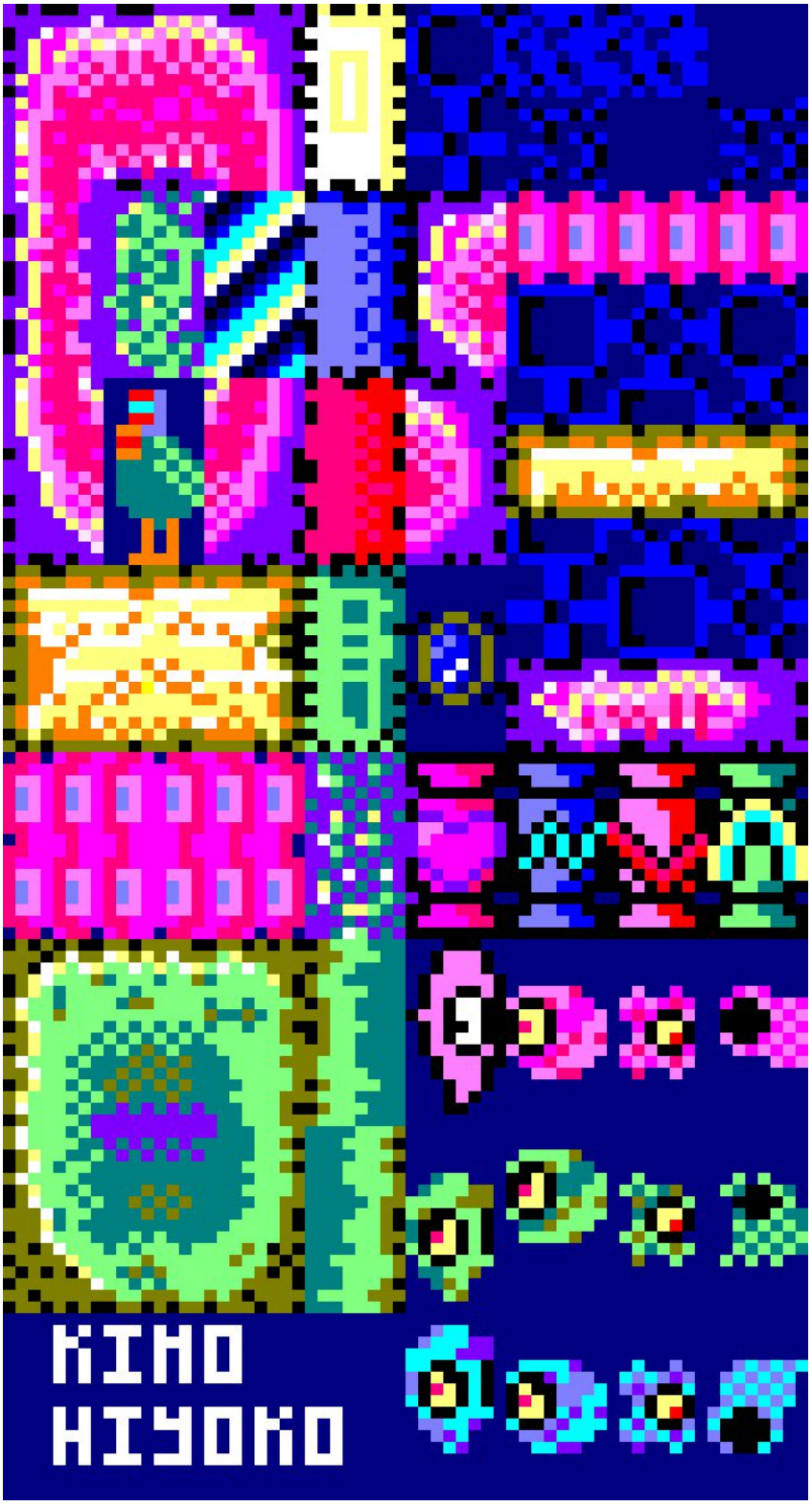
Así es como empecé a concebir un nuevo sistema de tiles que optimizara lo más posible los problemas encontrados en la versión anterior. Se me ocurrió un sistema de metatiles que en lugar de representar un solo tile de la pantalla final ocupara un espacio de 2*4 tiles. Estos tiles verticales el doble de altos que de anchos mitigarían el efecto de aplastamiento y permiten ocupar una sola pantalla de 128*192 con 16*12 tiles. Si además estos metatiles se limitan a 64 o menos se pueden almacenar a 6 bits cada uno en lugar de 8. Restando una fila para el HUD cada nivel se queda en 16*11*6/8=132 bytes por nivel, casi 10 veces menos que en el planteamiento original.

