Manual de PIX37

Primero editor, luego código. Copia impresa del libro en pix37.party/docs.

Editor

PIX37 DEV es el editor. Aquí dibujas, escribes y escuchas el juego, y luego exportas un cartucho o una compilación independiente. También puede reproducir un cartucho y abrir PIX37.PARTY para descargar cartuchos y usar el catálogo. PIX37 DEV EDUCATIONAL es el mismo editor y es gratuito. No tiene compilaciones, ni exportación de cartuchos, ni acceso a PIX37.PARTY. PIX37 Console es el reproductor. Reproduce cartuchos `.p37` y abres PIX37.PARTY para descargar cartuchos y explorar el catálogo.

Proyectos y cartuchos de juegos

Un archivo `.pix37` es el proyecto: pestañas de código fuente, hojas de sprites, mapas, sonido y tablas de datos. Un archivo `.p37` es el cartucho del juego que ejecuta el jugador: código de bytes compactado.

Los archivos de guardado y los logros se encuentran en la carpeta del autor, junto a los datos de la aplicación. En Windows, esta carpeta se ve como `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` para un archivo de guardado y `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` para los logros. En Linux, los mismos archivos son `~/.local/share/PIX37/Mira/Pong.p37s` y `~/.local/share/PIX37/Mira/Pong.ach`. Ada es la cuenta en el ordenador. Mira es el nombre del autor, y ese nombre corresponde a la carpeta. Pong es la ranura de guardado, que por defecto es el título del juego. `save` escribe el archivo `.p37s`. El primer logro desbloqueado escribe el archivo `.ach`. Si los archivos ya existen, el programa los lee al iniciar el cartucho del juego.

Menú de desarrollador

La tecla Escape abre el menú de desarrollador. Si la partida está en curso, al pulsar Escape se regresa al editor. Confirma con la tecla Enter o el botón A (V por defecto). Para retroceder, pulsa el botón B (C por defecto).

Sin ningún proyecto cargado, el menú es: CREAR PROYECTO, CARGAR PROYECTO, COMPILAR, CARGAR cartucho de juego, CONECTAR A PIX37.PARTY, AJUSTES.

Con un proyecto cargado, se puede: GUARDAR PROYECTO, CARGAR PROYECTO, CREAR PROYECTO, ACTIVOS, COMPILAR, CARGAR cartucho de juego, CONECTAR A PIX37.PARTY, AJUSTES.

Ctrl+S guarda el proyecto abierto. Ctrl+R lo ejecuta.

Crear un proyecto

La función CREAR PROYECTO solicita el cartucho del juego antes de comenzar a crear el proyecto.

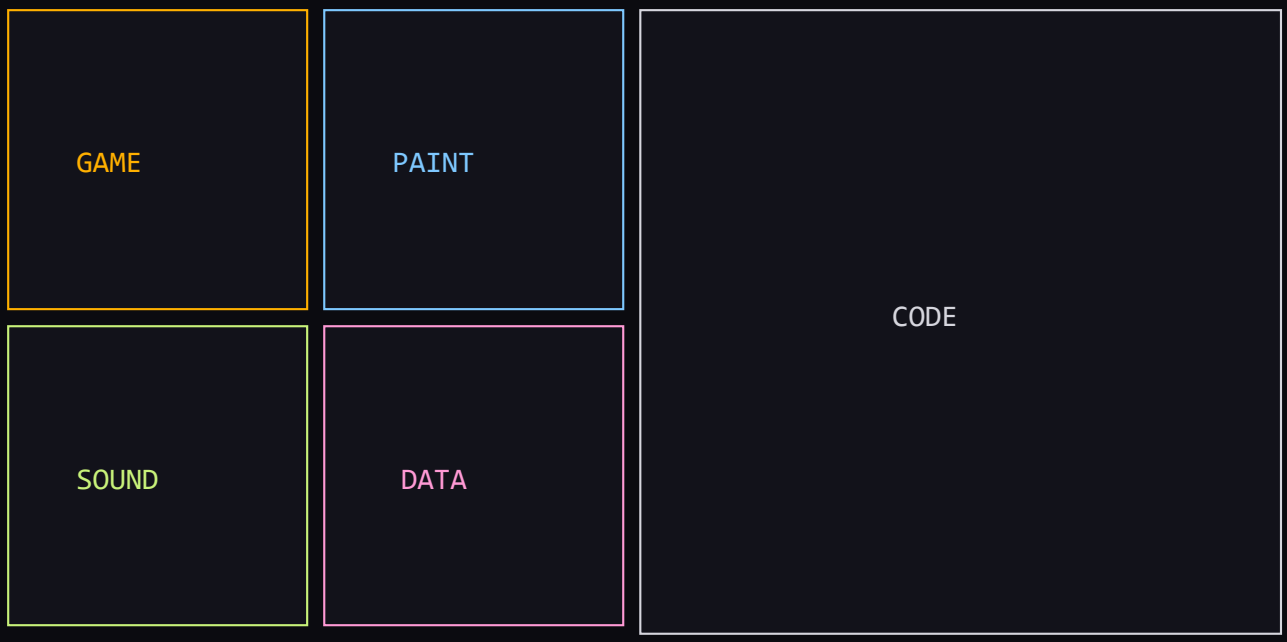
- **VIDEO** — 1 BIT (2 colores), 2 BIT (4), 4 BIT (16), 8 BIT (256).
- **AUDIO** : profundidad de muestreo del sintetizador: 16, 8, 4, 2 o 1 bit.
- **PANTALLA** : los ajustes preestablecidos incluyen 64×64, 128×128, 160×90, 256×256 y 320×180. El tamaño PERSONALIZADO le permite elegir cualquier tamaño desde 1×1 hasta 512×512.
- **PALETAS** — bancos de colores. El primer banco comienza con el conjunto de 16 colores que se muestra a continuación. CARGAR PALETA aplica un conjunto integrado: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 y una lista más extensa tomada de Lospec (Resurrect 64, Apollo, Endesga 32 y 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century y otros). También puede cargar un archivo de paleta o cargar su propio PNG y extraer la paleta de esa imagen.



Índices de color utilizados por `cls`, `pset`, `spr` y el resto. Puedes volver a pintar cualquier índice en el editor de paletas.

Los cinco paneles

El editor se compone de JUEGO, PINTURA, CÓDIGO, SONIDO y DATOS, en ese orden. El mapa se encuentra en PINTURA. Los efectos de sonido y las canciones se encuentran en SONIDO. Haz clic en un panel para trabajar en él. Cuando CÓDIGO tiene el foco, SONIDO y DATOS ignoran las teclas de letras, por lo que al escribir no se reproduce el piano.



Diseño completo, escrito `1:1:4:1:1`. Cada número indica qué parte de la fila ocupa el panel: JUEGO, PINTURA, CÓDIGO, SONIDO, DATOS. Una pantalla pequeña se ubica centrada en una celda de 128×128 píxeles. Una pantalla de más de 128 píxeles amplía la columna JUEGO.

AJUSTES → GLOBAL → DISEÑO cambia las disposiciones:

Código	Lo que ves
`1:1:4:1:1`	Todo. JUEGO y SONIDO apilados, PINTURA y DATOS apilados, CÓDIGO a la derecha.
`1:1:4:1:1 2X`	La misma disposición. El código se dibuja con el doble de densidad, por lo que cabe el doble de texto.
`1:0:1:0:1`	JUEGO, DATOS, CÓDIGO.
`1:0:1:0:1 2X`	La misma disposición, con el CÓDIGO más denso.
`0:1:1:1:1`	Sin vista previa del juego. PINTURA, CÓDIGO, SONIDO, DATOS.
`0:1:1:1:1 2X`	La misma disposición, con el CÓDIGO más denso.
`0:1:4:1:2`	Estudio. Código ancho, una tira de datos más alta, pintura y sonido.
`0:1:4:1:2 2X`	La misma disposición, con el CÓDIGO más denso.

`0:0:4:0:4`	Solo CÓDIGO y DATOS, ambos de gran tamaño.
`0:0:4:0:4 2X`	La misma disposición, con el CÓDIGO más denso.
`0:1:4:1:0`	PINTURA, SONIDO y un amplio CÓDIGO.
`0:1:4:1:0 2X`	La misma disposición, con el CÓDIGO más denso.
`0:4:0:0:0`	Solo pintura, 256×256, un píxel de la hoja por un píxel del lienzo.

Una fila `2X` mantiene los mismos paneles. El panel CODE se dibuja al doble de resolución y se ajusta a la misma celda, por lo que el texto, las pestañas y el margen tienen la mitad de altura y cabe el doble de código fuente en la pantalla.

JUEGO

Vista previa del cartucho del juego en ejecución. Haz clic en el panel JUEGO o pulsa Ctrl+R para ejecutar la vista previa. El panel se configura en AJUSTES → PANEL DE JUEGO.

- **Vista previa** : color de reposo, continuo o recarga automática al editar.
- **Jugar en una ventana nueva** : el juego se abre junto al editor.
- **Visualización** : vista previa en RGB888 o RGB565.
- **JUEGO 64** : una pantalla de 64×64 se centra 1:1 en la celda o se escala 2× para llenarla.
- **Shaders** : filtros CRT y del juego, independientes de los filtros del editor.
- **Color** : fondo del panel.

PINTAR

Hojas de sprites y, en el lado del mapa del mismo panel, mapas de mosaicos. Las hojas pueden ser más grandes que la pantalla. Los tamaños de vista previa de los sprites son 8, 16, 32, 64 y 128 (128 solo cuando la hoja tiene al menos 128 píxeles). Desplace la rueda del ratón sobre la vista previa del sprite para ampliarla. Los tamaños de pincel son 1, 2, 3 y 4.

Llave	Herramienta
B	Lápiz
METRO	Seleccionar rectángulo
Q	Lazo
S	Estampilla

GRAMO	Balde
I	Cuentagotas
V	Mover
L / Mayús+L	Línea / curva
U / Mayús+U	Rectángulo hueco / elipse hueca
F / Mayús+F	Rectángulo relleno / elipse rellena
D	Vacilar
[ ]	Pincel más pequeño/más grande
Ctrl+Z / Ctrl+Y	Deshacer/rehacer. Ctrl+Shift+Z también rehace.
Ctrl+C / Ctrl+X / Ctrl+V	Copiar, cortar, pegar
Ctrl+A	Seleccione el lienzo

El mapa utiliza la misma idea, con menos herramientas: B (lápiz), M (selección rectangular), S (tamaño), V (mover) y Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Mantén pulsada la tecla Alt para ampliar o reducir la textura del mapa. El tamaño del pincel en el mapa se mantiene con las herramientas de pintura.

Cada objeto tiene ocho bits de bandera. Paint los establece en el objeto; `fget` y `fset` los leen en el juego.

