Manuale PIX37

Editor, poi codice. Copia offline del libro su pix37.party/docs.

Redattore

PIX37 DEV è l'editor. Qui puoi disegnare, scrivere e ascoltare il gioco, quindi esportare una cartuccia di gioco o una versione standalone. Può anche riprodurre una cartuccia di gioco e aprire PIX37.PARTY per scaricare cartucce di gioco e utilizzare il catalogo. PIX37 DEV EDUCATIONAL è lo stesso editor, ma è gratuito. Non supporta la creazione di build, l'esportazione di cartucce di gioco e non consente l'accesso a PIX37.PARTY. PIX37 Console è il lettore. Riproduce le cartucce di gioco `.p37` e permette di aprire PIX37.PARTY per scaricare cartucce di gioco e sfogliare il catalogo.

Progetti e cartucce di gioco

Un file `.pix37` rappresenta il progetto: schede sorgente, fogli di sprite, mappe, suoni e tabelle dati. Un file `.p37` rappresenta la cartuccia di gioco eseguita dal giocatore: bytecode compresso in modo compatto.

I salvataggi e gli obiettivi si trovano nella cartella dell'autore, accanto ai dati dell'applicazione. Su Windows, il percorso è `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` per i salvataggi e `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` per gli obiettivi. Su Linux, i file corrispondenti sono `~/.local/share/PIX37/Mira/Pong.p37s` e `~/.local/share/PIX37/Mira/Pong.ach`. Ada è l'account sul computer. Mira è il nome dell'autore, che corrisponde al nome della cartella. Pong è lo slot di salvataggio, che di default corrisponde al titolo del gioco. `save` scrive il file `.p37s`. Il primo obiettivo sbloccato scrive il file `.ach`. Se i file sono già presenti, il programma di avvio li legge all'avvio della cartuccia di gioco.

Menu sviluppatore

Premendo Esc si apre il menu sviluppatore. Premendo Esc durante l'esecuzione di un gioco si torna all'editor. Confermare con Invio o con il pulsante A (V di default). Il pulsante Indietro è il pulsante B (C di default).

Senza alcun progetto caricato, il menu è: CREA PROGETTO, CARICA PROGETTO, COMPILA, CARICA cartuccia di gioco, CONNETTITI A PIX37.PARTY, IMPOSTAZIONI.

Con un progetto caricato, le opzioni sono: SALVA PROGETTO, CARICA PROGETTO, CREA PROGETTO, RISORSE, COMPILA, CARICA cartuccia di gioco, CONNETTI A PIX37.PARTY, IMPOSTAZIONI.

Ctrl+S salva il progetto aperto. Ctrl+R lo esegue.

Crea un progetto

La funzione CREA PROGETTO richiede la cartuccia di gioco prima di poter iniziare a creare il progetto.

- **VIDEO** — 1 BIT (2 colori), 2 BIT (4), 4 BIT (16), 8 BIT (256).
- **AUDIO** — profondità di campionamento del sintetizzatore: 16, 8, 4, 2 o 1 bit.
- **SCHERMO** — le impostazioni predefinite includono 64×64, 128×128, 160×90, 256×256 e 320×180. La modalità DIMENSIONI PERSONALIZZATE consente di scegliere qualsiasi dimensione da 1×1 a 512×512.
- **TAVOLOZZE** — gruppi di colori. Il primo gruppo inizia con il set di 16 colori mostrato di seguito. CARICA TAVOLOZZE applica un set predefinito: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 e un elenco più lungo tratto da Lospec (Resurrect 64, Apollo, Endesga 32 e 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century e altri). È anche possibile caricare un file di tavolozza o caricare un'immagine PNG e estrarre la tavolozza da essa.



Indici di colore utilizzati da `cls`, `pset`, `spr` e gli altri. È possibile ridisegnare qualsiasi indice nell'editor della tavolozza.

I cinque pannelli

L'editor è composto da GAME, PAINT, CODE, SOUND e DATA, in quest'ordine. La mappa si trova all'interno di PAINT. Gli effetti sonori e le canzoni si trovano all'interno di SOUND. Clicca su un pannello per lavorarci. Quando CODE è attivo, SOUND e DATA ignorano i tasti alfanumerici, quindi la digitazione non riproduce anche il pianoforte.



Layout completo, scritto `1:1:4:1:1`. Ogni numero indica la porzione di riga occupata dal pannello: GAME, PAINT, CODE, SOUND, DATA. Uno schermo piccolo è centrato in una cella di 128×128 pixel. Uno schermo più grande di 128 pixel allarga la colonna GAME.

IMPOSTAZIONI → GLOBALE → LAYOUT ciclicamente le disposizioni:

Codice	Ciò che vedi
`1:1:4:1:1`	Tutto. GIOCO e SUONO sovrapposti, COLORE e DATI sovrapposti, CODICE a destra.
`1:1:4:1:1 2X`	Stessa disposizione. Il CODICE viene disegnato con una densità doppia, quindi ci sta il doppio del testo.
`1:0:1:0:1`	GIOCO, DATI, CODICE.
`1:0:1:0:1 2X`	La stessa configurazione, con il CODICE più denso.
`0:1:1:1:1`	Nessuna anteprima di gioco. PITTURA, CODICE, SUONO, DATI.
`0:1:1:1:1 2X`	La stessa configurazione, con il CODICE più denso.
`0:1:4:1:2`	Studio. CODICE largo, una striscia DATI più alta, PITTURA e SUONO.
`0:1:4:1:2 2X`	La stessa configurazione, con il CODICE più denso.

`0:0:4:0:4`	Solo CODICE e DATI, entrambi di grandi dimensioni.
`0:0:4:0:4 2X`	La stessa configurazione, con il CODICE più denso.
`0:1:4:1:0`	VERNICE, SUONO e un CODICE ampio.
`0:1:4:1:0 2X`	La stessa configurazione, con il CODICE più denso.
`0:4:0:0:0`	SOLO PITTURA, 256×256, un pixel del foglio per un pixel della tela.

Una riga `2X` mantiene gli stessi pannelli. Il pannello CODE viene disegnato con una risoluzione doppia e ridimensionato nella stessa cella, in modo che il testo, le tabulazioni e la spaziatura siano alti la metà e il doppio del codice sorgente possa essere visualizzato sullo schermo.

GIOCO

Anteprima della cartuccia di gioco in esecuzione. Fai clic sul pannello GIOCO oppure premi Ctrl+R per avviare l'anteprima. IMPOSTAZIONI → PANNELLO GIOCO è il punto in cui è possibile configurare il pannello:

- **Anteprima** : colore fisso, continuo o ricaricamento automatico durante la modifica.
- **Avvia il gioco in una nuova finestra** : il gioco si aprirà accanto all'editor.
- **Visualizzazione** : anteprima in formato RGB888 o RGB565.
- **GAME 64** — una schermata 64×64 è centrata 1:1 nella cella oppure scalata 2× per riempirla.
- **Shader** : filtri e CRT applicati al gioco, separati dai filtri presenti nell'editor.
- **Colore** : sfondo del pannello.

COLORE

Fogli di sprite e, sul lato della mappa dello stesso pannello, mappe a tessere. I fogli possono essere più grandi dello schermo. Le dimensioni dell'anteprima degli sprite sono 8, 16, 32, 64 e 128 (128 solo se il foglio è di almeno 128 px). Passa la rotellina del mouse sull'anteprima dello sprite per ingrandirla. Le dimensioni del pennello sono 1, 2, 3 e 4.

Chiave	Attrezzo
B	Matita
M	Selezione rettangolare
Q	Lazo
S	Timbro

G	Secchio
IO	Contagocce
V	Mossa
L / Maiusc+L	Linea/curva
U / Maiusc+U	rettangolo cavo / ellisse cava
F / Maiusc+F	Rettangolo pieno / ellisse piena
D	Esitare
[ ]	Pennello più piccolo/più grande
Ctrl+Z / Ctrl+Y	Annulla/Ripristina. Anche Ctrl+Maiusc+Z ripristina.
Ctrl+C / Ctrl+X / Ctrl+V	Copia, taglia, incolla
Ctrl+A	Seleziona la tela

La mappa utilizza la stessa idea, con meno strumenti: B matita, M selezione rettangolare, S timbro, V sposta e Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Tieni premuto Alt per ingrandire la texture della mappa. La dimensione del pennello sulla mappa rimane invariata rispetto agli strumenti di pittura.

Ogni sprite ha otto bit flag. Paint li imposta sullo sprite; `fget` e `fset` li leggono nel gioco.