Cada uno de los metatiles son reglas que calculan los 8 subtiles que pinta en pantalla. Algunas reglas simplemente dibujan el grupo de tiles asignado tal cual, mientras que otras combinan distintos tipos de tiles para formar distintos entramados en la pared o cogen tiles “de aquí y allá” para crear los distintos tipos de plataformas a distintas alturas o formar distintas plataformas combinando tiles.

Además con este sistema podría crear reglas para introducir las coordenadas de enemigos e ítems “gratis” pues estarían incorporadas en los datos del escenario.

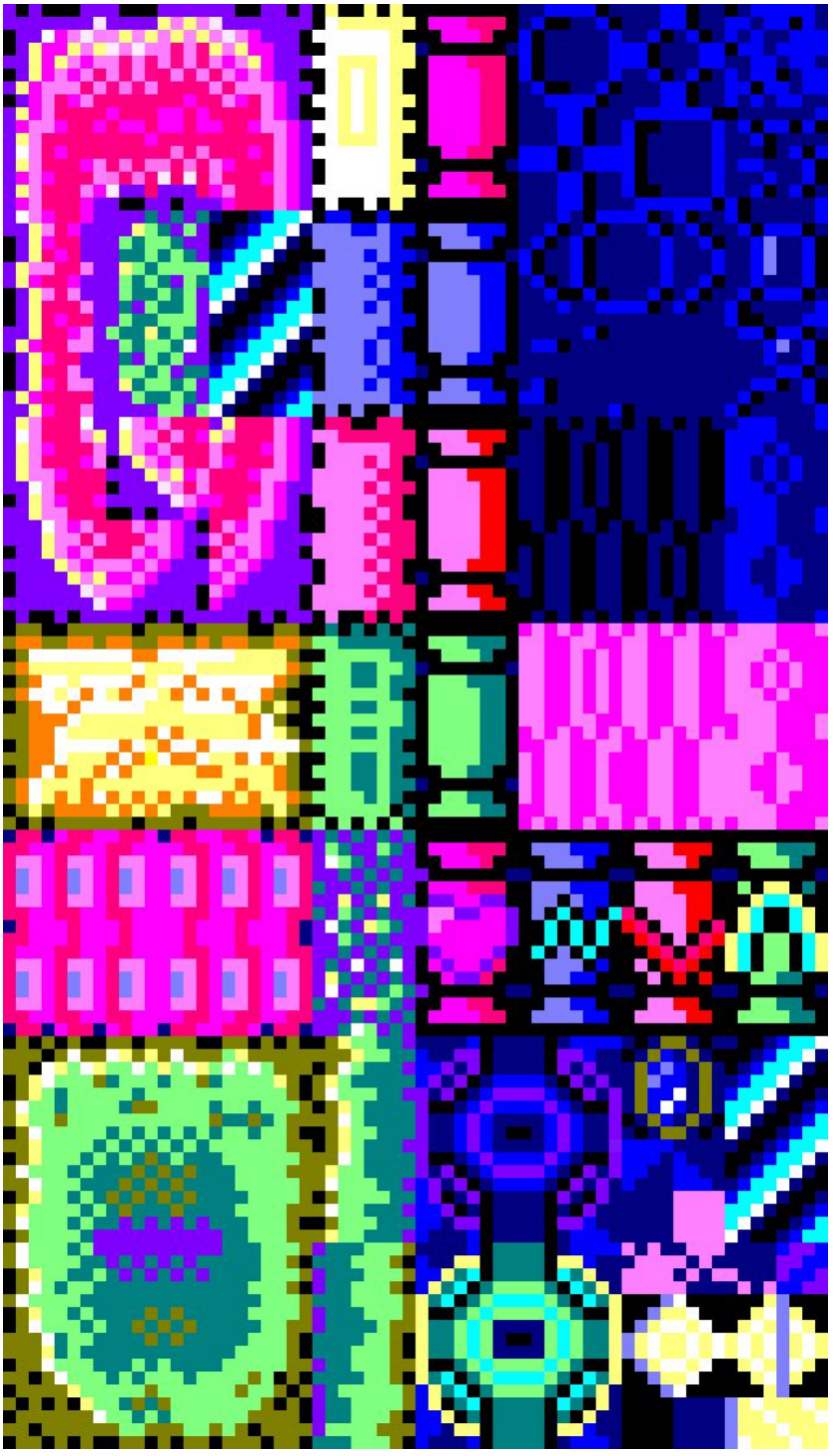
Tileset

Así este es el metatilset; no presente en el juego, solo se usa en Tiled para construir los niveles:



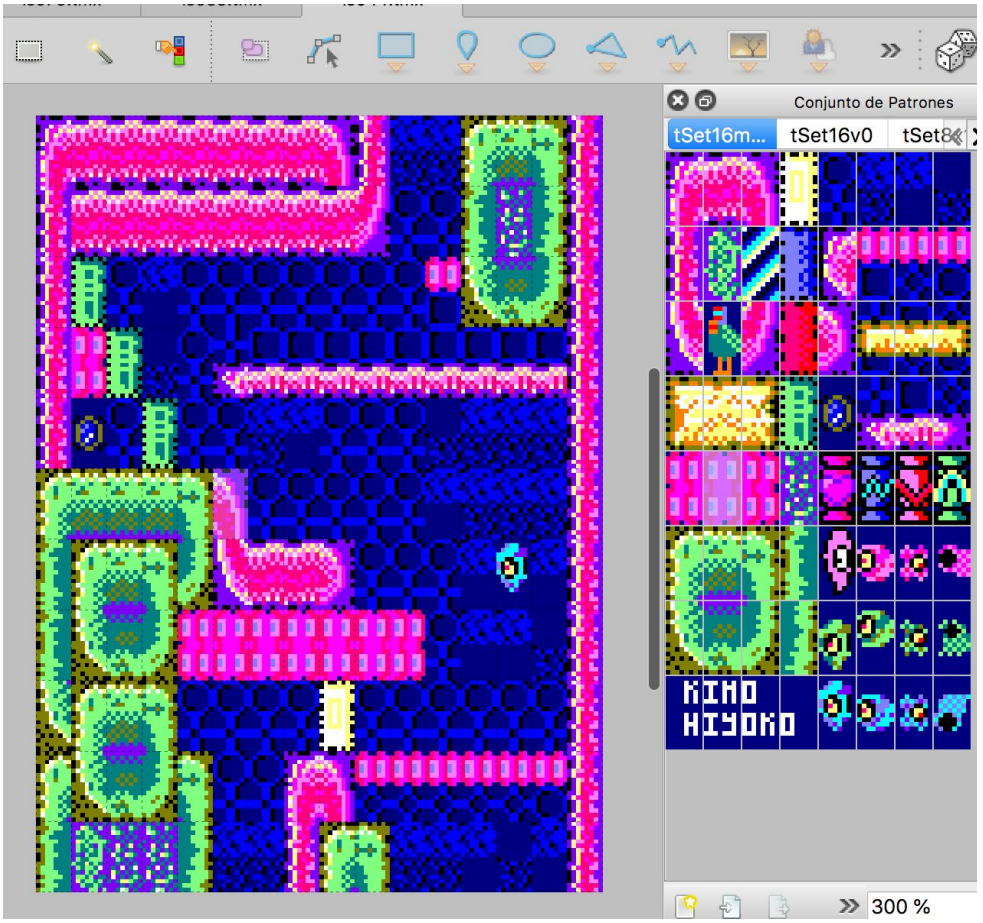
Metatileset final de KimoHiyoko

Y este es el tileset guardado en el juego, 448 tiles que ocupan 3,584Kb. En él están también guardados los tiles con los que se forman las pantallas de inicio, intro, título, final y el HUD:



Tilset final de KimoHiyoko

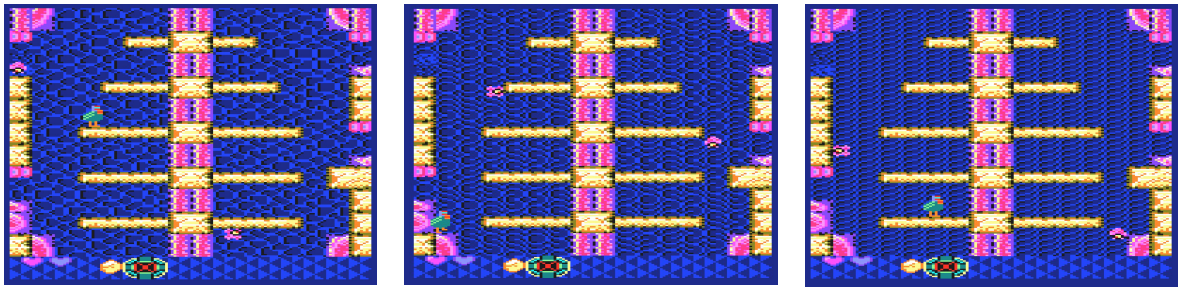
Así se puede construir un Tiled un nivel como este:



Y este es el aspecto que tiene en el juego final:

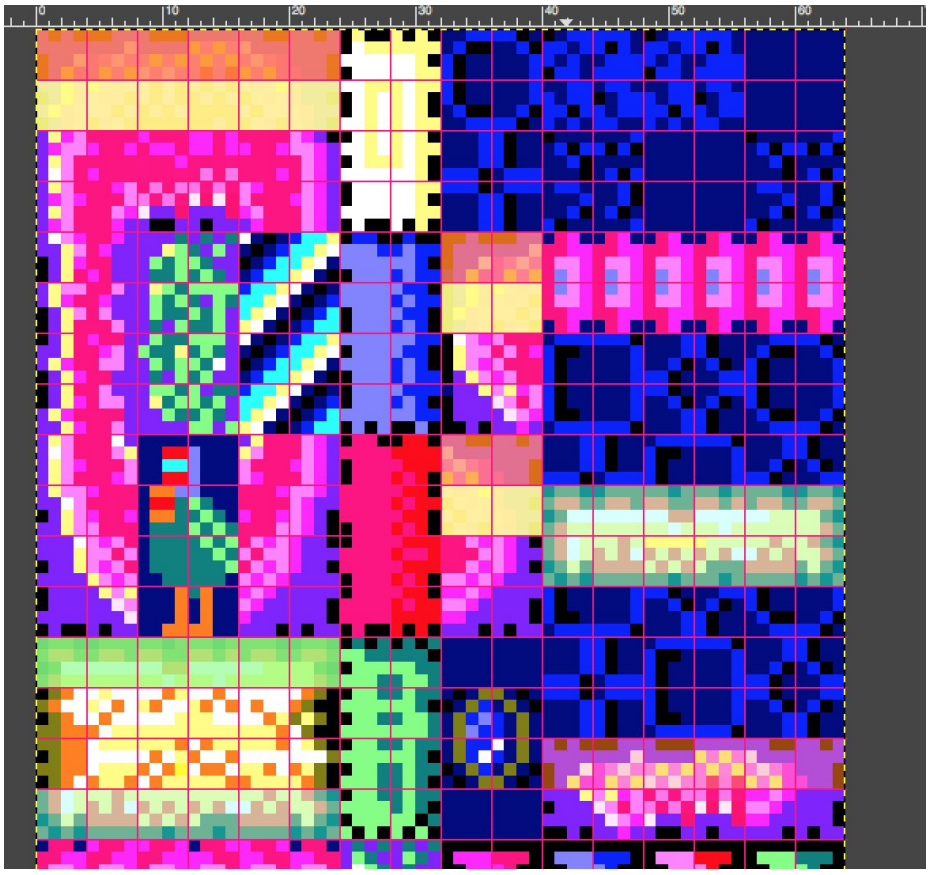
MAKING OF KIMOHIYOKO
2018 Álvaro Amador Ruiz
@sararyCow

En las imágenes de arriba se puede apreciar como el resultado es prácticamente idéntico salvo por los tiles del fondo. Esto se debe a que el entramado del fondo cambia según los metadatos de cada pantalla. Los mismos tiles se pueden combinar de distintas formas para formar distintos patrones:



Distintos patrones de fondo usando los mismos tiles

Por otra parte también se reutilizan tiles en distintas reglas para crear distintas plataformas. Esto permite aportar más varierdad gráfica al juego a un coste de memoria muy bajo:



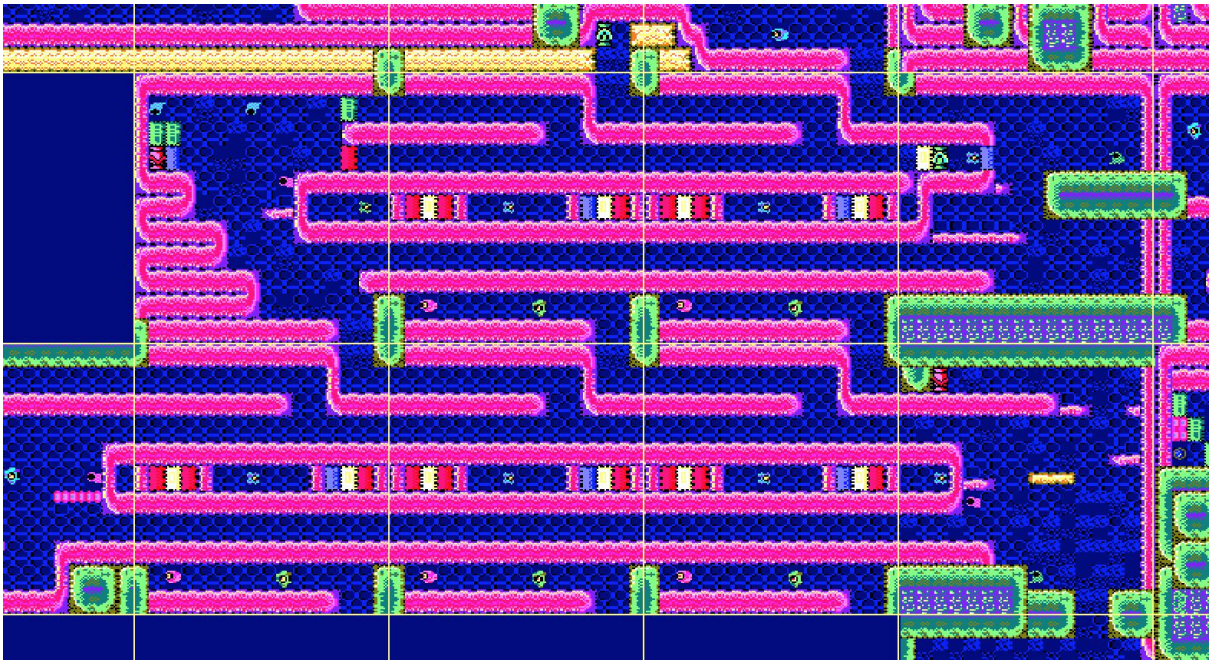
Detalle del metatilset. En él se aprecian los subtiles que se repiten dentro de distintas reglas. Los tiles coloreados del mismo color son los mismos en unas y otras, pero combinados de distinta forma.

Así también se pueden crear plataformas de distintos tamaños, como las plataformas finas que también son “one way up” lo cual no solo aporta al juego mayor variedad gráfica si no mayor variedad jugable también.

A cada pantalla le acompañan 8 bytes de metadatos con la siguiente información:

- pantallas adyacentes por arriba, abajo, izquierda y derecha (4 bytes)
- puntero al metatilset correspondiente (2 bytes)
- patrón de fondo (2 bits), entramado de fondo (2 bits), música de la pantalla (2 bits) (todo cabe en un 1 byte)
- byte extra para almacenar información relevante de la pantalla (si contiene ampliación, el índice necesario para chequear si se ha cogido ya ese ítem o no; si contiene un teletransporte, la pantalla a donde te lleva, etc).