AJUSTES → EDITOR DE PINTURA es donde se configura ese panel:

- **Paleta** : qué banco se utiliza, además de cargar y guardar.
- **Vista previa** : RGB888 o RGB565.
- **Cursor del pincel** : el puntero mientras pintas.
- **Ajuste de píxeles** : cómo el pincel se ajusta a los píxeles.
- **Un arbusto grande** : un arbusto grande se rompe desde su parte superior izquierda o desde toda su base.
- **Piel de cebolla** : un fantasma de otros fotogramas, y hacia dónde mira ese fantasma.
- **Zoom del mapa** : píxeles reales o zoom libre.
- **Colores** : los colores de la interfaz de usuario de pintura.

CÓDIGO

Las pestañas son archivos separados dentro del proyecto. Haga clic en el nombre de una pestaña para cambiarle el nombre; presione Enter para confirmar. Varias pestañas, cada una definida como `_init`, `_update` o `_draw`, se combinan en orden de tabulación: la primera pestaña se ejecuta primero y la última, al final. El

código que no está dentro de un gancho pasa a formar parte de `_init`. Un cartucho de juego sin ganchos se trata como un único `_draw`.

Llave	Acción
Flechas	Mueva el cursor. La tecla Mayús amplía la selección.
Ctrl+Izquierda/Derecha	Por palabra.
Inicio / Fin, Ctrl+Inicio / Fin	Inicio o final de la línea o de la tabulación.
Pestaña	Inserta dos espacios. En la lista de autocompletar, acepta el nombre.
Ingresar	Nueva línea, con sangría igual que la anterior. Acepta una autocompletación.
Ctrl+C / X / V	Copiar, cortar, pegar.
Ctrl+Z / Ctrl+Y	Deshacer / rehacer.
Ctrl+A	Seleccione la pestaña.
Ctrl+D	Duplica la línea o la selección hacia abajo.
Ctrl+/	Comenta o borra el comentario con `--`.
Ctrl+F	Buscar. F3 salta al siguiente golpe. Escape cierra la barra.
Ctrl+G	Vaya a un número de línea.
Escapar	Cierra primero la función de búsqueda, la opción "ir a" o la lista de pistas. Si pulsas Escape por segunda vez, saldrás del editor y volverás al menú de desarrollador.

Aparecen sugerencias mientras escribes el nombre de una función. Presiona Enter o Tab para insertar el nombre y un `(` si COMPLETE (está activado en la configuración. CONFIGURACIÓN → EDITOR DE CÓDIGO:

- **Buscar** : Ctrl+F en la pestaña activa o en todas las pestañas.
- **Texto en alto** : recorta los glifos a 5 px o muestra el glifo completo.
- **Tamaño de cadena** : tamaño de fuente de las cadenas literales.
- **Fuente para comentarios** : 3×5, o la fuente Fusion del lenguaje de la interfaz de usuario actual.
- **Complete (** — si los tipos de finalización incluyen el paréntesis.
- **Sugerencias de código** : activadas o desactivadas.
- **Idioma de las sugerencias** : el idioma del menú, o inglés.
- **Espacio de circunflejo** : margen adicional en las líneas vacías.
- **Cúpula** — números de línea.
- **Color** : fondo, comentarios, tabulaciones, números, operadores, palabras clave, cadenas, funciones del motor, sus funciones, variables, corchetes, circunflejo, selección, línea de error, advertencia.

SONIDO

El panel tiene dos pestañas. SFX es un patrón: una breve secuencia de notas. SOUND es una canción: coloca esos patrones en ocho canales. TIC indica cuántos ticks espera un paso. El reloj marca 120 ticks por segundo, y un paso es una semicorchea, por lo que 15 ticks equivalen a 120 pulsaciones por minuto. Un TIC más alto indica un patrón más lento. Dos patrones permanecen sincronizados cuando un TIC divide al otro: 15 y 30 lo hacen, 15 y 8 no.

El piano suena al seleccionar una nota con el ratón. Las teclas blancas son ZXCVBNM (CDEFGAB). Las teclas negras son SDGHJ (C# D# F# G# A#). Coloca el cursor sobre cualquier valor y escribe un número, o bien, coloca el cursor sobre él y gira la rueda del ratón. Los números de tono en el código van de 0 (C-0) a 63 (D#5).

En un patrón, LOOP y LEN son los mismos dos números. OUT por encima de 0 repite desde IN hasta OUT, y el título dice LOOP. OUT en 0 e IN por encima de 0 es LEN: esa cantidad de notas se reproducen desde el principio, luego el patrón termina y el título dice LEN. Ambos en 0 reproducen cada paso.

La cantidad de sintetizador que puedes editar depende de la profundidad de bits elegida para el cartucho del juego. En AJUSTES → EDITOR DE EFECTOS DE SONIDO se configura esa profundidad, si una nota avanza automáticamente (desactivado, cadena de notas o cadena de campos), si la barra espaciadora reinicia la reproducción o alterna entre inicio y parada, y los colores del editor.

16 bits

Cada paso tiene una nota, un volumen, una onda y un efecto. El volumen va de 0 a 7. 0 significa silencio.

Ondas:

- **0 triángulo**
- **1 sierra inclinada**
- **2 vio**
- **3 cuadrados**
- **4 pulsos** : un pulso estrecho
- **5 órganos**
- **6 ruido**
- **7 fáser**
- **8 PWM** : un cuadrado cuyo ancho se mueve
- **9 pulsos 25%**
- **10 seno**
- **11 ruido metálico**
- **Campana de 12 FM**
- **13 arrancar**
- **14 supersierra**

- **15 PWM (C64)**

Efectos en una sola nota:

- **0 ninguno**
- **1 deslizamiento** : el tono y el volumen se deslizan hacia la siguiente nota a lo largo de este paso.
- **2 vibrato** — oscilaciones de tono
- **3 caída** : el tono baja hasta el silencio, el golpe habitual
- **4. Fundido de entrada**
- **5 desvanecimiento**
- **6 arpegio** : recorre esta nota y las tres siguientes, una cada 8 ticks. En TIC 8 o inferior, cada 4 ticks.
- **7 arpegios rápidos** : las mismas cuatro notas, cada 4 ticks, o cada 2 cuando TIC es 8 o menos.

El encabezado del patrón, encima de las notas:

- **TIC** — tics por paso, de 1 a 255. 15 equivale a 120 pulsaciones por minuto. Escriba el número o auméntelo de uno en uno.
- **BUCLE DE ENTRADA/SALIDA** : con SALIDA por encima de 0, el patrón se repite de ENTRADA a SALIDA. Si se deja SALIDA en 0, el título se convierte en LONGITUD: ENTRADA indica cuántas notas se reproducen y 0 significa que se reproduce todo el patrón.
- **RUIDO** : todo el patrón se convierte en ruido.
- **CRUSH** : una variante vibrante de cada ola.
- **DTUNE** — 0 apagado, 1 coro suave, 2 coro fuerte.
- **RVERB** — reverberación: 0 desactivada, 1 corta, 2 larga.
- **DAMPN** — un filtro de paso bajo: 0 apagado, 1 suave, 2 intenso.
- **NIVEL** — volumen de todo el patrón, de 0 a 7.
- **SWING** — arrastrar los pies. Los pasos impares se esperan un poco más, los pares un poco menos.
- **PASOS** : cuántos pasos utiliza el patrón: 32, 24, 16 o 12.

8 bits

Un paso tiene una nota y una onda. Las cuatro ondas son cuadrada, triangular, sierra y ruido. El encabezado tiene TIC, LOOP IN / OUT, FX y LEVEL. TIC es uno de 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60. FX es un efecto para todo el patrón: 0 ninguno, 1 vibrato, 2 arpegio, 3 fundido. LEVEL es de 0 a 3.

4 bits

Un paso es simplemente una nota. El encabezado tiene TIC, LOOP (activado o desactivado), WAVE (las mismas cuatro ondas) y OCT (la octava en la que se encuentran las notas). TIC puede ser 4, 6, 8, 10, 12, 15, 20 o 30.

2 bits

Un paso es un espacio vacío, un tambor, un golpe o un aplastamiento. El encabezado tiene TIC y BASE. TIC puede ser 8, 15, 16 o 30. BASE establece la altura del tambor y del golpe.

1 bit

Un paso es una nota. TIC es 8, 15, 16 o 30, las mismas cuatro velocidades que 2-bit.

Canciones

La pestaña SONIDO muestra la canción abierta. Ocho columnas representan los canales. Cada celda contiene un identificador de patrón. El valor 255, que aparece vacío, indica que el canal está en silencio en esa fila. La columna X contiene un comando: continuar, detener, iniciar bucle o finalizar bucle. Detener finaliza la canción después de la fila. Finalizar bucle regresa al último inicio de bucle.

Una fila termina cuando finaliza el patrón más a la izquierda con LOOP OUT en 0. Los patrones más a la derecha no alargan la fila: si es más larga, se corta; si es más corta, vuelve a empezar desde el primer paso hasta que la fila termina. Un patrón con LOOP OUT por encima de 0 se repite entre LOOP IN y LOOP OUT y no finaliza la fila. Si todos los patrones de la fila se repiten, la fila permanece en esa posición.

Atajos

Llave	Acción
teclas de piano	Escribe una nota sobre el paso seleccionado con el ratón.
Arriba / Abajo	Cambia el valor que se encuentra debajo del cursor.
Espacio	Reproduce el efecto de sonido abierto o la canción abierta de la primera fila. Inicia, detén o reinicia desde la configuración del EDITOR DE EFECTOS DE SONIDO.
Ctrl+Z / Ctrl+Y	Deshacer / rehacer.
Ctrl+C / Ctrl+X / Ctrl+V	Copia, corta y pega la selección.

DATOS

Tablas con nombre. Cada tabla tiene filas y columnas. En el código, `dget`, `dset`, `dstr`, `drows` y `dcols` utilizan estas tablas. La numeración de filas y columnas en el código comienza en 0.

Tipos de columna, ordenados según el encabezado:

- **bool** — un poco.
- **2** — dos bits.
- **4** — cuatro bits.
- **8** — un byte.
- **16** — 16 bits, para identificadores y números más grandes.
- **spr** — un identificador de sprite.

- **str** — texto, dibujado con la fuente del sistema.
- **str8** y **str10** : el mismo texto, dibujado más alto.
- **f16** y **f32** : un número de coma flotante de precisión media y un número de coma flotante completa.

Con el puntero sobre el encabezado de una columna, Ctrl + rueda del ratón cambia el tipo de columna. Una rueda del ratón normal permite desplazarse por las columnas o hacia abajo, según la opción seleccionada en CONFIGURACIÓN → EDITOR DE DATOS. Ctrl invierte esa dirección, por lo que al pulsar Ctrl + rueda del ratón sobre un encabezado se cambia el tipo de columna cuando la rueda está configurada para columnas.