IMPOSTAZIONI → EDITOR DI PITTURA è il punto in cui viene configurato quel pannello:

- **Tavolozza** : quale banco di carte viene utilizzato, più carica e salva.
- **Anteprima** : RGB888 o RGB565.
- **Cursore a pennello** : il puntatore che usi mentre dipingi.
- **Pixel snap** : come il pennello si aggancia ai pixel.
- **Grande pennello** : un grande pennello si spezza dalla sua parte superiore sinistra o da tutta la sua base.
- **Buccia di cipolla** : un fantasma di altri fotogrammi e la direzione in cui quel fantasma è rivolto.
- **Zoom della mappa** : pixel reali o zoom libero.
- **Colori** : i colori dell'interfaccia utente di Paint.

CODICE

Le schede sono file separati all'interno del progetto. Fai clic sul nome di una scheda per rinominarla; premi Invio per confermare. Diverse schede, ognuna delle quali definisce `_init`, `_update` o `_draw`, vengono unite in ordine di tabulazione: la prima scheda viene eseguita per prima, l'ultima per ultima. Il codice che non si trova

all'interno di un hook diventa parte di `_init`. Una cartuccia di gioco senza hook viene trattata come un singolo `_draw`.

Chiave	Azione
Frecce	Sposta il cursore. Il tasto Maiusc estende la selezione.
Ctrl+Sinistra/Destra	A parole.
Home / Fine, Ctrl+Home / Fine	Inizio o fine della riga, oppure della tabulazione.
Tab	Inserisci due spazi. Nell'elenco di completamento, accetta il nome.
Inserisci	Nuova riga, rientrata come la precedente. Accetta un completamento.
Ctrl+C / X / V	Copia, taglia, incolla.
Ctrl+Z / Ctrl+Y	Annulla / Ripeti.
Ctrl+A	Seleziona la scheda.
Ctrl+D	Duplica la riga o la selezione verso il basso.
Ctrl+/	Commenta o decommenta con `--`.
Ctrl+F	Trova. F3 passa al risultato successivo. Esc chiude la barra.
Ctrl+G	Vai a un numero di riga.
Fuga	Chiude prima la ricerca, la navigazione o l'elenco dei suggerimenti. Premendo nuovamente Esc si esce dall'editor e si accede al menu sviluppatore.

Vengono visualizzati suggerimenti mentre digiti il nome di una funzione. Premi Invio o Tab per inserire il nome e un `(` se COMPLETE (è attivo nelle impostazioni. IMPOSTAZIONI → EDITOR DI CODICE:

- **Ricerca** : premi Ctrl+F nella scheda attiva oppure in tutte le schede.
- **Testo alto** : ritaglia i glifi a 5 px oppure mostra il glifo per intero.
- **Dimensione della stringa** : dimensione del carattere delle stringhe letterali.
- **Carattere per i commenti** : 3×5, oppure il carattere Fusion della lingua dell'interfaccia utente corrente.
- **Completa (** — se il completamento digita tra parentesi.
- **Suggerimenti per il codice** : attivabili o disattivabili.
- **Lingua dei suggerimenti** : la lingua del menu, oppure l'inglese.
- **Spazio del cursore** : offset aggiuntivo sulle righe vuote.
- **Margine di separazione** — numeri di riga.
- **Colore** — sfondo, commenti, tabulazioni, numeri, operatori, parole chiave, stringhe, funzioni del motore, le tue funzioni, variabili, parentesi, accento circonflesso, selezione, riga di errore, avviso.

SUONO

Il pannello ha due schede. SFX rappresenta un pattern: una breve sequenza di note. SOUND rappresenta un brano: posiziona i pattern su otto canali. TIC indica quanti tick attende un passo. L'orologio è a 120 tick al secondo e un passo corrisponde a un sedicesimo, quindi 15 tick corrispondono a 120 battiti al minuto. Un TIC più alto corrisponde a un pattern più lento. Due pattern rimangono sincronizzati quando un TIC è divisibile per l'altro: 15 e 30 lo sono, 15 e 8 no.

Il pianoforte suona quando si seleziona una nota con il mouse. I tasti bianchi corrispondono a ZXCVBNM (CDEFGAB). I tasti neri corrispondono a SDGHJ (C# D# F# G# A#). Passando il mouse sopra un valore, è possibile digitare un numero oppure ruotare la rotella. I numeri delle altezze dei suoni vanno da 0 (C-0) a 63 (D#5).

In un pattern, LOOP e LEN corrispondono agli stessi due numeri. OUT sopra 0 ripete il ciclo da IN a OUT e il titolo è LOOP. OUT a 0 e IN sopra 0 è LEN: vengono riprodotte tante note dall'inizio, poi il pattern termina e il titolo è LEN. Entrambi a 0 vengono riprodotti a ogni passo.

La quantità di sintetizzatore che è possibile modificare dipende dalla profondità di bit scelta per la cartuccia di gioco. IMPOSTAZIONI → EDITOR EFFETTI SONORI imposta tale profondità, se una nota avanza automaticamente (disattivato, catena di note o catena di campi), se la barra spaziatrice riavvia la riproduzione o attiva/disattiva l'avvio e l'arresto, e i colori dell'editor.

16 bit

Ogni passaggio ha una nota, un volume, una forma d'onda e un effetto. Il volume va da 0 a 7. 0 corrisponde al silenzio.

Onde:

- **0 triangolo**
- **1 sega inclinata**
- **2 seghe**
- **3 quadrati**
- **4 impulsi** — un impulso stretto
- **5 organi**
- **6 rumore**
- **7 phaser**
- **8 PWM** — un quadrato la cui larghezza si muove
- **9 impulsi 25%**
- **10 seno**
- **11 rumore metallico**
- **12 FM bell**
- **13 pizzicare**

- **14 supersaw**
- **15 PWM (C64)**

Effetti su una singola nota:

- **0 nessuno**
- **1 diapositiva** : l'intonazione e il volume si spostano gradualmente alla nota successiva su questo gradino.
- **2 vibrato** — oscillazioni di intonazione
- **3 drop** — il tono cade nel silenzio, il solito calcio
- **4 dissolvenza in entrata**
- **5 dissolvenza in uscita**
- **6 arpeggio** — esegue questa nota e le tre successive, una ogni 8 tick. Al TIC 8 o inferiore, ogni 4 tick.
- **7 arpeggio veloce** : le stesse quattro note, ogni 4 tick, o ogni 2 quando TIC è pari o inferiore a 8.

L'intestazione del modello, sopra le note:

- **TIC** — tick per passo, da 1 a 255. 15 corrisponde a 120 battiti al minuto. Digita il numero o incrementalo di uno.
- **LOOP IN / OUT** — con OUT maggiore di 0, il pattern si ripete da IN a OUT. Lasciando OUT a 0, il titolo diventa LEN: IN indica quante note vengono riprodotte e 0 indica l'intero pattern.
- **RUMORE** — l'intero schema diventa rumore.
- **CRUSH** — una variante vibrante di ogni onda.
- **DTUNE** — 0 spento, 1 un ritornello leggero, 2 un ritornello forte.
- **RVERB** — riverbero: 0 disattivato, 1 breve, 2 lungo.
- **DAMPN** — un filtro passa-basso: 0 spento, 1 leggero, 2 forte.
- **LIVELLO** — volume dell'intero schema, da 0 a 7.
- **SWING** — passo incrociato. I passi dispari aspettano un po' di più, i passi pari un po' di meno.
- **PASSAGGI** — indica quanti passaggi richiede il modello: 32, 24, 16 o 12.

8 bit

Uno step ha una nota e un'onda. Le quattro forme d'onda sono quadra, triangolare, a dente di sega e rumore. L'intestazione contiene TIC, LOOP IN/OUT, FX e LEVEL. TIC può assumere uno dei seguenti valori: 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40, 60. FX indica un effetto per l'intero pattern: 0 nessuno, 1 vibrato, 2 arpeggio, 3 dissolvenza in uscita. LEVEL è compreso tra 0 e 3.

4 bit

Un passo è solo una nota. L'intestazione ha TIC, LOOP (acceso o spento), WAVE (le stesse quattro onde) e OCT (l'ottava in cui si trovano le note). TIC può essere uno tra 4, 6, 8, 10, 12, 15, 20, 30.

2 bit

Uno step può essere vuoto, un tamburo, un colpo o una schiacciata. L'intestazione ha TIC e BASE. TIC può essere 8, 15, 16 o 30. BASE imposta l'altezza del tamburo e del colpo.

1 bit

Un passo corrisponde a una nota. TIC è 8, 15, 16 o 30, le stesse quattro velocità del 2-bit.