Este sistema permite reutilizar pantallas en distintos lugares o encajándolas como un puzzle, incluso cambiándole totalmente fondos y música para darle mayor variedad a los escenarios, al módico precio de 8 bytes por pantalla extra. Esto se hace en el área del esófago (ver ilustración de ejemplo de variación de fondos) o en el área del laberinto del intestino:



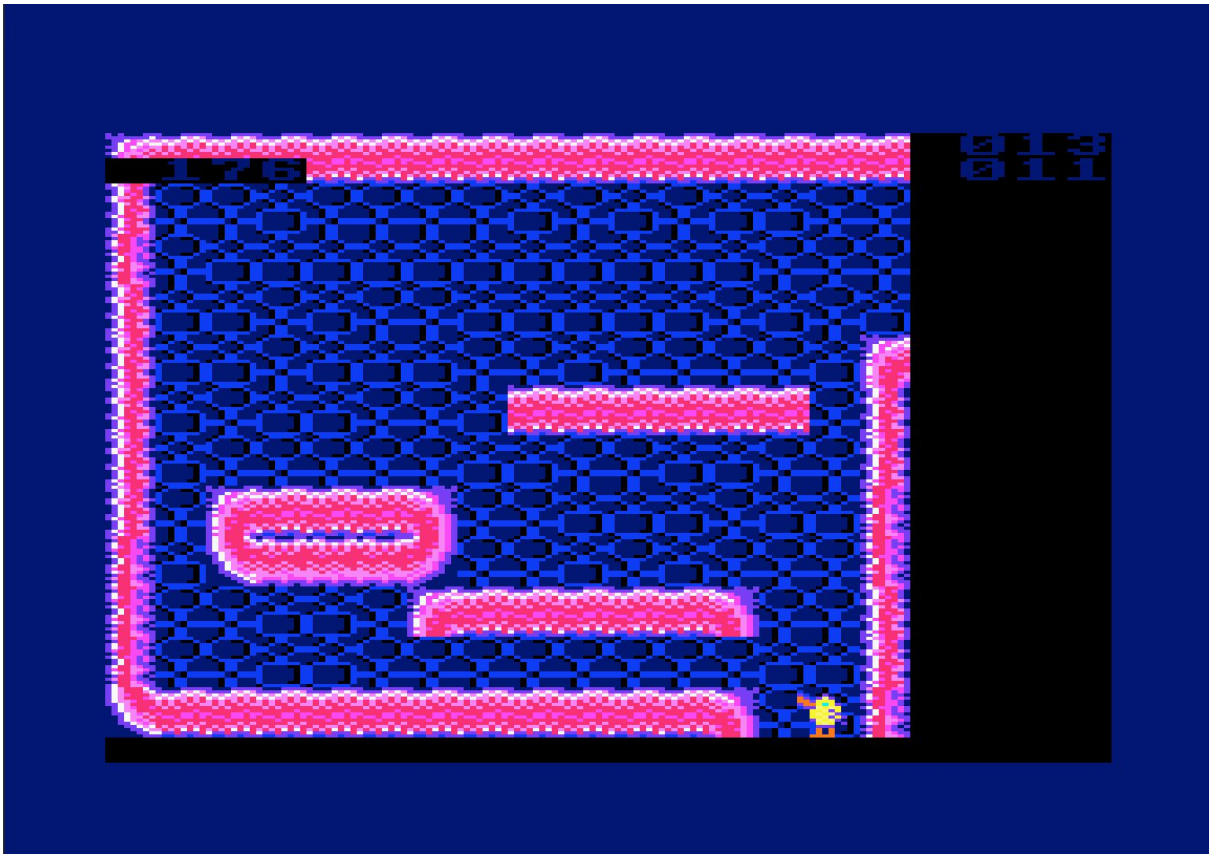
La misma pantalla se reutiliza hasta 5 veces para formar este laberinto

También se utiliza esta técnica en el laberinto final dentro del corazón, más adelante se discute esta parte más en detalle.

Finalmente este sistema ha permitido diseñar y montar fácilmente un mapa de 84 pantallas, de las cuales 75 son completamente disintas.