Las teclas de flecha permiten moverse. La tecla Intro edita una celda. AJUSTES → EDITOR DE DATOS:

- **Rueda** : desplaza columnas o filas.
- **Colores** : la cuadrícula, el encabezado y la selección.

Activos

ASSETS importa y exporta la pieza que está abierta, no el proyecto completo. Los sprites usan el sprite abierto en PAINT, en la hoja que está abierta. El mapa es el mapa que está abierto. SFX es el patrón que está abierto. Música es la canción que está abierta. Datos es la tabla que está abierta. El código es la excepción: cada pestaña se escribe y cada pestaña se lee.

Amable	Archivo	Lo que se mueve
Sprites	`.png`	El sprite abierto.
Mapa	`.p37map`	El mapa abierto.
SFX	`.p37sfx`	El patrón abierto.
Música	`.p37mus`	La canción de apertura. Los números de patrón se mantienen sin cambios. Exporta los efectos de sonido por separado si los necesitas.
Datos	`.p37dat`	La mesa abierta.
Código	`.lua`	Cada pestaña.

Ajustes

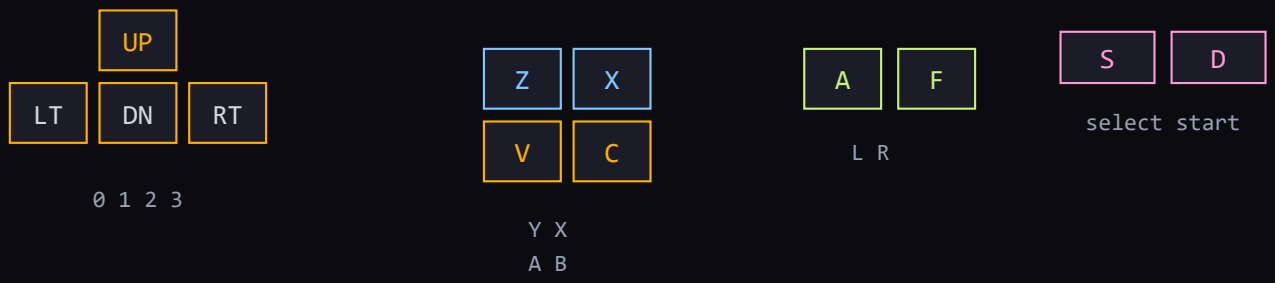
La lista raíz:

Página	Lo que almacena
PERFILES	Guarda y carga todas las configuraciones. Crea, renombra, sobrescribe, elimina. Copia o pega el perfil como texto.

Menú	Colores de ventana, teclas y menú; fondo opcional.
PANEL DE JUEGO	Vista previa y ventana de reproducción.
EDITOR DE CÓDIGO	Resaltar y dar pistas.
EDITOR DE PINTURA	Pincel, paleta, vista de mapa.
EDITOR DE EFECTOS DE SONIDO	Sintetizador y entrada de notas.
EDITOR DE DATOS	Dirección y colores de las ruedas.
IDIOMA	Idioma del menú y de las sugerencias.
LICENCIAS	Textos de licencia, incluyendo la fuente Fusion Pixel y SDL3.

GLOBAL:

- **DISEÑO** — Disposición de los paneles, arriba.
- **VOLUMEN** — nivel maestro, de 0 a 10.
- **ASIGNACIÓN DE TECLAS** : teclas para cada uno de los 8 jugadores. Pulsa una tecla para asignarla. Direcciones, ABXYLR, seleccionar.
- **Presente** : omitir fotogramas inactivos o dibujar todos los fotogramas.
- **Pantalla completa** — sin bordes.
- **Actualizar** : FPS de la ventana. Pasos a izquierda y derecha: 60, 90, 120 y máximo. Introduzca un número de hasta 1000. 0 significa sin límite.
- **Vsync** : espera a que la pantalla se muestre entre fotogramas.
- **Escala** : multiplicador de ventana entera.
- **Shaders** : filtros y CRT en el editor, no en el juego.
- **Color** : texto del menú y resaltado.
- **Fondo** : una imagen PNG opcional, recortada a 128 píxeles, situada detrás del menú. Borrar la imagen la elimina.
- **Cursor del editor** : forma del puntero.
- **Ratón** : puntero flotante o ajustado a píxeles.
- **Velocidad de la interfaz de usuario** : la rapidez con la que se repite una dirección mantenida en una lista.
- **Explorador** de archivos: las listas de archivos comienzan en la raíz del equipo o en la carpeta del archivo ejecutable (.exe).



Teclado predeterminado. A es confirmar en los menús. B es retroceder. En un juego, estos son los botones 4 y 5. Seleccionar (S) es el botón 10, no un acceso directo al menú.

Botón	Role	Clave predeterminada
0	Izquierda	flecha izquierda
1	Bien	Flecha derecha
2	Arriba	flecha hacia arriba
3	Abajo	flecha hacia abajo
4	A, confirmar	V
5	B, atrás	do
6	incógnita	incógnita
7	Y	Z
8	R	F
9	L	A
10	Seleccionar	S
11	Comenzar	D

Construir

Elige una plataforma: Windows, Linux, Mac, Android, iOS o Web.

- **Nombre** : nombre del archivo de compilación.
- **Autor** — en Windows y Linux. Este es el nombre de la carpeta donde se guardan los archivos.
- **ID de la aplicación Steam (** en Windows y Linux). Si está vacío, significa que no hay Steamworks. Un ID completo asigna `achieve(0)`…`achieve(15)` a los logros de Steam ACH_0…ACH_15.

- **Id** : identificador del paquete cuando la plataforma lo solicita.
- **Código oculto** : en superposiciones de escritorio y móviles. Ralentiza el `peek` del cartucho del juego. No es un bloqueo.
- **Pantalla completa** : el reproductor exportado se inicia en pantalla completa.
- **Shaders** : hasta tres capas integradas en el reproductor.
- **Descargar .p37** — Solo en la web. Muestra una entrada de descarga en el menú de pausa.
- **Icono** : una ficha de una hoja de sprites: hoja, tamaño, x, y. O una imagen PNG que puedas recortar.
- **Exportar** : seleccione la carpeta. El menú indica si la compilación se guardó o si la compilación falló.

Crea una compilación con conexión a internet. La pantalla de inicio puede preguntar al servidor, de forma anónima, si se conoce el hash de esta compilación. Esta verificación tiene dos razones. Me protege a mí, el autor de PIX37, de copias de la herramienta que no me pertenecen para distribuirlas. También protege a la persona que creó el juego. Tomar el juego de otra persona y publicarlo como propio se está volviendo común en el mercado de aplicaciones móviles, como Play (Android) o iOS. El juego se inicia igualmente. Si el equipo está sin conexión, la introducción indica que el juego no está conectado. Esta línea también le da al jugador una pista de que la copia podría no provenir de una fuente legal. Es una pequeña verificación, para que si tu juego fue robado y vendido sin tu consentimiento por otra persona, puedas presentar una reclamación en una tienda como Play (Android), Steam o itch.io. El servidor almacena la hora en que se registró el hash por primera vez, junto al hash, y no lo modifica posteriormente. La hora más antigua corresponde a la compilación original. Un hash registrado después de esa hora corresponde a una copia posterior.

Quiero que este control proteja a quienes desarrollan videojuegos. El motor gráfico ya les facilita mucho el trabajo, y esto es una de esas cosas. Entiendo que puede ser controvertido. Si la comunidad decide que es mejor eliminar este micro-DRM, lo haré.

Un cartucho de juego antiguo de un editor anterior puede negarse a arrancar después de los cambios de idioma. Abra el proyecto `.pix37` en este editor y exporte un nuevo `.p37`.

Conéctate a PIX37.PARTY

La opción CONECTARSE A PIX37.PARTY está integrada en la aplicación. No hay página de inicio de sesión en el navegador. Se inicia sesión con una cuenta de Steam o con itch.io, usando el mando y el ratón. El servicio muestra un nombre de usuario único de entre 2 y 16 caracteres. La clave de licencia y el ID de Steam no se muestran a otros jugadores.

Tras aceptar los términos del servicio, el centro principal se compone de Favoritos, Catálogo, Clasificación, Jugadores y tu perfil.

La licencia de desarrollador es para el editor. Creas el juego, lo subes y dejas estrellas. La licencia de jugador es para el jugador. Los corazones provienen de personas que juegan en una partida rápida.

- **Catálogo** : explore los juegos. Los filtros incluyen género (hasta tres), tamaño de pantalla, tamaño del cartucho y si la carga incluye un proyecto `.pix37`.

- **Favoritos** : juegos que marcaste.
- **Subir** : requiere una licencia de desarrollador. Envía un `.p37` y puedes adjuntar el proyecto `.pix37`. Un `.p37` es código de bytes empaquetado. Es difícil de leer y puede ser ofuscado intencionalmente. Un `.pix37` es el archivo del proyecto en el que realmente trabajas: el código fuente, las imágenes y el sonido, en un formato que otra persona puede abrir y del que puede aprender. Enviarlo permite que otros creadores estudien el juego en lugar de un cartucho de juego comprimido. El título, el autor, los géneros, la portada, el icono y los nombres de los logros se incluyen con él. El número de RAM enviado es el pico alcanzado por el cartucho del juego.
- **Comentarios y logros** : en la página del juego. Si desbloqueas un logro mientras tienes la sesión iniciada, se guardará tanto en tu cuenta como en el ordenador.
- **Clasificación** : desarrolladores, ordenados por estrellas o por corazones. Cada fila muestra ambos recuentos. Abre un nombre para ver los juegos de ese autor, ordenados por corazones.
- **Jugadores** : cuentas ordenadas por logros desbloqueados.

Código

El lenguaje es un pequeño script al estilo Lua. Esta sección parte de un proyecto vacío, enseña el lenguaje y luego muestra un breve ejemplo de cada función.

Un primer programa

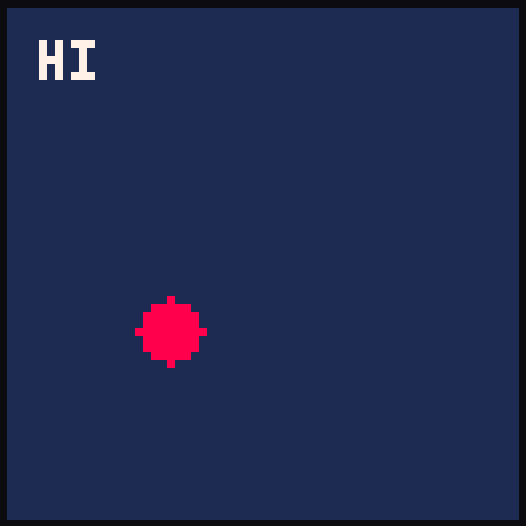
Crea un proyecto de 64x64 píxeles con color de 4 bits. Abre CODE. Reemplaza la pestaña con:

```
_init()
  x = 20
  y = 40
end

_update()
  if btn(0) then x -= 1 end
  if btn(1) then x += 1 end
  if btn(2) then y -= 1 end
  if btn(3) then y += 1 end
end

_draw()
  cls(1)
  print("HI", 4, 4, 7)
  circfill(x, y, 4, 8)
end
```

Ctrl+R lo ejecuta. Las teclas de flecha mueven el círculo. Escape detiene el juego y regresa al editor.