Canzoni

La scheda SUONO mostra il brano aperto. Otto colonne rappresentano i canali. Ogni cella ha un ID di pattern. Il valore 255, se vuoto, indica che il canale corrispondente è silenzioso in quella riga. La colonna X rappresenta un comando: continua, interrompi, avvia loop o termina loop. Interrompi termina il brano dopo la riga corrente. Termina loop torna all'ultimo avvio del loop.

Una riga termina quando il motivo più a sinistra con LOOP OUT a 0 è terminato. I motivi più a destra non allungano la riga: un motivo più lungo viene tagliato, un motivo più corto ricomincia dal suo primo passo finché la riga non termina. Un motivo con LOOP OUT superiore a 0 si ripete ciclicamente tra LOOP IN e LOOP OUT e non termina la riga. Se tutti i motivi della riga si ripetono ciclicamente, la riga rimane invariata.

Scorciatoie

Chiave	Azione
Tasti del pianoforte	Scrivi una nota sul passaggio selezionato con il mouse.
Su / Giù	Modifica il valore sotto il cursore.
Spazio	Riproduci gli effetti sonori aperti o il brano aperto dalla prima riga. Avvia/interrompi o riavvia dalle impostazioni dell'EDITOR SFX.
Ctrl+Z / Ctrl+Y	Annulla / Ripeti.
Ctrl+C / Ctrl+X / Ctrl+V	Copia, taglia e incolla la selezione.

DATI

Tabelle denominate. Ogni tabella ha righe e colonne. Nel codice, `dget`, `dset`, `dstr`, `drows` e `dcols` utilizzano queste tabelle. I numeri di riga e di colonna nel codice iniziano da 0.

Tipi di colonna, visualizzati ciclicamente nell'intestazione:

- **bool** — un bit.
- **2** — due bit.

- **4** — quattro bit.
- **8** — un byte.
- **16** — 16 bit, per ID e numeri più grandi.
- **spr** — un ID di sprite.
- **str** — testo, disegnato con il carattere di sistema.
- **str8** e **str10** : lo stesso testo, disegnato più alto.
- **f16** e **f32** : un float a mezza precisione e un float a piena precisione.

Con il puntatore sull'intestazione di una colonna, Ctrl+rotellina del mouse cambia il tipo di colonna. La semplice rotellina del mouse si sposta tra le colonne o tra le righe, a seconda dell'impostazione selezionata in IMPOSTAZIONI → EDITOR DATI. Ctrl inverte la direzione, motivo per cui Ctrl+rotellina del mouse sull'intestazione cambia il tipo quando la rotellina è impostata su colonne.

I tasti freccia consentono di spostare le celle. Il tasto Invio modifica una cella. IMPOSTAZIONI → EDITOR DATI:

- **Rotella** — scorri colonne o scorri righe.
- **Colori** : della griglia, dell'intestazione e della selezione.

Attività

ASSETS importa ed esporta il componente aperto, non l'intero progetto. Gli sprite utilizzano lo sprite aperto in PAINT, sul foglio aperto. La mappa è la mappa aperta. SFX è il pattern aperto. Musica è il brano aperto. Dati è la tabella aperta. Codice è l'eccezione: ogni tabulazione viene scritta e ogni tabulazione viene letta.

Tipo	File	Cosa si muove?
Sprite	`.png`	Lo sprite aperto.
Mappa	`.p37map`	La mappa aperta.
Effetti sonori	`.p37sfx`	Il modello aperto.
Musica	`.p37mus`	Il brano di apertura. I numeri dei pattern rimangono invariati. Esporta gli effetti sonori separatamente se ti servono.
Dati	`.p37dat`	La tavola aperta.
Codice	`.lua`	Ogni scheda.

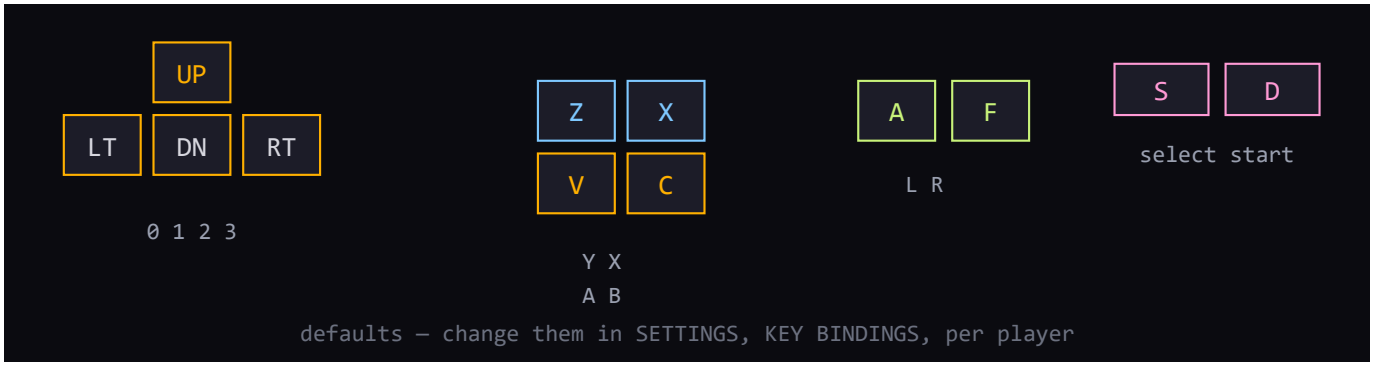
Impostazioni

L'elenco delle radici:

Pagina	Cosa memorizza
PROFILI	Salva e carica ogni impostazione. Crea, rinomina, sovrascrivi, elimina. Copia o incolla il profilo come testo.
Menu	Colori di finestra, tasti, menu e sfondo opzionale.
PANNELLO DI GIOCO	Anteprima e finestra di riproduzione.
EDITOR DI CODICE	Evidenziazioni e suggerimenti.
EDITOR DI PITTURA	Pennello, tavolozza, vista mappa.
EDITOR DI EFFETTI SONORI	Sintesi e inserimento delle note.
EDITOR DI DATI	Direzione e colori delle ruote.
LINGUA	Lingua dei menu e dei suggerimenti.
LICENZE	Testi di licenza, inclusi il font Fusion Pixel e SDL3.

GLOBALE:

- **DISPOSIZIONE** — disposizione dei pannelli, sopra.
- **VOLUME** — livello master, da 0 a 10.
- **ASSEGNAZIONE DEI TASTI** — tasti per ciascuno degli 8 giocatori. Premi un tasto per catturarlo. Direzioni, ABXYLR, seleziona.
- **Presente** : salta i fotogrammi inattivi oppure disegna ogni fotogramma.
- **Schermo intero** - senza bordi.
- **Aggiorna** — FPS della finestra. Incremento a sinistra e a destra di 60, 90, 120 e massimo. Inserisci un numero fino a 1000. 0 significa nessun limite.
- **Vsync** : attende che la visualizzazione si completi tra un fotogramma e l'altro.
- **Scala** — moltiplicatore della finestra intera.
- **Shader** : CRT e filtri nell'editor, non nel gioco.
- **Colore** : testo del menu ed evidenziazione.
- **Sfondo** : un'immagine PNG opzionale, ritagliata a 128 pixel, posizionata dietro il menu. L'opzione "Cancella" la rimuove.
- **Cursore dell'editor** : forma del puntatore.
- **Mouse** : puntatore mobile o agganciato ai pixel.
- **Velocità dell'interfaccia utente** : la velocità con cui una direzione tenuta premuta si ripete in un elenco.
- **Esplora risorse** : gli elenchi dei file iniziano dalla directory principale del computer o dalla cartella del file eseguibile (.exe).



Tastiera predefinita. A conferma nei menu. B torna indietro. Nei giochi corrispondono ai tasti 4 e 5. Select (S) è il tasto 10, non una scorciatoia per i menu.

Pulsante	Ruolo	Chiave predefinita
0	Sinistra	Freccia sinistra
1	Giusto	Freccia a destra
2	Su	freccia in su
3	Giù	Freccia in basso
4	A, confermare	V
5	B, indietro	C
6	X	X
7	Y	Z
8	R	F
9	L	UN
10	Selezionare	S
11	Inizio	D

Costruire

Scegli una piattaforma: Windows, Linux, Mac, Android, iOS o Web.

- **Nome** — nome del file di compilazione.
- **Autore** — su Windows e Linux. Questo è il nome della cartella per i salvataggi.
- **ID dell'app Steam** — su Windows e Linux. Un valore vuoto indica l'assenza di Steamworks. Un ID compilato associa i valori `achieve(0)`…`achieve(15)` agli obiettivi Steam ACH_0…ACH_15.