Formato

Como ya se ha comentado en la sección anterior no me convencía el formato panorámico del Amstrad, así que desde el principio del desarrollo opté por un formato de pantalla de juego de 128*192 px para que el juego tuviera un aspecto más cuadrado, dejando todo el margen sobrante para el HUD. Fue entonces cuando leyendo los foros de Amstrad me enteré de que se podían modificar los registros del CRTC para ofrecer dicha resolución en pantalla dejando el espacio de memoria sobrante para lo que quisiera. No solo podía poner el formato de pantalla que quería, si no que se liberaban 8Kb extra que a la postre resultaron imprescindibles todos y cada uno de ellos.



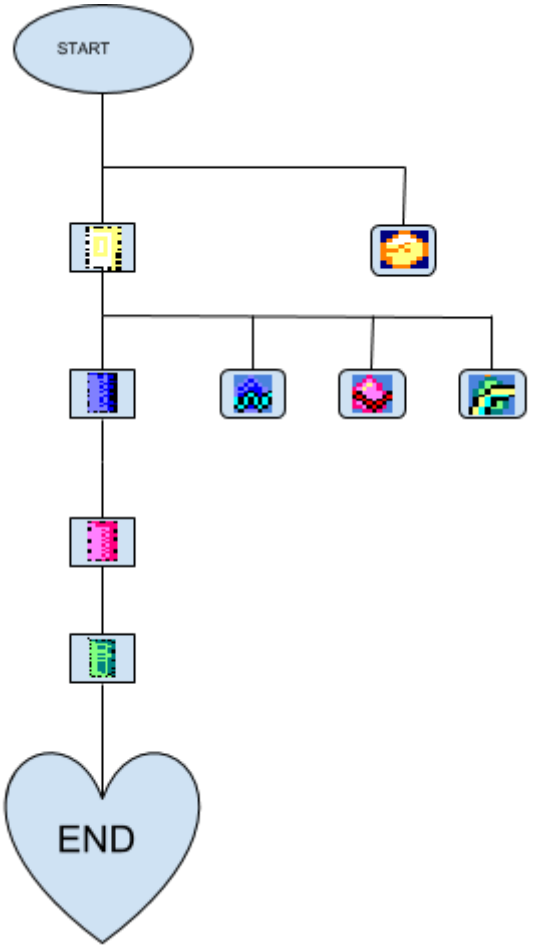
Estado del desarrollo al cabo de 2 semanas aproximadamente

No me costó demasiado tomar la decisión. Lo único necesario para ello fue crear una función para modificar el CRTC (única función del juego escrita en ensamblador nativo) y modificar los scripts de CPCtelera de dibujado de sprites para que pinten en las nuevas direcciones de memoria asignadas a cada pixel (al final en el juego la única rutina modificada que se utiliza es `cpct_drawSpriteMaskedAlignedTable`).

Esta función para modificar el CRTC demostró ser un acierto pues con ella se pueden conseguir efectos de transición entre pantallas como la que hay entre la intro del juego y la pantalla de título o el screen shake (nota: no puedes hacer un juego indie en 2018 sin screen shake).

Gameplay

El mapa de Kimohiyoko está diseñado para poder explorarlo de forma no lineal una vez has superado la parte del tutorial y hasta la parte final en el corazón, tal y como muestra este diagrama de Mark Brown:



Esto da la posibilidad al jugador de “perderse” por el mapeado sin por ello sentir que está perdiendo el tiempo, pues todo el rato encontrará armas y mejoras que recoger.

Si bien es cierto que se puede recorrer en el orden que se quiera, sí existe un camino “óptimo” para recoger lo más posible en la menor cantidad de tiempo. Averiguarlo es algo que dejo para los speedrunners (si es que los hay). Por otra parte no es necesario recorrer todo el juego para completarlo, en teoría se puede completar sin recoger ninguna mejora de vida o arma (otro reto para youtubers ;P).

En el diseño de niveles se ha tenido en cuenta la colocación de los objetos clave para colocar a los enemigos según el tipo de armas que podrías o deberías tener en ese punto. A la hora de colocar a los enemigos en detalle dentro de cada pantalla también se ha tenido en cuenta la vulnerabilidad de los enemigos según su tipo y las posibles trayectorias de las

armas más adecuadas para atacarles, de modo que suele haber una forma “más fácil” de lo que parece de superar una pantalla.