Las mismas tres llamadas: `cls(1)` pinta la pantalla azul marino, `print` dibuja HI y `circfill` dibuja el círculo rosa.

Qué funciona y cuándo.

- `_init` se ejecuta una vez.
- `_update` funciona a 60 Hz. Coloque aquí el movimiento y los botones.
- `_draw` pinta. Sin argumento de FPS, pinta tan rápido como lo permitan la computadora y el monitor. `_draw(30)` limita la pintura a 30 FPS y deja la lógica a 60.
- `_set_fps(n)` cambia ese límite más adelante. 0 lo borra.

Los nombres no distinguen entre mayúsculas y minúsculas; el editor los almacena en minúsculas. `function` y `fun` son la misma palabra clave.

Idioma

Conceptos básicos de programación

Un programa es una lista de líneas que ejecuta el cartucho del juego. Cada encabezado muestra primero la estructura de la línea y luego un pequeño ejemplo. Una línea que comienza con `--` es un comentario: una nota para ti. El juego la omite.

Variables

Una variable es un nombre que almacena un valor. Tú eliges ese nombre. Las líneas posteriores pueden leerlo y almacenar un nuevo valor con el mismo nombre.

La forma es `name = value`. `local name = value` mantiene ese nombre dentro del bloque actual, por lo que el resto del cartucho del juego no puede verlo. Sin `local`, el nombre es global y todas las pestañas pueden verlo.

```
hp = 3            -- global: the whole cart can read and change hp
local coins = 0   -- local: only this block can see coins
hp = hp + 1       -- read 3, add 1, store 4 back into hp
```

`nil` significa que no hay ningún valor. `true` y `false` son sí y no. Un nombre que nunca hayas configurado se lee como `nil`.

```
bonus = nil            -- nothing stored yet
if bonus == nil then   -- condition: is bonus empty?
  bonus = 0            -- body: store a number
end
```

Tablas

Una tabla almacena muchos valores bajo un mismo nombre. Una lista utiliza números. El primer elemento es 1, no 0. Un registro utiliza nombres. Una tabla puede contener otra tabla.

Forma de lista: `{ value, value }`. Forma de registro: `{ name = value, name = value }`. Una lista de registros es una tabla cuyos elementos son tablas.

```
-- a list. [1] is the first item
scores = {10, 20, 30}
print(scores[1], 1, 1, 7)      -- print draws text. this draws 10

-- a record. player.x is the field named x
player = {x = 10, y = 20}
print(player.x, 1, 8, 7)       -- draws 10

-- a list of records. enemies[1] is the first record
enemies = { {x = 4, y = 8}, {x = 20, y = 12} }
print(enemies[1].x, 1, 16, 7)  -- draws 4
print(enemies[2].y, 1, 24, 7)  -- draws 12
```

Estado y cuerpo

Una condición es una pregunta con una respuesta de sí o no, como `hp > 0` o `btnd(4)`. Un cuerpo es el grupo de líneas que se ejecutan al entrar en un bloque. `if` y cada bucle utilizan la misma estructura: una condición, luego un cuerpo, luego `end`. `repeat` finaliza con `until` en lugar de `end`.

Si

`if` ejecuta su cuerpo solo cuando la condición es verdadera. `elseif` intenta otra condición. `else` se ejecuta cuando todas las condiciones son falsas.

Forma: `if condition then`, luego el cuerpo, luego `end`.

```
hp = 2
if hp > 0 then                -- condition: is hp above zero?
  print("ok", 1, 1, 7)        -- body: print draws the word ok
elseif hp == 0 then           -- another condition
  print("empty", 1, 1, 7)
else
  print("no", 1, 1, 7)        -- body when every condition was false
end

if hp > 0 print(hp, 1, 32, 7)  -- one call, no then/end. draws 2
```

Bucles

Un bucle repite un cuerpo. `while` comprueba primero la condición y continúa mientras se mantenga verdadera. `repeat` ejecuta el cuerpo y luego comprueba, por lo que el cuerpo siempre se ejecuta al menos una vez. `for` cuenta. Se incluye el último número. Sin un tercer número, el paso es 1, por lo que si el inicio ya ha pasado el final, el cuerpo no se ejecuta. La cuenta regresiva requiere un paso negativo.

```
n = 0
while n < 3 do         -- condition: keep going while n is under 3
  n += 1
  print(n)             -- 1, then 2, then 3: how many times the body has run
end

n = 0
repeat
  n += 1
  print(n)             -- body first, then the condition. prints 1, then 2, then 3
until n > 2

for i = 1, 3 do        -- i is 1, then 2, then 3
  print(i)
end

for i = 0, 4, 2 do     -- start, end, step of 2
  print(i)             -- 0, then 2, then 4
end

for i = 3, 1, -1 do    -- counting down needs the step
  print(i)             -- 3, then 2, then 1
end

for i = 1, 5 do
  if i == 3 then break end  -- leave the loop when i reaches 3
  print(i)                  -- 1, then 2
end
```

Caminando sobre una mesa

`all` visita cada elemento de una lista, desde el primero. `pairs` visita cada nombre y el valor almacenado bajo ese nombre.

```
t = {1, 2, 3}                -- list: t[1] is 1, t[2] is 2, t[3] is 3
for v in all(t) do           -- v is 1, then 2, then 3
  print(v)                   -- print draws that number
end

player = {x = 10, y = 20}    -- record: a field x and a field y
for k, v in pairs(player) do
  print(k)                   -- k is the name, "x" then "y"
end
```

Funciones

Una función es una receta con nombre. Su estructura es `function name(arguments)`, seguida de un cuerpo y, finalmente, `end`. Se invoca escribiendo el nombre entre paréntesis. `return` devuelve un valor, pudiendo devolver más de uno. Los argumentos son variables locales de dicha función.

```
function greet(who)       -- who is a local that holds what you pass in
  return "hi "..who       -- .. glues two texts. return sends the result back
end

print(greet("ada"))       -- draws hi ada
```

```
function pair()
  return 1, 2             -- two values
end

a, b = pair()             -- a becomes 1, b becomes 2
```

`local function` es una función que puede ver las variables locales que la rodean. Eso es un cierre. Una tabla puede contener uno, y se llama con un punto.

```
function make()
  local n = 0
  local m = {}
  m.bump = function()     -- this function sees n
    n += 1
    return n
  end
  return m
end

counter = make()
print(counter.bump())     -- 1
print(counter.bump())     -- 2
```

Sí, no y elegir

`not` invierte una condición. `and` es verdadero solo cuando ambos lados son verdaderos. `or` es verdadero cuando cualquiera de los lados es verdadero. `a and b or c` elige `b` cuando `a` es verdadero, de lo contrario `c`. `cond ? a : b` es la misma idea en una línea: `a` cuando la condición es verdadera, `b` cuando no lo es.

```
hp = 2
hit = false
night = false

if hp > 0 and not hit then         -- both parts are true, so the body runs
  print("safe", 1, 1, 7)           -- draws safe
end

label = hp > 0 and "ok" or "out"   -- hp is above 0, so label is ok
print(label, 1, 8, 7)

shade = night ? 1 : 7              -- night is false, so shade is 7
print(shade, 1, 16, 7)
```

Matemáticas y texto

`+ - * /` son suma, resta, multiplicación y división. `//` divide y mantiene el número entero. `%` es el resto. `^` es potencia. `..` pega dos textos. `#t` es cuántos elementos hay en una lista. Las comparaciones son `==` (igual), `~=` o `!=` (diferente) y `< > <= >=`. `x += 1` suma 1 a x. La misma forma funciona para `-= *= /= ^= %=`.

```
scores = {10, 20, 30}    -- the list named scores
tiles = 15 // 8          -- 1, the fraction is dropped
left = 15 % 8            -- 7, what is left over
title = "hi ".."ada"     -- the text hi ada
n = #scores              -- how many items are in the list named scores
print(n, 1, 1, 7)        -- 3
print(title, 1, 8, 7)

score = 0
best = 10
if score ~= best then    -- condition: score is not the best
  score += 10            -- add 10 to score
end
print(score, 1, 16, 7)   -- 10
```

Donde vive un lugareño

`local` dura hasta que `end` cierra su bloque. `if` / `end` es un bloque independiente. Se puede especificar una variable local antes de la línea que la define, pero hasta que se ejecute esa línea, el valor será `nil`.

```
if true then
  print(coins)       -- nil: the next line has not run yet
  local coins = 5
  print(coins)       -- 5
end
-- coins does not exist out here
```

Saltar

`::label::` marca una línea. `goto label` salta allí, pero solo dentro de la misma función.

```
x = 0
::again::
x += 1                      -- add 1
if x < 3 then goto again end  -- condition: jump back while x is under 3
```

Memoria, en resumen

Las direcciones `peek` y `poke` visualizan los píxeles en tiempo real mediante una dirección de 16 bits. No almacenan una copia del cartucho del juego. Las direcciones `0x5F00` a `0x5FFF` corresponden a los registros de la cámara, el clip (solo lectura), el color de dibujo, el patrón de relleno, las tres reasignaciones de paleta y los números de página. Los 65280 bytes a ambos lados de ese orificio conforman la ventana de píxeles.

Cuando la pantalla cabe en esos 65280 bytes (64×64 y 128×128 caben, pero 256×256 no), la pantalla comienza en la dirección 0, fila por fila. Lo que queda es la hoja de sprites seleccionada por `0x5F41` (0 es la hoja 0). Si esa hoja es más grande que el espacio restante, `0x5F42` selecciona la página y el mapa no está en la ventana. Si la hoja cabe, la cola son celdas del mapa del mapa seleccionado por `0x5F43`, de uno o dos bytes cada una, y `0x5F44` pagina el mapa cuando no cabe en la cola.

Cuando la pantalla no cabe, toda la ventana es una página de 65280 píxeles. `poke(0x5F40, n)` cambia la página de la pantalla. `poke(0x5F41, n)` con n distinto de 0 muestra la hoja n en su lugar, y la hoja 0 no tiene vista en este modo. `poke(0x5F43, n)` con n distinto de 0 muestra el mapa n, y este tiene prioridad sobre la hoja. El mapa 0 en este modo es solo `mget` y `mset`.

Un código `peek` que no coincide con un píxel o un registro devuelve 0. Un código `poke` en ese caso no produce ningún efecto. Las banderas de sprite, los efectos de sonido y las tablas de datos no se encuentran en esta ventana. Utilice los códigos `fget`, `sfxget` y `dget`.