- **ID** — identificativo del pacchetto, laddove richiesto dalla piattaforma.
- **Nascondi il codice** — su overlay per desktop e dispositivi mobili. Rallenta `peek` della cartuccia di gioco. Non è un blocco.
- **Schermo intero** : il lettore esportato si avvia a schermo intero.
- **Shader** : fino a tre livelli integrati nel player.
- **Scarica il file .p37** — Solo per il Web. Mostra la voce di download nel menu di pausa.
- **Icona** : una tessera da un foglio di sprite: foglio, dimensioni, x, y. Oppure una copertina PNG che ritagli.
- **Esporta** — seleziona la cartella. Il menu visualizza BUILD SAVED o BUILD FAILED.

Crea una build con la connessione internet attiva. La schermata iniziale potrebbe chiedere al server, in modo anonimo, se l'hash di questa build è noto. Questo controllo serve a due scopi. Protegge me, l'autore di PIX37, da copie del programma che non mi spettavano e che ho distribuito senza autorizzazione. Protegge anche chi ha creato il gioco. Prendere il gioco di qualcun altro e spacciarlo per proprio sta diventando una pratica comune sul mercato dei dispositivi mobili, come Play Store (Android) o iOS. Il gioco si avvia comunque. Se il dispositivo è offline, l'introduzione indica che il gioco non è connesso. Questa frase è anche un suggerimento per il giocatore che la copia potrebbe non provenire da una fonte legale. È un piccolo controllo, in modo che se il tuo gioco è stato rubato e venduto senza il tuo consenso da qualcun altro, tu possa presentare un reclamo su uno store come Play Store (Android), Steam o itch.io. Il server memorizza l'ora in cui l'hash è stato registrato per la prima volta, accanto all'hash stesso, e non lo modifica in seguito. L'ora precedente corrisponde alla build originale. Un hash registrato successivamente corrisponde a una copia successiva.

Voglio che questo controllo serva a proteggere chi sviluppa i videogiochi. Il motore di gioco fa già gran parte del lavoro per loro, e questo è uno di quegli aspetti. Capisco che possa essere controverso. Se la community deciderà che è meglio abbandonare questo micro-DRM, lo farò.

Una vecchia cartuccia di gioco proveniente da un editor precedente potrebbe non avviarsi dopo il cambio di lingua. Apri il progetto `.pix37` in questo editor ed esporta un nuovo `.p37`.

Connettiti a PIX37.PARTY

CONNECT TO PIX37.PARTY è integrato nell'app. Non è presente una pagina di accesso nel browser. L'accesso avviene tramite un ticket di sessione Steam o tramite itch.io, utilizzando il gamepad e il mouse. Il servizio mostra un nome visualizzato, da 2 a 16 caratteri, univoco. La chiave di licenza e l'ID Steam non sono visibili agli altri giocatori.

Dopo i termini di servizio, la sezione principale comprende Preferiti, Catalogo, Classifica, Giocatori e il tuo profilo.

La licenza Sviluppatore è per l'editor. Crei il gioco, lo carichi e lasci delle stelle. La licenza Giocatore è per il giocatore. I cuori provengono dalle persone che giocano su un runner.

- **Catalogo** — sfoglia i giochi. I filtri includono genere (fino a tre), dimensioni dello schermo, dimensioni della cartuccia di gioco e se il caricamento include un progetto `.pix37`.

- **Preferiti** : i giochi che hai contrassegnato.
- **Caricamento** — richiede una licenza per sviluppatori. Devi inviare un file `.p37` e puoi allegare il progetto `.pix37`. Un file `.p37` è un bytecode compresso. È difficile da leggere e può essere offuscato intenzionalmente. Un file `.pix37` è il file di progetto su cui lavori effettivamente: il codice sorgente, le immagini e l'audio, in un formato che un'altra persona può aprire e da cui può imparare. Inviandolo, altri creatori possono studiare il gioco invece di una cartuccia di gioco compressa. Vengono inviati anche il titolo, l'autore, i generi, la copertina, l'icona e i nomi degli obiettivi. Il valore di RAM inviato corrisponde al picco di RAM raggiunto dalla cartuccia di gioco.
- **Commenti e obiettivi** — sulla pagina del gioco. Lo sblocco di un obiettivo mentre si è connessi viene memorizzato sia su quell'account che sul computer.
- **Classifica** degli sviluppatori, ordinati per stelle o per cuori. Ogni riga mostra entrambi i conteggi. Apri un nome per visualizzare i giochi di quell'autore, ordinati per cuori.
- **Giocatori** : account ordinati in base agli obiettivi sbloccati.

Codice

Il linguaggio è un piccolo script in stile Lua. Questa parte inizia da un progetto vuoto, insegna il linguaggio e poi mostra un breve esempio di ogni funzione.

Un primo programma

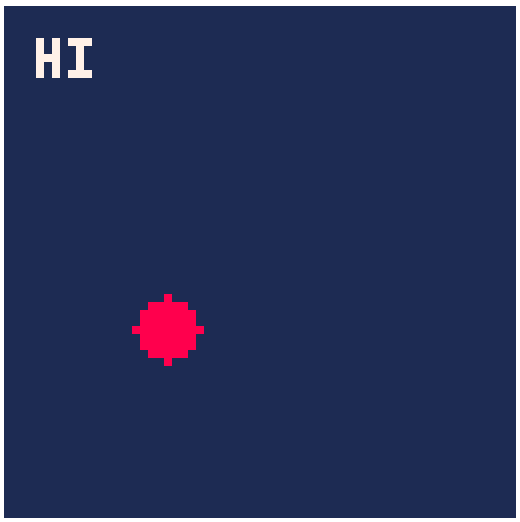
Crea un progetto a 64×64, colore a 4 bit. Apri CODE. Sostituisci la scheda con:

```
_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 avvia il gioco. I tasti freccia muovono il cerchio. Esc interrompe il gioco e torna all'editor.



Le stesse tre chiamate: `cls(1)` disegna lo schermo blu scuro, `print` disegna HI e `circfill` disegna il cerchio rosa.

Cosa funziona e quando

- `_init` viene eseguito una sola volta.
- `_update` funziona a 60 Hz. Inserire qui il movimento e i pulsanti.
- `_draw` esegue il disegno. Senza l'argomento fps, disegna alla massima velocità consentita dal computer e dal monitor. `_draw(30)` limita il disegno a 30 fps e lascia la logica a 60.
- `_set_fps(n)` modifica quel condensatore in seguito. 0 lo azzera.

I nomi non fanno distinzione tra maiuscole e minuscole; l'editor li memorizza in minuscolo. `function` e `fun` sono la stessa parola chiave.

Lingua

Nozioni di base di programmazione

Un programma è un elenco di righe eseguite dalla cartuccia di gioco. Ogni intestazione mostra prima la struttura della riga, poi un piccolo esempio. Una riga che inizia con `--` è un commento: una nota per te. Il gioco la ignora.

Variabili

Una variabile è un nome che contiene un valore. Il nome viene scelto dall'utente. Le righe di codice successive possono leggerlo e memorizzare un nuovo valore con lo stesso nome.

La forma è `name = value`. `local name = value` mantiene quel nome all'interno del blocco corrente, in modo che il resto della cartuccia di gioco non possa vederlo. Senza `local`, il nome è globale e può essere visto da ogni scheda.

```
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 che non c'è alcun valore. `true` e `false` significano sì e no. Un nome che non hai mai impostato viene letto come `nil`.

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

Tavoli

Una tabella contiene molti valori sotto un unico nome. Un elenco utilizza i numeri. Il primo elemento è 1, non 0. Un record utilizza i nomi. Una tabella può contenere un'altra tabella.

Formato elenco: `{ value, value }`. Formato record: `{ name = value, name = value }`. Un elenco di record è una tabella i cui elementi sono tabelle.

```
-- 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
```

Condizione e corpo

Una condizione è una domanda con una risposta sì o no, come `hp > 0` o `btnd(4)`. Il corpo è il gruppo di righe che vengono eseguite quando si entra in un blocco. `if` e ogni ciclo utilizzano la stessa struttura: una condizione, poi un corpo, poi `end`. `repeat` si chiude con `until` invece di `end`.

Se

`if` esegue il suo corpo solo quando la condizione è vera. `elseif` prova un'altra condizione. `else` viene eseguito quando tutte le condizioni sono false.