Armas y Enemigos

Además de la estándar que consigues al principio y tiene munición infinita, hay tres tipos más de armas en KimoHiyoko. Cada una tiene un color representativo: azul, verde y rojo. Los enemigos a su vez también son de ese mismo rango cromático. Si se experimenta lo suficiente con las armas se puede descubrir que cada tipo de arma afecta a cada tipo de enemigo de una forma, siguiendo una tabla simplicada de la dependencia entre los tipos de los Pokémon primarios de cualquier juego de Pokémon (es decir la de los tipos planta, agua y fuego):

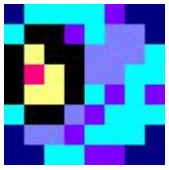
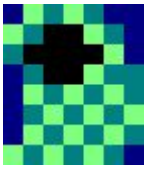
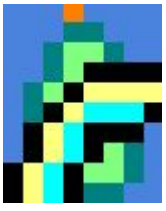
			
	-2	-3	-1
	-1	-2	-3
	-3	-1	-2

Tabla de tipos en KimoHiyoko

El arma estándar por su parte siempre inflinge 1 de vida pero tiene la trayectoria más controlable y es infinita, por lo que debes valorar en todo momento qué arma emplear en cada situación.

Cada clase de enemigo cuenta con una máquina de estados diferente como IA, de modo que es sencillo aprender los patrones de los enemigos y esquivarlos o atacarlos como se desee.

Una vez dominas todos los conceptos y has obtenido suficientes mejoras puedes alcanzar esa sensación de “invencibilidad” que se busca en los metroidvanias, pudiendo avanzar por donde antes solo habían obstáculos.

Música

En el apartado musical han confluido una serie de factores:

- Cuando concebí el juego no lo imaginaba con música (aunque sí le hubiera sentado bien un tema principal) si no con sonidos de ambiente dentro del cuerpo generando una atmósfera tétrica y oscura.
- El escaso espacio en memoria restante para incluir música en el juego
- El escaso (nulo) talento musical del autor

La convergencia de estos factores fue que la banda sonora de KimoHiyoko es en realidad el sonido del corazón del huésped, que puedes escucharlo en todo momento desde cualquier punto. Existen varias variaciones de los latidos del corazón que indican cuando el jugador se está acercando a un punto sensible y se intensifican aún más cuando llega a dicho punto sensible. Esto viene a ser un sistema rudimentario de música adaptativa que responde a los eventos del juego.

Por otra parte, todas las variaciones de los latidos del corazón tienen como base un patrón de reggeaton. Si eso no crea una atmósfera tétrica y oscura no sé qué más hacer.

El corazón

Llegar al corazón es el objetivo final en KimoHiyoko. Todo lo que tiene que hacer nuestro nanopollo es recoger las muestras de la infección y diseminarlas por las venas junto al antídoto que lleva.

Cuando por fin has llegado te das cuenta de que la aventura no ha terminado, hay que encontrar el torrente sanguíneo. Pero el corazón es un laberinto en el que vayas por donde vayas siempre acabas en el mismo sitio.

Para encontrar el camino que lleva al final hay que estar atento durante la partida. Las pantallas donde obtienes las armas tienen en el fondo unas inscripciones: una flecha y un número. El número representa cada pantalla del corazón, y las flechas la dirección que hay que tomar en cada una. Si no has tomado nota de esto aún hay una posibilidad de resolverlo probando al azar siguiendo los latidos del corazón del Hiyo huésped.

Este epílogo es un homenaje al recurso del laberinto repitiendo pantallas, un socorrido aunque sucio truco usado en juegos antiguos para alargar la vida de los títulos. Juegos tan míticos como Super Mario Bros. (NES, 1984) o The Legend of Zelda (NES, 1986) lo emplean.

Al llegar al final aparece el ending y después puedes consultar las estadísticas de la partida (tiempo, muertes, y porcentaje de ítems recolectados).

El mensaje de KimoHiyoko

Al final, Kimohiyoko es una metáfora de la vida. La pasas buscando el corazón, y cuando lo encuentras este resulta ser un laberinto sin salida. Si caes puedes abandonar, o puedes volver a pulsar 1 y afrontar de nuevo los retos que surgen más allá de la zona de confort. Eso es KimoHiyoko, y eso es también la vida.

Espero que os guste :)
Álvaro (@sararyCow)