Uso práctico

Todas las funciones se enumeran al final de esta página, junto con ejemplos prácticos de uso de las funciones de PIX37. Los enlaces debajo de cada ejemplo ofrecen más información sobre las funciones.



Movimiento suave

Un cartucho de juego de 128×128 píxeles, visto desde arriba. El sprite 1 ya está dibujado. Las flechas se mueven con rapidez en cualquier dirección, mientras que la fricción reduce la velocidad. El sprite se despliega: la mitad que queda fuera de la parte inferior se dibuja en la parte superior, y lo mismo ocurre con los demás bordes.

`_init` `_update` `_draw` `btn` `mid` `cls` `spr` `palt`



Comprobando si el siguiente movimiento está bloqueado

Un cartucho de juego de 64x64. El suelo y la pared derecha ya están en el mapa, pintados con el sprite 3, y la bandera 0 está colocada sobre ese sprite. Las flechas mueven el sprite 1. La palabra en la parte superior dice "bloqueado" cuando el siguiente paso se superpondría a una casilla sólida, y "libre" cuando no.

`flr` `fget` `mget` `btn` `cls` `map` `spr` `print` `_update` `_draw`



Un paño con volteo

Pulsa A. El barrido se dibuja con flip(), fila negra a fila, en lugar de dentro de _draw(). _draw no interfiere mientras se ejecuta, por lo que cada fila permanece en pantalla.

`rectfill` `flip` `btnd` `cls` `print` `_update` `_draw`



Tabla de funciones (Clase)

El sprite 1 ya está dibujado. La función make_monster funciona como una clase. Devuelve una tabla, y los métodos update y draw mantienen las coordenadas x e y de ese monstruo. El número en la parte superior indica la distancia que ha recorrido.

`spr` `cls` `print` `flr` `_init` `_update` `_draw`



Cuatro maneras de leer un botón

A es el botón 4. Manténlo pulsado y la carga subirá. Suéltalo y la carga volverá a 0. Repítete arriba y abajo mientras lo mantienes pulsado, para que la fila del menú siga avanzando. btnd es el primer fotograma de una pulsación, que este cartucho de juego utiliza únicamente como inicio de esa pulsación.

`btn` `btnd` `btnp` `btnu` `cls` `print` `_update` `_draw`



Dispara con el ratón

El sprite 1 ya está dibujado en el centro. El punto blanco es el cursor. Al hacer clic izquierdo, se dispara un píxel hacia él. La rueda, impresa en la esquina, cambia la velocidad.

`scroll` `clickl` `mouse` `sqrt` `add` `all` `cls` `spr` `pset` `print` `_update` `_draw`

 Escribe un nombre y confírmalo.

Pulsa A para abrir el campo. Arriba y abajo cambian una letra parpadeante. A la escribe, X la borra, Y es un espacio y B confirma. A continuación, aparece tu nombre.

`btnd` `keyon` `keystr` `flr` `time` `cls` `print` `_update` `_draw`

 Cámara enfocada en el jugador

Un cartucho de juego de 128×128. El sprite 1 comienza en el centro de un suelo de baldosas. Las flechas se mueven. La cámara mantiene el sprite centrado. Los ejes x e y en la esquina no se desplazan, porque la función camera() sin argumentos vuelve a poner la vista en el HUD.

`btn` `screen` `camera` `cls` `map` `spr` `print` `flr` `_update` `_draw`

 Unas pocas líneas de una novela visual

Las líneas están en DATA, en inglés y polaco. A continúa o selecciona permanecer cuando hay una opción en pantalla. B selecciona salir cuando hay una opción y, de lo contrario, cambia el idioma. X cambia el idioma mientras se muestra una opción. El diálogo usa la fuente de 8 píxeles para que se muestren las letras polacas.

`dstr` `dget` `btnd` `cls` `print8` `print` `_update` `_draw`

 Pinta con peek y poke

Se trata de un cartucho de juego de 64x64 píxeles, por lo que la dirección 0 corresponde al píxel superior izquierdo y la siguiente fila comienza en la 64. El cartucho escribe píxeles, copia una fila, borra una fila e imprime los valores que ha leído.

`cls` `poke` `peek` `poke2` `peek2` `poke4` `peek4` `memcpy` `memset` `print` `_draw`

 Una lista que puedes agregar

A suma un número, hasta 8. B elimina el último. X elimina el primero. La lista se imprime en la pantalla.

`btnd` `count` `add` `deli` `cls` `ipairs` `print` `pack` `unpack` `next` `foreach` `select` `pcall` `assert` `del` `_update` `_draw`

 Un nombre desmenuzado

El cartucho del juego divide ada,12,blue e imprime el nombre, su primera letra, 12 y la palabra blue.

`cls` `split` `sub` `chr` `ord` `tonum` `tostr` `print` `_draw`

 Toca una nota y luego cámbiala.

El patrón 0 consta de una sola nota, y cada repetición tiene el mismo volumen. A lo reproduce de nuevo cuando la nota termina. B sube el tono y reproduce esa nota.

`sfxset` `sfxhdr` `sfx` `btnd` `sfxstat` `sfxget` `cls` `print` `_init` `_update` `_draw`

 Una canción escrita en el cartucho del juego.

Dos patrones se repiten en la fila 0, uno en cada uno de los dos primeros canales. La fila permanece en 0. ch0 y ch1 son las notas en las que se encuentra cada canal. Editar el patrón, LEN y LOOP en el cartucho del juego permite ver la función de estos controles en el panel de música.

`songget` `songset` `music` `musicstat` `cls` `print` `_init` `_draw`

 Tres monstruos en DATA

El panel DATOS ya tiene una columna de nombre y una columna de puntos de vida, con limo, murciélago y ojo. A daña al limo. Los números de la derecha son sus puntos de vida.

`drows` `dstr` `dget` `add` `btnd` `dset` `dcols` `cls` `print` `all` `_init` `_update` `_draw`

 Recuerda el marcador

A suma un punto y lo guarda. Cierra el juego y vuelve a abrir el cartucho: el número sigue ahí. B lo olvida.

`load` `btnd` `save` `delsave` `cls` `print` `_init` `_update` `_draw`

  Un segundo capítulo en otro archivo

Dos cartuchos de juego comparten la función de guardar partida. Parte 1: A indica que el juego ha terminado. La Parte 2, que se abre primero, indica que hay que terminar la Parte 1. Después, se abre la Parte 2 y B borra la partida guardada para poder volver a jugar la Parte 1.

`load` `saveslot` `btnd` `save` `delsave` `cls` `print` `_init` `_update` `_draw`

 Un logro

El sprite 8 ya está dibujado. Pulsa A diez veces. Aparecerá un mensaje indicando el logro, y el sprite 8 permanecerá porque está desbloqueado. B borra ese logro del archivo local y reinicia el contador.

`achdef` `btnd` `achieved` `achieve` `achclear` `cls` `print` `spr` `_init` `_update` `_draw`

 Una habitación en la que puedes entrar

Un cartucho de juego de 256×256. Un punto blanco es el cursor. El menú es anfitrión, unirse o la lista de servidores. Anfitrión abre una sala llamada caminar y comienza el juego. Unirse desplaza una dirección (dígitos, un punto, dos puntos) y B se conecta. La lista de servidores se mueve con arriba y abajo, y A se une a la fila. En el juego, las flechas caminan al sprite 1. Se comparte una moneda: el anfitrión la mueve, y al recogerla se suma 1 a la puntuación de ese jugador y aparece el siguiente. Cada sprite, incluido el tuyo, muestra esa puntuación al lado. La esquina muestra cuántos jugadores hay, la dirección LAN y la dirección pública del anfitrión. B se va: tú y los demás regresan al menú, y el siguiente anfitrión inicia una nueva sala con una puntuación de 0.

`btn` `btnd` `mouse` `pset` `spr` `print` `flr` `rnd` `pairs` `nlisten` `nhost` `nannounce` `nlist` `njoin` `nleave` `nsend` `nkb` `nkbstr` `whoami` `nplayers` `nlan` `_mp` `_mp_join` `_mp_leave` `_mp_fail` `palt` `cls` `time` `_init` `_update` `_draw`

 Diálogo que espera

A muestra la siguiente línea. La función se detiene en cada yield hasta que vuelvas a pulsar A.

`yield` `cocreate` `btnd` `coresume` `cls` `print` `_init` `_update` `_draw`

 Espera medio segundo

La palabra cambia de ... a hola después de aproximadamente medio segundo. La pantalla continúa dibujando mientras la corrutina está en pausa.

`yield` `cocreate` `costatus` `coresume` `cls` `print` `_init` `_update` `_draw`

 Círculos que llegan uno a uno

Cinco círculos gotean desde la parte superior, con dos fotogramas de diferencia, y luego caen.

`add` `yield` `cocreate` `coresume` `all` `cls` `circfill` `_init` `_update` `_draw`

 Dos fotogramas, del reloj

El sprite 1 y el sprite 2 son dos fotogramas del mismo personaje. La imagen cambia cuatro veces por segundo.

`cls` `flr` `time` `spr` `_draw`

Cada función

Los argumentos entre corchetes son opcionales. Los índices de tablas y cadenas comienzan en 1, excepto las filas y columnas de datos, que comienzan en 0. Los colores son índices de paleta.

Índice

_dibujar
_inicio
_mp
_mp_fail
_mp_join
_mp_leave
_mp_actualización
_establecer_fps
_actualizar
abdominales
claro
achdef
lograr
logrado
agregar
todos
afirmar
atan2
banda
bno
bor
btn
bnd
btnp
btnu
bxor
cámara
techo
chr
circulo
circunvalar
hacer clic
hacer clic
clicker
clip
cls
cocrear
color
continuar
porque
costato
contar
cursor
dcols
del
delicatessen
delsave
dget
se ahoga
dset
dstr
fget
llenar
voltear
flr
para cada uno
fset
información
pares
tecla apagada
keyon
clave
teclastr
linea
cargar
iniciar sesión
lshr
mapa
máximo
memcpy
conjunto de memorias
conseguir
medio
mín.
ratón
mset
musica
estadística musical
anunciar
llamar
siguiente
nhost
unirse
nkb
nkbstr
nlan
dejar
lista
escuchar
nota
jugadores
enviar
nstún
orden
ovalado
relleno ovalado
paquete
pares
amigo
palrgb
palta
pcall
echar un vistazo
mirar2
mirar4
rendimiento
obtener
empujar
empujar2
empujar4
imprimir
imprimir10
imprimir8
pset
recto
rectificar
rnd
rotl
rotr
correcto

relleno correcto
guardar
ranura de guardado
pantalla
desplazarse
seleccionar
mapa de configuración
paleta de conjuntos
conjuntospr
settex
fijar vuelta
efectos especiales
sfxget
sfxhdr
conjunto de efectos
sfxstat
sget
firmar
shl
mostrarestadísticas
shr
pecado
canción
conjunto de canciones
dividir
primavera
raíz cuadrada
sardina
conjunto
sspr
estadística
sub
tiempo
línea t
tono
tostr
desempacar
whoami
rendimiento