Forma: `if condition then`, poi il corpo, poi `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
```

Cicli

Un ciclo ripete il corpo di un'istruzione. `while` controlla prima la condizione e continua finché questa rimane vera. `repeat` esegue il corpo dell'istruzione, poi controlla, quindi il corpo viene sempre eseguito almeno una volta. `for` conta. L'ultimo numero è incluso. Senza un terzo numero, il passo è 1, quindi se l'inizio è già dopo la fine, il corpo non viene eseguito. Il conteggio all'indietro richiede un passo 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
```

Camminare su un tavolo

`all` visita ogni elemento di una lista, a partire dal primo. `pairs` visita ogni nome e il valore memorizzato sotto quel nome.

```
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
```

Funzioni

Una funzione è una ricetta denominata. La sua struttura è `function name(arguments)`, seguita da un corpo e poi da `end`. La si chiama scrivendo il nome tra parentesi. `return` restituisce un valore, e può restituirne più di uno. Gli argomenti sono variabili locali di quella funzione.

```
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` è una funzione che può accedere alle variabili locali circostanti. Si tratta di una closure. Una tabella può contenerne una e la si richiama 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
```

Sì, no e scegliere

`not` inverte una condizione. `and` è vero solo quando entrambi i lati sono veri. `or` è vero quando almeno uno dei due lati è vero. `a and b or c` sceglie `b` quando `a` è vero, altrimenti `c`. `cond ? a : b` è la stessa idea in una riga: `a` quando la condizione è vera, `b` quando non lo è.

```
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)
```

Matematica e testo

`+ - * /` rappresenta addizione, sottrazione, moltiplicazione e divisione. `//` divide mantenendo il numero intero. `%` rappresenta il resto. `^` rappresenta la potenza. `..` unisce due testi. `#t` indica il numero di elementi in una lista. I confronti sono `==` (uguale), `~=` o `!=` (diverso) e `< > <= >=`. `x += 1` aggiunge 1 a x. La stessa forma funziona per `-= *= /= ^= %=`.

```
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
```

Dove vive un abitante del posto

`local` dura fino a quando `end` chiude il suo blocco. `if` / `end` costituisce un blocco a sé stante. È possibile specificare una variabile locale sopra la riga che la imposta, ma fino a quando tale riga non viene eseguita, il valore è `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
```

saltare

`::label::` contrassegna una riga. `goto label` salta in quella posizione, ma solo all'interno della stessa funzione.

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

La memoria, in breve

`peek` e `poke` leggono i pixel in tempo reale tramite un indirizzo a 16 bit. Non mantengono una seconda copia della cartuccia di gioco. Gli indirizzi da `0x5F00` a `0x5FFF` corrispondono ai registri: telecamera, clip (sola lettura), colore di disegno, schema di riempimento, le tre rimappature della tavolozza e i numeri di pagina. I byte su entrambi i lati di quel foro, 65280 in tutto, costituiscono la finestra dei pixel.

Quando lo schermo rientra nei 65280 byte, 64×64 e 128×128 lo fanno mentre 256×256 no, lo schermo inizia dall'indirizzo 0, riga per riga. Ciò che rimane è il foglio degli sprite selezionato da `0x5F41` (0 è il foglio 0). Se tale foglio è più grande dello spazio rimanente, `0x5F42` seleziona la pagina e la mappa non è nella finestra. Se il foglio rientra, la parte finale è costituita dalle celle della mappa selezionate da `0x5F43`, uno o due byte ciascuna, e `0x5F44` sposta la mappa a pagina quando non rientra nella parte finale.

Quando lo schermo non si adatta, l'intera finestra è una pagina di 65280 pixel. `poke(0x5F40, n)` cambia pagina dello schermo. `poke(0x5F41, n)` con n diverso da 0 mostra invece il foglio n, e il foglio 0 non è visibile in questa modalità. `poke(0x5F43, n)` con n diverso da 0 mostra la mappa n, che ha la precedenza sul foglio. La mappa 0 in questa modalità è rappresentata solo da `mget` e `mset`.

Un `peek` che non atterra su un pixel o un registro restituisce 0. Un `poke` in quella posizione non ha alcun effetto. I flag degli sprite, gli SFX e le tabelle DATA non sono in questa finestra. Utilizzare `fget`, `sfxget` e `dget`.

Uso pratico

In fondo a questa pagina sono elencate tutte le funzioni, con esempi pratici di utilizzo delle funzioni di PIX37. I link sotto ogni esempio forniscono ulteriori informazioni sulle funzioni.



movimento fluido

Una cartuccia di gioco 128×128, vista dall'alto. Lo sprite 1 è già disegnato. Le frecce accelerano in qualsiasi direzione e l'attrito rallenta. Lo sprite si avvolge: la metà che esce dal bordo inferiore viene disegnata in alto, e lo stesso accade per gli altri bordi.

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



Verifica se la prossima mossa è bloccata

Una cartuccia di gioco 64×64. Il pavimento e la parete destra sono già presenti sulla mappa, colorati con lo sprite 3, e la bandiera 0 è impostata su quello sprite. Le frecce muovono lo sprite 1. La scritta in alto indica "bloccato" quando il passo successivo si sovrapporrebbe a una tessera piena, e "libero" quando non si sovrapporrebbe.

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



Una salvietta con capovolgimento

Premi A. La dissolvenza viene disegnata da flip(), una riga nera alla volta, invece che all'interno di _draw(). _draw non interferisce durante l'esecuzione di flip(), quindi ogni riga rimane sullo schermo.

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



Una tabella di funzioni (Classe)

Lo sprite 1 è già stato disegnato. make_monster funziona come una classe. Restituisce una tabella, e update e draw sono metodi che mantengono le coordinate x e y di quel mostro. Il numero in alto indica la distanza percorsa.

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



Quattro modi per leggere un pulsante

A è il pulsante 4. Tienilo premuto e la carica aumenta. Rilascialo e la carica torna a 0. Su e giù si ripetono mentre lo tieni premuto, quindi la riga del menu continua a spostarsi. btnd è il primo fotogramma di una pressione, che questa cartuccia di gioco usa solo come inizio di quella pressione prolungata.

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



Spara con il mouse

Lo sprite 1 è già disegnato al centro. Il punto bianco è il cursore. Il clic sinistro spara un pixel verso di esso. La rotella cambia la velocità, indicata nell'angolo.

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



Digita un nome e confermalo

Premi A per aprire il campo. Su e giù cambiano una lettera lampeggiante. A scrive, X cancella, Y è spazio e B conferma. La riga poi dice il tuo nome.

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



Telecamera sul giocatore

Una cartuccia di gioco 128×128. Lo sprite 1 inizia al centro di un pavimento di piastrelle. Le frecce si muovono. La telecamera mantiene lo sprite centrato. Gli assi x e y nell'angolo non scorrono, perché la funzione camera() senza argomenti riporta la visuale all'interno dell'HUD.

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



Alcune righe di una visual novel

Le righe sono in DATA, in inglese e polacco. A continua, o seleziona resta quando è presente un'opzione sullo schermo. B seleziona esci quando è presente un'opzione, altrimenti cambia la lingua. X cambia la lingua mentre è visualizzata un'opzione. Il dialogo utilizza un carattere da 8px, quindi le lettere polacche vengono visualizzate correttamente.

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



Dipingere con la tecnica del "peek and poke"

Una cartuccia di gioco 64×64, quindi l'indirizzo 0 è il pixel in alto a sinistra e la riga successiva inizia da 64. La cartuccia di gioco scrive i pixel, copia una riga, cancella una riga e stampa i valori letti.

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



Un elenco che puoi aggiungere

A aggiunge un numero, fino a un massimo di 8. B rimuove l'ultimo. X rimuove il primo. L'elenco viene stampato sullo schermo dall'alto verso il basso.

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



Un nome scomposto

La cartuccia di gioco divide ada,12,blue e stampa il nome, la sua prima lettera, 12, e la parola blue.

`cls split sub chr ord tonum tostr print _draw`

 Suona una nota, poi cambiala

Il pattern 0 è composto da una sola nota e ogni ripetizione ha la stessa intensità. A lo riproduce di nuovo quando la nota è terminata. B alza l'intonazione e riproduce quella nota.

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

 Una canzone scritta nella cartuccia del gioco

Due pattern si ripetono sulla riga 0, uno su ciascuno dei primi due canali. La riga rimane 0. ch0 e ch1 rappresentano la nota su cui si trova ciascun canale. Modificare il pattern, LEN e LOOP nella cartuccia di gioco è un modo per vedere cosa fanno questi controlli nel pannello musicale.

`songget songset music musicstat cls print _init _draw`

 Tre mostri nei DATI

Il pannello DATI ha già una colonna nome e una colonna hp, con slime, pipistrello e occhio. A ferisce lo slime. I numeri a destra sono i suoi hp.

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

 Ricorda il punteggio

A aggiunge un punto e lo salva. Esce e riapre la cartuccia di gioco: il numero è ancora lì. B lo dimentica.

`load btnd save delsave cls print _init _update _draw`

  Un secondo capitolo in un altro file

Due cartucce di gioco condividono lo slot di salvataggio. Parte 1: A indica che il gioco è terminato. La Parte 2, aperta per prima, indica di completare la Parte 1. Dopodiché, si apre la Parte 2 e B cancella il salvataggio in modo che la Parte 1 possa essere giocata di nuovo.

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

 Un risultato

Lo sprite 8 è già stato disegnato. Premi A dieci volte. Il messaggio a comparsa indica il risultato dell'obiettivo e lo sprite 8 rimane perché è sbloccato. B cancella l'obiettivo dal file locale e azzera il conteggio.

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



Una stanza in cui puoi entrare

Una cartuccia di gioco 256×256. Il punto bianco è il cursore. Il menu permette di scegliere tra "Host", "Join" o "Elenco server". "Host" apre una stanza chiamata "Walk" e avvia la partita. "Join" fa scorrere un indirizzo (cifre, un punto, due punti) e B si connette. L'elenco dei server si sposta con i tasti su e giù e A si unisce alla riga. Nel gioco, le frecce muovono lo sprite 1. Una moneta è condivisa: l'host la muove e raccogliendola si aggiunge 1 al punteggio di quel giocatore e si fa apparire il successivo. Ogni sprite, incluso il tuo, mostra il punteggio accanto. Nell'angolo in alto a destra vengono mostrati il numero di giocatori presenti, l'indirizzo LAN e l'indirizzo pubblico dell'host. B esce: tu e gli altri tornate al menu e l'host successivo avvia una nuova stanza con punteggio 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`



Un dialogo che aspetta

A indica la riga successiva. La funzione si interrompe a ogni interruzione di riga finché non si preme nuovamente A.

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



Aspetta mezzo secondo

La parola cambia da ... a ciao dopo circa mezzo secondo. Lo schermo continua a disegnare mentre la coroutine è in pausa.

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



Cerchi che arrivano uno ad uno

Cinque cerchi gocciolano dall'alto, a due fotogrammi di distanza l'uno dall'altro, per poi cadere.

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



Due fotogrammi, dall'orologio

Lo sprite 1 e lo sprite 2 sono due fotogrammi dello stesso personaggio. L'immagine si scambia quattro volte al secondo.

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

Ogni funzione

Gli argomenti tra parentesi quadre sono facoltativi. Gli indici di tabelle e stringhe iniziano da 1, ad eccezione di righe e colonne DATA, che iniziano da 0. I colori sono indici di tavolozza.

Indice

_draw
_init
_mp
_mp_fail
_mp_join
_mp_lascia
_mp_aggiornamento
_set_fps
_aggiornamento
addominali
achclear
achdef
raggiungere
raggiunto
aggiungere
tutto
affermare
atan2
banda
bnon
bor
comunque
comunque
btnp
btnu
bxor
fotocamera
ceil
chr
circ
Circfill
cliccal
cliccam
clickr
clip
cls
cocreate
colore
coresume
cos
costato
contare
cursore
dcols
del
gastronomia
delsave
dget
drows
dset
dstr
vabbè
riempi
capovolgere
flr
per ciascuno
fset
informazioni
ipairs
keyoff
keyon
keystat
tasti
linea
caricare
registro
lshr
mappa
massimo
memcpy
memset
capisco
metà
min
topo
mset
musica
musicstat
nannunciare
ncall
successivo
nhost
unisciti
nkb
nkbstr
nlan
nlasciare
nlist
nascolta
nota
nplayers
nsend
nstun
ord
ovale
ovalfill
pacco
coppie
amico
palrgb
pal
pcall
sbirciare
sbirciare2
sbirciare4
perf
pget
colpire
poke2
poke4
stampa

stampa10	setpalette	set di canzoni
stampa8	setpr	diviso
pset	settex	spr
rect	setturn	mq
riempimento	sfx	srand
rnd	sfxget	sset
rotl	sfxhdr	sspr
rotr	sfxset	stat
corretto	sfxstat	sost
rrectfill	sget	tempo
salvare	segno	tline
saveslot	shl	tonum
schermo	showstats	tostr
scorrere	shr	disfare le valigie
selezionare	peccato	whoami
setmap	songget	rendimento

Ciclo vitale

_init

`_init() ... end`

Viene eseguito una sola volta all'avvio della cartuccia di gioco. Il codice che si trova al di fuori di qualsiasi hook viene anch'esso incluso in _init.

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

_update