Ciclo vital

_init

`_init() ... end`

Se ejecuta una vez al iniciar el cartucho del juego. El código que se encuentra fuera de cualquier gancho también se incluye en _init.

```
_init()
  x = 64
  y = 40
end
```

_update

`_update() ... end`

Lógica del juego. Se ejecuta a 60 Hz, incluso cuando _draw está limitado o el host es más lento. Lee los botones aquí.

```
_update()
  if btn(0) then x -= 1 end
  if btn(1) then x += 1 end
end
```

_draw

`_draw([fps]) ... end`

Dibuja un fotograma. Sin argumentos, o con 0, se ejecuta con la frecuencia que indique el host. Pase n (1..999) para limitar el dibujo a n fotogramas por segundo. _update se mantiene a 60 Hz.

```
_draw(30)
  cls(1)
  spr(1, x, y)
end
```

_set_fps

`_set_fps(n)`

Cambia el límite de _draw mientras el juego está en ejecución. 0 significa sin límite. n se limita a 0..999.

```
if btn(4) then _set_fps(60) end
```

flip

`flip()`

Presenta la pantalla y cede el paso hasta el siguiente fotograma. Úselo solo si está dibujando su propio bucle de fotogramas. No espere activamente después de usarlo.

```
cls(0)
print("tick", 4, 4, 7)
flip()
```

time

`time()`

Segundos desde _init.

```
print(time(), 4, 4, 7)
```

Aporte

btn

`btn([b], [p])`

1 mientras se mantiene presionado el botón b, de lo contrario 0. p es el jugador, 0..7. Sin argumentos devuelve una máscara de bits del jugador 0. Botones: 0 izquierda, 1 derecha, 2 arriba, 3 abajo, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 seleccionar, 11 iniciar.

```
if btn(4) then shoot() end
```

btnd

`btnd(b, [p])`

Verdadero solo en el marco _update cuando se presiona el botón b.

```
if btnd(5) then menu_open = not menu_open end
```

btnp

`btnp([b], [p])`

Verdadero al presionar, y luego nuevamente mientras se mantiene presionado: 30 ticks de retraso, luego cada 8 ticks. Sin ningún botón, devuelve una máscara.

```
if btnp(2) then cursor -= 1 end
```

btnu

`btnu(b, [p])`

Esto solo es cierto en el momento en que se suelta el botón.

```
if btnu(4) then charge = 0 end
```

mouse

`mouse()`

Devuelve las coordenadas x e y del ratón en píxeles del juego.

```
mx, my = mouse()
pset(mx, my, 7)
```

clickl

`clickl([p])`

Botón izquierdo del ratón. Se mantiene pulsado clickl(). clickl(1) es el borde pulsado.

```
if clickl(1) then place(mouse()) end
```

clickr

`clickr([p])`

Botón derecho del ratón. Mismo borde de 1 argumento que clickl.

```
if clickr() then cancel() end
```

clickm

`clickm([p])`

Botón central o clic de la rueda de desplazamiento.

```
if clickm(1) then drag = true end
```

scroll

`scroll()`

Movimiento de la rueda del ratón en este marco. Los valores positivos y negativos dependen de la dirección de la rueda.

```
cam_y -= scroll()
```

keyon

`keyon()`

Abre un campo de texto controlado por botones. Las flechas arriba y abajo cambian una letra (de la A a la Z). La tecla A escribe la letra. La tecla X borra. La tecla Y es un espacio. La tecla B confirma y cierra el campo. El teclado no escribe en él.

```
if btnd(4) then keyon() end
```

keyoff

`keyoff()`

Oculta el campo de texto.

```
keyoff()
```

keystr

`keystr([s])`

Texto en el campo, luego 1 en el marco B confirma, luego la letra arriba y abajo cambian. Esa letra queda vacía una vez que se cierra el campo. keystr("") borra el texto.

```
name, done, letter = keystr()
if done == 1 then start(name) end
```

keystat

`keystat()`

1 mientras el campo de texto esté abierto, de lo contrario 0.

```
if keystat() == 1 then return end
```

Dibujo

screen

`screen()`

Tamaño actual de la pantalla del cartucho del juego en píxeles. Devuelve el ancho y luego la altura. Esta es la resolución del juego, no la de la ventana del editor.

```
w, h = screen()
```

cls

`cls([col])`

Borra la pantalla. El color predeterminado es 0.

```
cls(1)
```

color

`color(col)`

Establece el color de dibujo que se utilizará cuando una llamada posterior omita `col`. Sin argumentos, devuelve el color actual.

```
color(7)
print("hi")
```

camera

`camera([x, y])`

Desplaza cada llamada de dibujo. La cámara() se restablece a 0, 0.

```
camera(cam_x, cam_y)
```

clip

`clip([x, y, w, h, clip])`

Limita el dibujo a un rectángulo. Un quinto argumento distinto de cero interseca con el recorte actual. clip() restaura la pantalla completa. La vista previa de los registros de recorte es de solo lectura.

```
clip(8, 8, 48, 48)
circfill(32, 32, 20, 8)
clip()
```

cursor

`cursor([x, y, col])`

Establece dónde se imprimirá la siguiente vez sin coordenadas y, opcionalmente, el color. cursor() devuelve x, y.

```
cursor(4, 4, 7)
print("line 1")
print("line 2")
```

pset

`pset(x, y, [col])`

Establece un píxel de pantalla. Cuenta para la estadística de píxeles.

```
pset(x, y, 10)
```

pget

`pget(x, y)`

Color del píxel de la pantalla en x, y.

```
if pget(x, y+8) == 3 then grounded = true end
```

print

`print(v, [x, y, col])`

Dibuja un texto de 3×5 (UTF-8, recortado de la fuente Fusion). Sin las coordenadas x e y, imprime en la posición del cursor y pasa a la siguiente línea.

```
print("score "..score, 2, 2, 7)
```

log

`log(...)`

Agrega los argumentos, separados por espacios, a un búfer de texto. pix37_run imprime ese búfer en la salida estándar. No dibuja.

```
log("hp", hp)
```

print8

`print8(v, [x, y, col])`

Igual que la versión impresa, con la fuente Fusion de 8 píxeles.

```
print8("pause", 16, 28, 7)
```

print10

`print10(v, [x, y, col])`

Igual que la versión impresa, con la fuente Fusion de 10 píxeles.

```
print10("pix37", 8, 8, 10)
```

line

`line([x0, y0, x1, y1, col])`

Línea desde x0,y0 hasta x1,y1. Dos argumentos continúan desde el extremo anterior. line() reinicia esa cadena.

```
line(0, 32, 63, 32, 6)
```

circ

`circ(x, y, r, [col])`

Contorno del círculo. El radio está en píxeles.

```
circ(32, 32, 12, 12)
```

circfill

`circfill(x, y, r, [col])`

Círculo relleno.

```
circfill(x, y, 3, 8)
```

oval

`oval(x0, y0, x1, y1, [col])`

Elipse delimitada dentro del recuadro. Se incluyen las esquinas.

```
oval(8, 20, 56, 44, 7)
```

ovalfill

`ovalfill(x0, y0, x1, y1, [col])`

Elipse rellena dentro de un cuadro delimitador.

```
ovalfill(10, 40, 54, 60, 3)
```

rect

`rect(x0, y0, x1, y1, [col])`

Contorno rectangular. Se incluyen las esquinas.

```
rect(0, 0, 63, 63, 7)
```

rectfill

`rectfill(x0, y0, x1, y1, [col])`

Rectángulo relleno.

```
rectfill(0, 56, 63, 63, 3)
```

rrect

`rrect(x, y, w, h, [r, col])`

Contorno de rectángulo redondeado. x, y son las coordenadas de la esquina superior izquierda. w y h son el tamaño en píxeles; ambos deben ser mayores que 0. r es el radio de la esquina y su valor predeterminado es 0. Se ajusta a la mitad del lado más corto.

```
rrect(8, 8, 48, 16, 3, 7)
```

rrectfill

`rrectfill(x, y, w, h, [r, col])`

Rectángulo redondeado relleno. Los mismos argumentos que rrect.

```
rrectfill(8, 40, 48, 16, 2, 1)
```

fillp

`fillp([p])`

Patrón de relleno de 16 bits para formas rellenas posteriormente. 0 es sólido. fillp() devuelve el patrón actual. Los bits recorren el patrón de 4×4.

```
fillp(0xA5A5)
rectfill(8, 8, 40, 40, 10)
fillp(0)
```

pal

`pal([c0, c1, [mode]])`

Reasigna el color c0 a c1. El modo 0 es la paleta de dibujo, el 1 la paleta de visualización, el 2 la paleta de patrones de relleno. pal() se reinicia.

```
pal(8, 10)
circfill(32, 32, 8)
```

palrgb

`palrgb(col, r, g, b)`

Establece una entrada de paleta en RGB, cada canal de 0 a 255.

```
palrgb(8, 255, 80, 40)
```

palt

`palt([col, t])`

Transparencia. palt(col) o palt(col, true) hace que col sea transparente. palt(col, false) lo hace sólido. palt() hace que todos los colores sean opacos.

```
palt(0)
spr(1, x, y)
```

setpalette

`setpalette(n)`

Selecciona el banco de paleta n. setpalette() devuelve el banco actual.

```
setpalette(1)
```

tline

`tline(x0, y0, x1, y1, mx, my, [mdx, mdy])`

Dibuja una línea en la pantalla mientras toma muestras de píxeles del mapa desde la posición del mapa mx, my. mdx, mdy avanza la muestra del mapa a lo largo de la línea.

```
tline(0, 40, 63, 20, 0, 0, 0.5, 0)
```

Sprites y mapas

spr

`spr(n, x, y, [w, h, fx, fy, tex, rot])`

Dibuja el sprite n en x, y. w y h son el tamaño en píxeles, por defecto 8. fx y fy invierten horizontal y verticalmente cuando no son cero. tex es la pestaña de la hoja de sprites. rot es el ángulo en giros (0..1), en la misma dirección que sin y cos. 0 deja el sprite sin rotar.

```
spr(1, x, y, 8, 8, 0, 0, 0, aim)
```

sspr

`sspr(sx, sy, sw, sh, dx, dy, [dw, dh, fx, fy, tex])`

Copia un rectángulo de la hoja de sprites a la pantalla, escalado a dw por dh. Por defecto, se mantiene el tamaño original.

```
sspr(0, 0, 8, 8, 16, 16, 32, 32)
```

sget

`sget(x, y, [tex])`

Color de un texel en la hoja de sprites.

```
c = sget(0, 0)
```

sset

`sset(x, y, [col, tex])`

Escribe un texel en la hoja de sprites.

```
sset(3, 3, 8)
```

fget

`fget(n, [f, tex])`

Banderas de sprite. fget(n) devuelve el byte de la bandera. fget(n, f) es verdadero cuando el bit de la bandera f (0..7) está activado.

```
if fget(tile, 0) then solid = true end
```

fset

`fset(n, [f], v, [tex])`

Establece indicadores de sprite. fset(n, v) escribe el byte completo. fset(n, f, v) establece un bit.

```
fset(2, 0, true)
```

setspr

`setspr(n)`

Selecciona la hoja de sprites activa. setspr() devuelve la hoja actual. Mismo banco que settex.

```
setspr(1)
```

settex

`settex(n)`

Alias de setspr.

```
settex(0)
```

map

`map(cx, cy, sx, sy, [cw, ch, layer, tex])`

Dibuja el mapa. Las celdas cx y cy del mapa se dibujan en las pantallas sx y sy. Si cw y ch son iguales a 0, se dibuja todo el mapa. Se acepta la capa y se ignora. tex es la hoja del conjunto de mosaicos.

```
map(0, 0, 0, 0, 16, 16)
```

mget

`mget(cx, cy)`

ID del sprite en la celda del mapa cx, cy.

```
if fget(mget(tx, ty), 0) then bounce() end
```

mset

`mset(cx, cy, n)`

Escribe el sprite n en la celda del mapa cx, cy.

```
mset(3, 4, 0)
```

setmap

`setmap(n)`

Selecciona el mapa n. setmap() devuelve el mapa actual. El mapa 0 es un mapa real.

```
setmap(1)
```

Memoria

peek

`peek(addr)`

Lee un byte de la ventana de 64 KB. La ventana es la pantalla activa, luego los píxeles del sprite, luego las celdas del mapa cuando la pantalla cabe en 65280 bytes. Las pantallas más grandes se paginan con poke(0x5F40, page). Las direcciones que no están mapeadas devuelven 0. También se escribe @addr.

```
px = peek(0)
```

poke

`poke(addr, ...)`

Escribe bytes en la dirección especificada. Los argumentos adicionales escriben los siguientes bytes. Las direcciones no asignadas se ignoran. También se escriben $addr y val. Tocar la pantalla cuenta como una escritura de píxeles.

```
poke(0x5F00, 4) -- camera x
```

peek2

`peek2(addr)`

Valor sin signo de 16 bits, little-endian, en la dirección.

```
w = peek2(0x6000)
```

poke2

`poke2(addr, val)`

Escribe un valor sin signo de 16 bits, en formato little-endian.

```
poke2(0x6000, 1000)
```

peek4

`peek4(addr)`

Valor sin signo de 32 bits, little-endian.

```
n = peek4(0x6000)
```

poke4

`poke4(addr, val)`

Escribe un valor sin signo de 32 bits, en formato little-endian.

```
poke4(0x6000, 1)
```

memcpy

`memcpy(dst, src, len)`

Copia len bytes dentro de la misma ventana de 64 KB.

```
memcpy(0x6000, 0x6100, 64)
```

memset

`memset(dest, val, len)`

Rellena len bytes comenzando en dest con val.

```
memset(0, 0, 64)
```

Matemáticas

abs

`abs(x)`

Valor absoluto.

```
d = abs(tx - x)
```

flr

`flr(x)`

Redondea hacia menos infinito.

```
tx = flr(x / 8)
```

ceil

`ceil(x)`

Redondeos hacia el infinito positivo.

```
n = ceil(hp / 8)
```

max

`max(a, b)`

El mayor de los dos valores.

```
x = max(0, x)
```

min

`min(a, b)`

El menor de los dos valores.

```
x = min(63, x)
```

mid

`mid(a, b, c)`

El valor intermedio de los tres. Útil como limitador.

```
x = mid(0, x, 63)
```

sgn

`sgn(x)`

Signo de x: -1, 0 o 1.

```
x += sgn(dir)
```

sqrt

`sqrt(x)`

Raíz cuadrada. Un argumento negativo no es un número.

```
dist = sqrt(dx*dx + dy*dy)
```

sin

`sin(x)`

Seno. x se mide en intervalos de: 0, 0.25, 0.5, 0.75 dan una vuelta completa al círculo. No en radianes.

```
y = 32 + sin(time()) * 8
```

cos

`cos(x)`

Coseno, a su vez, misma convención que el pecado.

```
x = 32 + cos(time()) * 8
```

atan2

`atan2(dx, dy)`

Ángulo del vector dx, dy, en intervalos de 0 a 1.

```
aim = atan2(mx - x, my - y)
```

rnd

`rnd([x])`

Número aleatorio en [0, x). El valor predeterminado de x es 1, por lo que rnd() es 0..1. Para un dado, flr(rnd(6))+1.

```
c = flr(rnd(16))
```

srand

`srand(seed)`

Inicializa el generador aleatorio para que una secuencia rnd posterior pueda repetirse.

```
srand(1)
```

band

`band(a, b)`

AND bit a bit. También se escribe a & b.

```
if band(flags, 1) ~= 0 then hit() end
```

bor

`bor(a, b)`

OR bit a bit. También se escribe a | b.

```
flags = bor(flags, 4)
```

bxor

`bxor(a, b)`

XOR bit a bit. También se escribe a ^^ b.

```
n = bxor(n, 255)
```

bnot

`bnot(a)`

NOT bit a bit. También se escribe ~a.

```
mask = bnot(1)
```

shl

`shl(a, n)`

Desplazamiento a la izquierda. También se escribe << n.

shr

`shr(a, n)`

Desplazamiento aritmético a la derecha. También se escribe >> n.

```
n = shr(n, 1)
```

lshr

`lshr(a, n)`

Desplazamiento lógico a la derecha. También se escribe >>> n.

```
n = lshr(n, 4)
```

rotl

`rotl(a, n)`

Gire los bits hacia la izquierda. También se escribe <<> n.

```
n = rotl(n, 1)
```

rotr

`rotr(a, n)`

Gira los bits a la derecha. También se escribe >>< n.

Tablas y valores

add

`add(t, v, [i])`

Agrega v a la tabla t. Si se proporciona i, inserta en ese índice basado en 1.

```
add(bullets, {x=x, y=y})
```

del

`del(t, v)`

Elimina la primera entrada igual a v.

```
del(items, gem)
```

deli

`deli(t, [i])`

Elimina el índice i y devuelve ese valor. Si no se especifica i, elimina la última entrada.

```
local b = deli(bullets, i)
```

all

`all(t)`

Iterador sobre la parte del array, en orden. Los huecos detienen el recorrido de la misma manera que termina un array de base 1.

```
for b in all(bullets) do
  b.x += 2
end
```

foreach

`foreach(t, fn)`

Llama a fn(v) para cada elemento de la matriz.

```
foreach(enemies, draw_enemy)
```

pairs

`pairs(t)`

Iterar sobre cada clave y valor, incluidos los campos con nombre.

```
for k, v in pairs(player) do
  print(k)
end
```

ipairs

`ipairs(t)`

Iterador sobre la parte del array con índice basado en 1.

```
for i, v in ipairs(deck) do
  spr(v, i*8, 8)
end
```

next

`next(t, [k])`

Devuelve la siguiente clave y valor después de k. next(t) comienza desde el principio. Utilizado por pares.

```
k, v = next(t)
```

count

`count(t, [v])`

Longitud de la parte del array. Si se proporciona v, cuenta cuántas entradas son iguales a v.

```
if count(gems) == 0 then win() end
```

pack

`pack(...)`

Agrupa los argumentos en una nueva tabla.

```
t = pack(1, 2, 3)
```

unpack

`unpack(t, [i, j])`

Devuelve t[i] a t[j] como valores separados. Los valores predeterminados cubren toda la parte del array.

```
a, b = unpack(t)
```

select

`select(i, ...)`

select("#", ...) devuelve cuántos argumentos siguen. select(i, ...) devuelve los argumentos a partir de i.

```
n = select("#", 1, 2, 3)
```

assert

`assert(c, [msg])`

Se detiene con un error cuando c es falso o nulo. msg reemplaza el texto predeterminado.

```
assert(hp > 0, "dead")
```

pcall

`pcall(f, ...)`

Llama a f. Si tiene éxito, devuelve verdadero y los resultados. Si hay un error, devuelve falso y el texto del error.

```
ok, err = pcall(load_level, 3)
```

Instrumentos de cuerda

chr

`chr(n)`

Un carácter de un código de bytes 0..255.

```
s = chr(65) -- A
```

ord

`ord(s, [i, j])`

Códigos de bytes de s desde el índice i hasta el j. Los índices comienzan en 1. Por defecto, se utiliza el primer carácter.

```
code = ord("a")
```

sub

`sub(s, i, [j])`

Subcadena desde i hasta j, con base 1. Omita j para tomar la cola.

```
name = sub(keystr(), 1, 8)
```

split

`split(s, [sep, conv])`

Divide s en una tabla. sep por defecto es coma. Un número sep es el ancho del fragmento. conv convierte piezas con tonum cuando es verdadero.

```
words = split("red,blue", ",")
```

tonum

`tonum(s, [base])`

Analiza un número. La base puede ser de 2 a 36. Devuelve nil cuando el texto no es un número.

```
n = tonum("255")
```

tostr

`tostr(v, [digits])`

Convierte un valor en texto. El parámetro digits establece los decimales para los números.

```
print(tostr(hp, 0), 4, 4, 7)
```