```
_update() ... end
```

Logica di gioco. Chiamata a 60 Hz, anche quando _draw è limitato o l'host è più lento. Leggi i pulsanti qui.

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

_draw

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

Disegna un fotogramma. Senza argomenti, o con valore 0, viene eseguito con la frequenza richiesta dall'host. Passare n (1..999) per limitare il disegno a n fotogrammi al secondo. _update rimane a 60 Hz.

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

_set_fps

```
_set_fps(n)
```

Modifica il limite di _draw mentre il gioco è in esecuzione. 0 significa illimitato. n è limitato tra 0 e 999.

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

flip

```
flip()
```

Presenta la schermata e cede il controllo fino al frame successivo. Usalo solo se stai disegnando il tuo ciclo di frame. Non attendere in background dopo averlo eseguito.

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

time

`time()`

Secondi trascorsi dall'inizializzazione.

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

Ingresso

btn

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

1 finché il pulsante b è tenuto premuto, altrimenti 0. p è il giocatore, 0..7. Senza argomenti restituisce una maschera di bit del giocatore 0. Pulsanti: 0 sinistra, 1 destra, 2 su, 3 giù, 4 A, 5 B, 6 X, 7 Y, 8 R, 9 L, 10 seleziona, 11 avvia.

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

btnd

`btnd(b, [p])`

Vero solo nel frame _update quando il pulsante b viene premuto.

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

btnp

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

Vero alla pressione, poi di nuovo tenendo premuto: 30 tick di ritardo, poi ogni 8 tick. Senza pulsante restituisce una maschera.

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

btnu

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

Vero solo nel momento in cui il pulsante viene rilasciato.

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

mouse

```
mouse()
```

Restituisce le coordinate x e y del mouse in pixel di gioco.

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

clickl

```
clickl([p])
```

Pulsante sinistro del mouse. clickl() è tenuto premuto. clickl(1) è il fronte di pressione.

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

clickr

```
clickr([p])
```

Pulsante destro del mouse. Stesso bordo a 1 argomento di clickl.

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

clickm

`clickm([p])`

Pulsante centrale, oppure un clic della rotellina del mouse.

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

scroll

`scroll()`

Movimento della rotellina del mouse su questo fotogramma. Positivo o negativo a seconda della direzione della rotellina.

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

keyon

`keyon()`

Apre un campo di testo controllato da pulsanti. Su e giù cambiano una lettera dalla A alla Z. A scrive la lettera. X cancella. Y è spazio. B conferma e chiude il campo. La tastiera non digita al suo interno.

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

keyoff

`keyoff()`

Nasconde il campo di testo.

```
keyoff()
```

keystr

```
keystr([s])
```

Testo nel campo, poi 1 sul riquadro B conferma, quindi la lettera su e giù cambiano. Quella lettera è vuota una volta chiuso il campo. keystr("") cancella il testo.

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

keystat

```
keystat()
```

1 finché il campo di testo è aperto, altrimenti 0.

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

Disegno

screen

```
screen()
```

Dimensioni attuali dello schermo della cartuccia di gioco in pixel. Restituisce prima la larghezza e poi l'altezza. Questa è la risoluzione del gioco, non della finestra dell'editor.

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

cls

```
cls([col])
```

Cancella lo schermo. Il colore predefinito è 0.