Audio

sfx

`sfx(n, [ch, offset, len])`

Reproduce el patrón n y devuelve el canal. Si se omite ch o se pasa -1, se elige el primer canal libre. ch -2 detiene el patrón n en todos los canales. n -1 detiene un canal, o todos los canales si se omite ch. n -2 permite que un patrón en bucle finalice en lugar de continuar el bucle. sfx() sin argumentos devuelve -1.

```
ch = sfx(0)
```

sfxget

`sfxget(n, step)`

Devuelve una tabla con un paso del patrón: [1] tono, [2] onda, [3] volumen, [4] efecto. El tono es de 0 a 63 (de Do-0 a Re#5). La onda es de 0 a 15. El volumen es de 0 a 7, y 0 significa silencio. El efecto es de 0 a 7.

```
step = sfxget(0, 0)
pitch = step[1]
```

sfxset

`sfxset(n, step, pitch, wave, vol, fx)`

Escribe un paso del patrón. Se requieren los seis argumentos.

```
sfxset(0, 0, 24, 3, 5, 0)
```

sfxhdr

`sfxhdr(n, [key, val])`

Encabezado de patrón. sfxhdr(n) devuelve la tabla de encabezado. sfxhdr(n, key, val) establece un campo: velocidad, loop_in, loop_out, nivel, ruido, crush, dtune, rverb, dampn, swing, pasos. Los números 0..10 seleccionan los mismos campos en ese orden. La velocidad es TIC, ticks por paso. Cuanto mayor sea, más lento será. 15 es 120 pulsaciones por minuto. 16-bit acepta 1..255. Los sintetizadores más estrechos se ajustan a su lista TIC.

```
sfxhdr(0, "speed", 15)
```

sfxstat

```
sfxstat([ch])
```

El ID del patrón se reproduce en el canal ch, o -1. El canal predeterminado es 0.

```
if sfxstat(0) < 0 then sfx(1) end
```

note

```
note(pitch, [wave, vol, fx, ch, ticks])
```

Reproduce una nota sin patrón. Tono 0..63, onda 0..15, volumen 0..7, efecto 0..7. Los ticks indican cuánto tiempo se mantiene, con los mismos ticks de 120 Hz que TIC. Si se omiten los ticks, dura 15, un paso predeterminado. El efecto se ejecuta durante ese tiempo y luego el canal se silencia.

```
note(36, 3, 5, 5, 0)
```

music

```
music([n, fade, mask])
```

Comienza la canción n desde el principio. music(-1) la detiene. Llamar a music con el ID de la canción que ya se está reproduciendo la deja en ejecución. music() devuelve el ID de la canción que se está reproduciendo, o -1. fade se mide en milisegundos: la canción aparece gradualmente, o desaparece gradualmente cuando n es -1. Los bits de máscara que están activados son los canales utilizados para la canción. 0 significa los ocho. Los demás canales permanecen libres para efectos de sonido.

```
music(0, 1000, 7)
```

musicstat

`musicstat()`

Tabla de la canción que se está reproduciendo. song y row son el ID de la canción y la fila, o -1 cuando no se está reproduciendo nada. step y pos tienen ocho números. step[1] es el canal 0, step[8] es el canal 7. step es el índice de la nota, o -1 cuando ese canal está en silencio. pos es qué tan lejos ha llegado esa nota, de 0 a 1. tick es cuántos ticks se ha reproducido esta fila. played es cuántas filas han comenzado desde music(), incluyendo la que se está reproduciendo.

```
st = musicstat()
row = st.row
note = st.step[1]
```

songget

`songget(song, [row, ch])`

El ID del patrón se almacena en una celda de canción. ch es de 0 a 7. 255 significa que la celda está vacía. Esta es la llamada en vivo; el texto de sugerencia del código aún muestra un orden de argumentos anterior.

```
id = songget(0, 0, 0)
```

songset

`songset(song, row, ch, pattern)`

Escribe un identificador de patrón en una celda de canción. Usa 255 para borrar la celda.

```
songset(0, 0, 0, 1)
```

stat

`stat(n)`

Solo audio. 16..23 es el ID de efectos de sonido en el canal n-16. 24..31 es el patrón musical en el canal n-24. Cualquier otro n devuelve 0. La temporización de fotogramas y la RAM son de rendimiento e información, no estadísticas.

```
id = stat(16)
```

Tablas de datos

dget

`dget(t, row, col)`

Lee una celda de la tabla de datos t desde el panel DATOS. La fila y la columna comienzan en 0.

```
hp = dget(0, 0, 1)
```

dset

`dset(t, row, col, v)`

Escribe en una celda. Utilice una cadena de texto para la columna.

```
dset(0, 0, 0, "slime")
```

dstr

`dstr(t, row, col)`

La celda como texto. Los números se convierten.

```
name = dstr(0, row, 0)
```

drows

`drows(t)`

¿Cuántas filas tiene la tabla t?

```
for r = 0, drows(0) - 1 do
end
```

dcols

`dcols(t)`

¿Cuántas columnas tiene la tabla t?

```
cols = dcols(0)
```

Guardar y logros

save

`save(k, v)`

Almacena v bajo la clave k en el archivo de guardado. v puede ser un número, una cadena de caracteres o una tabla de ambos. El archivo está comprimido.

```
save("hp", hp)
```

load

`load(k)`

Lee la clave k. Devuelve nil si la clave no está presente.

```
hp = load("hp") or 3
```

delsave

`delsave(k)`

Elimina la tecla k del archivo guardado.

```
delsave("hp")
```

saveslot

`saveslot([name])`

Selecciona el archivo de guardado dentro de la carpeta del autor. Si no tiene nombre, la ranura será el título del juego. Devuelve 1 si el archivo ya existe.

```
if saveslot("slot1") == 1 then
  hp = load("hp") or 3
end
```

achieve

`achieve(i)`

Desbloquea el logro i, del 0 al 15. Se almacena localmente. Una compilación de Steam también establece ACH_i. Una sesión de catálogo lo reporta a la cuenta del jugador.

```
if coins >= 100 then achieve(0) end
```

achieved

`achieved(i)`

1 cuando el logro i ya está desbloqueado.

```
if achieved(0) == 1 then spr(8, 4, 4) end
```

achdef

`achdef(i, text, [spr, tex])`

Texto emergente para el logro i, que se muestra durante 15 segundos desde la parte inferior de la pantalla. spr es una ficha opcional de 8×8, tex su hoja.

```
achdef(0, "rich", 8)
```

achclear

`achclear(i)`

Borra el logro i en el archivo local. Un ACH_i de Steam que ya estaba configurado permanece configurado.

```
if btnd(5) then achclear(0) end
```

Red

whoami

`whoami()`

Tu ID de jugador, 0..15, mientras la sesión esté activa. Sin conexión, es -1.

```
id = whoami()
```

nhost

`nhost()`

1 si esta máquina es la anfitriona, de lo contrario 0.

```
if nhost() == 1 then spawn() end
```

nplayers

`nplayers()`

¿Cuántos IDs activos existen en el rango 0..15?

```
print(nplayers(), 4, 4, 7)
```

nlisten

`nlisten()`

Comienza a alojar UDP y devuelve la dirección LAN como ip:puerto.

```
addr = nlisten()
```

njoin

`njoin(addr)`

Se une mediante IP o IP:puerto.

```
njoin("10.0.0.8:7777")
```

nleave

`nleave()`

Abandona la sesión. Los pares ven _mp_leave con why 0.

```
if btnd(11) then nleave() end
```

nsend

`nsend([p,] name, ...)`

Envía un mensaje con pérdida de datos. El nombre es el evento. Le siguen hasta cuatro valores adicionales. p de 0 a 15 susurros a ese par; omita p para enviar a todos.

```
nsend("pos", x, y)
```

ncall

`ncall([p,] name, ...)`

Tiene la misma estructura que nsend, pero el mensaje se reintenta hasta que se recibe una confirmación. Tres fallos consecutivos finalizan la sesión.

```
ncall("hit", target, dmg)
```

nlan

`nlan()`

IP:puerto LAN actual después de nlisten.

```
print(nlan(), 2, 2, 7)
```

nstun

`nstun()`

Tu IP pública:puerto de STUN. Un invitado en otra red puede intentar conectarse.

```
pub = nstun()
```

setturn

`setturn(addr, [user, pass])`

TURN es un mecanismo de reserva que se activa cuando falla una ruta directa. Si se utiliza setturn() con argumentos vacíos, se desactiva. La dirección es host:puerto.

```
setturn("turn.example:3478", user, pass)
```

nlist

`nlist(game)`

Tabla de sesiones abiertas para ese nombre de juego. Cada entrada tiene la dirección, los jugadores y el nombre.

```
rooms = nlist("duel")
```

nannounce

`nannounce(game)`

Latido del corazón para que este anfitrión aparezca en la lista de correo para el juego. Llámalo con un temporizador mientras sea anfitrión.

```
nannounce("duel")
```

nkb

`nkb([on])`

Campo de dirección. nkb(1) lo abre, nkb(0) lo oculta, nkb() indica si está abierto. Las teclas arriba y abajo cambian un carácter (del 0 al 9), un punto o dos puntos. A escribe en él. X borra. El teclado no escribe en él.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Texto en el campo de dirección, luego los caracteres cambian hacia arriba y hacia abajo. nkbstr("") borra el texto.

```
addr, digit = nkbstr()
```

_mp

`_mp(from, name, ...)`

Escribe esta función. Se ejecuta cuando llega un mensaje. `from` es el ID del remitente. `name` es el evento. A continuación, se especifican hasta cuatro valores.

```
function _mp(from, name, a, b)
  if name == "pos" then
    px[from] = a
    py[from] = b
  end
end
```

_mp_join

`_mp_join(p)`

Escribes esta función. Un par recibió el ID p. También se ejecuta para ti; whoami() es ese ID.

```
function _mp_join(p)
  add(log, p)
end
```

_mp_leave

`_mp_leave(p, why)`

Escribe esta función. El par p se fue. Por qué 0 es una salida normal. Por qué 1 es una conexión perdida.

```
function _mp_leave(p, why)
  px[p] = nil
end
```

_mp_update

`_mp_update()`

Escribe esta función. Se ejecuta cada 15 fotogramas _update, aproximadamente a 4 Hz. Un buen lugar para enviar posiciones.

```
function _mp_update()
  nsend("pos", x, y)
end
```

_mp_fail

`_mp_fail()`

Usted escribe esta función. La sesión ha terminado: el host desapareció o ncall falló tres veces.

```
function _mp_fail()
  online = false
end
```

Corrutinas

cocreate

`cocreate(f)`

Crea una corrutina a partir de la función f. No se ejecuta hasta que se reanuda la ejecución.

```
co = cocreate(cutscene)
```

coresume

`coresume(c, ...)`

Ejecuta la corrutina c hasta que cede o devuelve un valor. En caso de éxito: verdadero y los valores cedidos. En caso de error: falso y el error.

```
ok, a = coresume(co)
```

costatus

`costatus(c)`

"suspendido", "en marcha" o "muerto".

```
if costatus(co) == "dead" then co = nil end
```

yield

`yield(...)`

Devuelve el control a quien llamó a coresume. Los valores se convierten en el resultado de esa coresume.

```
yield(1)
```

Medida

info

`info([x, y, col])`

Dibuja el HUD en vivo en _draw: fps (promedio de 1 segundo), upd, drw, px, max, ram, peak, ramc y net cuando hay una sesión activa. info() usa el cursor de impresión. info(col) solo establece el color. info(x, y) o info(x, y, col) lo coloca. Los números usan B, k o m para que se ajusten a una pantalla pequeña. La llamada en sí no se cuenta en px.

```
info(1, 1, 7)
```

showstats

`showstats([on])`

Mismo HUD que info, dibujado automáticamente en 0, 0 después de _draw, ignorando la cámara y el clip. showstats(0) lo oculta. showstats(1) o showstats() lo muestra.

```
showstats(1)
```

perf

`perf(n)`

Un número HUD, para que puedas dibujarlo tú mismo. 0 fps, 1 upd, 2 drw, 3 px, 4 max (fotograma más pesado en este segundo), 5 ram, 6 ramc (ram más el cartucho del juego empaquetado), 7 pico de ram desde que se inició el cartucho del juego.

```
print(perf(0), 2, 2, 7)
```