```
cls(1)
```

color

`color(col)`

Imposta il colore di disegno utilizzato quando una chiamata successiva omette col. Senza argomenti restituisce il colore corrente.

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

camera

`camera([x, y])`

Esegue l'offset ad ogni chiamata di disegno. camera() reimposta a 0, 0.

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

clip

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

Limita il disegno a un rettangolo. Un quinto argomento diverso da zero interseca il ritaglio corrente. clip() ripristina l'intero schermo. L'anteprima dei registri di ritaglio è di sola lettura.

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

cursor

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

Imposta la posizione della prossima stampa senza coordinate e, facoltativamente, il colore. cursor() restituisce x, y.

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

pset

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

Imposta un pixel dello schermo. Contribuisce al conteggio dei pixel.

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

pget

`pget(x, y)`

Colore del pixel dello schermo alle coordinate x, y.

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

print

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

Disegna testo 3×5 (UTF-8, ritagliato dal font Fusion). Senza x, y, stampa in corrispondenza del cursore e passa alla riga successiva.

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

log

`log(...)`

Aggiunge gli argomenti, separati da spazi, a un buffer di testo. pix37_run stampa quel buffer sullo standard output. Non effettua alcun disegno.

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

print8

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

Come nella versione stampata, con il carattere Fusion da 8 px.

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

print10

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

Come nella versione stampata, con il carattere Fusion da 10 px.

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

line

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

Linea da x0,y0 a x1,y1. Due argomenti continuano dall'estremità precedente. line() reimposta quella catena.

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

circ

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

Contorno del cerchio. Il raggio è espresso in pixel.

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

circfill

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

Cerchio pieno.

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

oval

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

Contorno ellittico all'interno del riquadro di delimitazione. Gli angoli sono inclusi.

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

ovalfill

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

Ellisse piena in un riquadro di delimitazione.

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

rect

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

Contorno rettangolare. Gli angoli sono inclusi.

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

rectfill

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

Rettangolo pieno.

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

rrect

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

Contorno rettangolare arrotondato. x, y indica l'angolo in alto a sinistra. w e h rappresentano le dimensioni in pixel; entrambi devono essere maggiori di 0. r è il raggio degli angoli e il suo valore predefinito è 0. Viene limitato alla metà del lato più corto.

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

rrectfill

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

Rettangolo arrotondato pieno. Stessi argomenti di retto.

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

fillp

`fillp([p])`

Schema di riempimento a 16 bit per le forme riempite in seguito. 0 indica pieno. fillp() restituisce lo schema corrente. I bit percorrono lo schema 4×4.

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

pal

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

Rimappa il colore c0 in c1. La modalità 0 è la tavolozza di disegno, la 1 la tavolozza di visualizzazione, la 2 la tavolozza del motivo di riempimento. pal() reimposta.

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

palrgb

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

Imposta una voce della tavolozza su RGB, ciascun canale da 0 a 255.

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

palt

```
palt([col, t])
```

Trasparenza. palt(col) o palt(col, true) rende col trasparente. palt(col, false) lo rende opaco. palt() rende tutti i colori opachi.

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

setpalette

```
setpalette(n)
```

Seleziona il banco di tavolozze n. setpalette() restituisce il banco corrente.

```
setpalette(1)
```

tline

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

Traccia una linea sullo schermo campionando i pixel della mappa dalla posizione mx, my. mdx, mdy e fa avanzare il campione della mappa lungo la linea.

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

Sprite e mappe

spr

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

Disegna lo sprite n in x, y. w e h sono le dimensioni in pixel, il valore predefinito è 8. fx e fy capovolgono orizzontalmente e verticalmente quando sono diversi da zero. tex è la tabulazione del foglio degli sprite. rot è l'angolo in giri (0..1), nella stessa direzione di sin e cos. 0 lascia lo sprite non ruotato.

```
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 rettangolo del foglio di sprite sullo schermo, ridimensionandolo da dw a dh. Le impostazioni predefinite mantengono le dimensioni originali.

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

sget

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

Colore di un texel sul foglio degli sprite.

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

sset

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

Scrive un texel sul foglio degli sprite.

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

fget

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

Flag degli sprite. fget(n) restituisce il byte del flag. fget(n, f) è vero quando il bit del flag f (0..7) è impostato.

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

fset

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

Imposta i flag dello sprite. fset(n, v) scrive l'intero byte. fset(n, f, v) imposta un bit.

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

setspr

```
setspr(n)
```

Seleziona il foglio sprite attivo. setspr() restituisce il foglio corrente. Stesso banco di settex.

```
setspr(1)
```

settex

```
settex(n)
```

Alias di setspr.

```
settex(0)
```

map

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

Disegna la mappa. La cella cx, cy della mappa viene disegnata sullo schermo sx, sy. cw, ch di 0 significa l'intera mappa. Il layer viene accettato e ignorato. tex è il foglio del tileset.

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

mget

`mget(cx, cy)`

ID dello sprite nella cella della mappa cx, cy.

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

mset

`mset(cx, cy, n)`

Scrive lo sprite n nella cella di mappa cx, cy.

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

setmap

`setmap(n)`

Seleziona la mappa n. setmap() restituisce la mappa corrente. La mappa 0 è una mappa reale.

```
setmap(1)
```

Memoria

peek

`peek(addr)`

Legge un byte della finestra da 64 KB. La finestra è lo schermo attivo, poi i pixel dello sprite, quindi le celle della mappa quando lo schermo rientra in 65280 byte. Gli schermi più grandi vengono paginati con poke(0x5F40, page). Gli indirizzi non mappati restituiscono 0. Scritto anche @addr.

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

poke

`poke(addr, ...)`

Scrive i byte all'indirizzo addr. Gli argomenti aggiuntivi scrivono i byte successivi. Gli indirizzi non mappati vengono ignorati. Vengono scritti anche $addr e val. Toccare lo schermo viene conteggiato come scrittura di un pixel.

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

peek2

`peek2(addr)`

Valore senza segno a 16 bit, little-endian, all'indirizzo.

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

poke2

`poke2(addr, val)`

Scrive un valore senza segno a 16 bit, in formato little-endian.

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

peek4

```
peek4(addr)
```

Valore senza segno a 32 bit, little-endian.

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

poke4

```
poke4(addr, val)
```

Scrive un valore senza segno a 32 bit, in formato little-endian.

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

memcpy

```
memcpy(dst, src, len)
```

Copia len byte all'interno della stessa finestra di 64 KB.

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

memset

```
memset(dest, val, len)
```

Riempie len byte a partire da dest con val.

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

Matematica

abs

`abs(x)`

Valore assoluto.

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

flr

`flr(x)`

Arrotonda verso meno infinito.

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

ceil

`ceil(x)`

Arrotonda verso più infinito.

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

max

`max(a, b)`

Il maggiore dei due valori.

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

min

`min(a, b)`

Il minore dei due valori.

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

mid

`mid(a, b, c)`

Valore intermedio tra i tre. Utile come morsetto.

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

sgn

`sgn(x)`

Segno di x: -1, 0 o 1.

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

sqrt

`sqrt(x)`

Radice quadrata. Un argomento negativo non è un numero.

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

sin

`sin(x)`

Seno. x è espresso in: 0, 0,25, 0,5, 0,75 giri intorno al cerchio una volta. Non radianti.

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

cos

`cos(x)`

Il coseno, a sua volta, segue la stessa convenzione del seno.

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

atan2

```
atan2(dx, dy)
```

Angolo del vettore dx, dy, che varia da 0 a 1.

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

rnd

```
rnd([x])
```

Numero casuale in [0, x). Il valore predefinito di x è 1, quindi rnd() è 0..1. Per un dado, flr(rnd(6))+1.

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

srand

```
srand(seed)
```

Imposta i parametri iniziali del generatore di numeri casuali in modo che una successiva sequenza rnd possa essere ripetuta.

```
srand(1)
```

band

```
band(a, b)
```

AND bit a bit. Scritto anche a e b.

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

bor

`bor(a, b)`

OR bit a bit. Scritto anche a | b.

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

bxor

`bxor(a, b)`

XOR bit a bit. Scritto anche a ^^ b.

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

bnot

`bnot(a)`

Bitwise NOT. Scritto anche ~a.

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

shl

`shl(a, n)`

Sposta a sinistra. Scritto anche come << n.

shr

`shr(a, n)`

Spostamento aritmetico a destra. Scritto anche come >> n.

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

lshr

```
lshr(a, n)
```

Spostamento logico a destra. Scritto anche come >>> n.

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

rotl

```
rotl(a, n)
```

Ruota i bit a sinistra. Scritto anche come <<> n.

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

rotr

```
rotr(a, n)
```

Ruota i bit a destra. Scritto anche come >>< n.

Tabelle e valori

add

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

Aggiunge v alla tabella t. Se viene specificato i, inserisce il valore all'indice corrispondente (basato su 1).

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

del

```
del(t, v)
```

Rimuove la prima voce uguale a v.

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

deli

```
deli(t, [i])
```

Rimuove l'indice i e restituisce quel valore. Senza i, rimuove l'ultima voce.

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

all

```
all(t)
```

Iteratore sulla parte dell'array, in ordine. I buchi interrompono la camminata come termina un array con indice base 1.

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

foreach

```
foreach(t, fn)
```

Chiama la funzione fn(v) per ogni elemento dell'array.

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

pairs

```
pairs(t)
```

Iteratore su ogni chiave e valore, inclusi i campi denominati.

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

ipairs

```
ipairs(t)
```

Iteratore sulla parte dell'array con indice basato su 1.

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

next

```
next(t, [k])
```

Restituisce la chiave e il valore successivi a k. next(t) inizia dall'inizio. Utilizzato dalle coppie.

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

count

```
count(t, [v])
```

Lunghezza della parte array. Se viene specificato v, conta quanti elementi sono uguali a v.

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

pack

```
pack(...)
```

Inserisce gli argomenti in una nuova tabella.

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

unpack

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

Restituisce t[i] fino a t[j] come valori separati. I valori predefiniti coprono l'intera parte dell'array.

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

select

```
select(i, ...)
```

select("#", ...) restituisce il numero di argomenti successivi. select(i, ...) restituisce gli argomenti a partire da i.

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

assert

```
assert(c, [msg])
```

Si arresta con un errore quando c è falso o nil. msg sostituisce il testo predefinito.

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

pcall

```
pcall(f, ...)
```

Chiama la funzione f. In caso di successo restituisce true e i risultati. In caso di errore restituisce false e il testo dell'errore.

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

Corde

chr

`chr(n)`

Un carattere del codice byte 0-255.

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

ord

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

Codici byte di s dall'indice i a j. Gli indici sono basati su 1. Il valore predefinito è il primo carattere.

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

sub

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

Sottostringa da i a j, con indice base 1. Ometti j per prendere la parte finale.

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

split

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

Divide s in una tabella. sep è impostato di default su virgola. Un numero sep rappresenta la larghezza del blocco. conv converte i pezzi con tonum quando è impostato su true.

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

tonum

```
tonum(s, [base])
```

Analizza un numero. La base può essere 2..36. Restituisce nil se il testo non è un numero.

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

tostr

```
tostr(v, [digits])
```

Converte un valore in testo. digits imposta il numero di cifre decimali per i numeri.

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

Audio

sfx

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

Riproduce il pattern n e restituisce il canale. Omettendo ch, o passando -1, viene scelto il primo canale libero. ch -2 interrompe il pattern n su ogni canale. n -1 interrompe un canale, o tutti i canali se ch viene omesso. n -2 consente a un pattern in loop di terminare invece di ripetersi. sfx() senza argomenti restituisce -1.

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

sfxget

```
sfxget(n, step)
```

Restituisce una tabella di un passo del pattern: [1] altezza, [2] forma d'onda, [3] volume, [4] effetto. L'altezza è 0..63 (da Do-0 a Re#5). La forma d'onda è 0..15. Il volume è 0..7 e 0 è silenzioso. L'effetto è 0..7.

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

sfxset

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

Scrive un passaggio del modello. Tutti e sei gli argomenti sono obbligatori.

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

sfxhdr

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

Intestazione del pattern. sfxhdr(n) restituisce la tabella di intestazione. sfxhdr(n, key, val) imposta un campo: speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps. I numeri da 0 a 10 selezionano gli stessi campi in quest'ordine. speed è TIC, tick per step. Più alto è, più lento è. 15 è 120 battiti al minuto. 16 bit accetta 1..255. I sintetizzatori più ristretti si adattano al loro elenco TIC.

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

sfxstat

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

ID del pattern in riproduzione sul canale ch, oppure -1. Il canale predefinito è 0.

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

note

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

Riproduce una nota senza uno schema. Intonazione 0..63, forma d'onda 0..15, volume 0..7, effetto 0..7. I tick indicano la durata, con la stessa frequenza di 120 Hz di TIC. Omettendo i tick, la durata è di 15, un passo predefinito. L'effetto viene applicato per tutta la durata, dopodiché il canale si silenzia.

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

music

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

Avvia il brano n dall'inizio. music(-1) lo interrompe. Chiamare music con l'ID di un brano già in riproduzione lo lascia in esecuzione. music() restituisce l'ID del brano in riproduzione, oppure -1. fade è espresso in millisecondi: il brano entra o esce gradualmente quando n è -1. I bit della maschera impostati rappresentano i canali utilizzati per il brano. 0 significa tutti e otto. Gli altri canali rimangono liberi per gli effetti sonori.

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

musicstat

```
musicstat()
```

Tabella del brano in riproduzione. song e row sono l'ID del brano e la riga, oppure -1 quando non è in riproduzione nulla. step e pos hanno otto numeri. step[1] è il canale 0, step[8] è il canale 7. step è l'indice della nota, oppure -1 quando quel canale è silenzioso. pos indica di quanto è andata quella nota, da 0 a 1. tick indica quanti tick ha già riprodotto questa riga. played indica quante righe sono iniziate da music(), inclusa quella in riproduzione.

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

songget

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

L'ID del pattern è memorizzato in una cella della canzone. ch è 0..7. 255 significa che la cella è vuota. Questa è la chiamata live; il testo del suggerimento del codice mostra ancora un ordine degli argomenti precedente.

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

songset

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

Scrive un ID di pattern in una cella di una canzone. Usa 255 per cancellare la cella.

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

stat

```
stat(n)
```

Solo audio. 16..23 è l'ID dell'effetto sonoro sul canale n-16. 24..31 è il pattern musicale sul canale n-24. Qualsiasi altro n restituisce 0. La temporizzazione dei frame e la RAM sono informazioni sulle prestazioni, non statistiche.

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

Tabelle dati

dget

```
dget(t, row, col)
```

Legge una cella della tabella dati t dal pannello DATI. Riga e colonna iniziano da 0.

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

dset

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

Scrive in una cella. Utilizzare una stringa per una colonna di testo.

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

dstr

`dstr(t, row, col)`

La cella come testo. I numeri vengono convertiti.

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

drows

`drows(t)`

Quante righe ha la tabella t?

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

dcols

`dcols(t)`

Quante colonne ha la tabella t?

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

Salvataggi e obiettivi

save

`save(k, v)`

Memorizza v sotto la chiave k nel file di salvataggio. v può essere un numero, una stringa o una tabella di questi due elementi. Il file è compresso.

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

load

`load(k)`

Legge la chiave k. Restituisce nil se la chiave non è presente.

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

delsave

`delsave(k)`

Elimina il tasto k dal salvataggio.

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

saveslot

`saveslot([name])`

Seleziona il file di salvataggio all'interno della cartella dell'autore. Se non viene specificato un nome, lo slot corrisponde al titolo del gioco. Restituisce 1 se il file esiste già.

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

achieve

`achieve(i)`

Sblocca l'obiettivo i, da 0 a 15. Memorizzato localmente. Anche una build di Steam imposta ACH_i. Una sessione del catalogo lo segnala all'account del giocatore.

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

achieved

```
achieved(i)
```

1 quando l'obiettivo i è già sbloccato.

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

achdef

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

Testo Toast per il risultato i, mostrato per 15 secondi dal basso dello schermo. spr è una tessera opzionale 8×8, tex il suo foglio.

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

achclear

```
achclear(i)
```

Cancella l'obiettivo i nel file locale. Un ACH_i di Steam già impostato rimane tale.

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

Rete

whoami

```
whoami()
```

Il tuo ID giocatore, da 0 a 15, è valido quando la sessione è attiva. Offline è -1.

```
id = whoami()
```

nhost

`nhost()`

1 se questa macchina è l'host, altrimenti 0.

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

nplayers

`nplayers()`

Quanti ID attivi esistono tra 0 e 15?

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

nlisten

`nlisten()`

Avvia l'hosting UDP e restituisce l'indirizzo LAN come ip:porta.

```
addr = nlisten()
```

njoin

`njoin(addr)`

Si connette tramite IP o IP:porta.

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

nleave

`nleave()`

Abbandona la sessione. I peer vedono _mp_leave con il motivo 0.

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

nsend

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

Invia un messaggio con perdita di dati. Il nome è l'evento. Seguono fino a quattro valori aggiuntivi. p da 0 a 15 invia un messaggio privato a quel peer; omettendo p, il messaggio viene inviato a tutti.

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

ncall

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

Stessa forma di send, ma il messaggio viene ritrasmesso finché non viene confermato. Tre errori consecutivi interrompono la sessione.

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

nlan

```
nlan()
```

Indirizzo IP e porta LAN correnti dopo nlisten.

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

nstun

```
nstun()
```

Il tuo indirizzo IP pubblico e la porta forniti da STUN. Un ospite su un'altra rete può tentare di connettersi.

```
pub = nstun()
```

setturn

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

TURN funge da fallback quando il percorso diretto fallisce. Il metodo setturn() con argomenti vuoti lo disattiva. L'indirizzo è host:porta.

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

nlist

`nlist(game)`

Tabella delle sessioni aperte per quel nome di gioco. Ogni voce contiene indirizzo, giocatori e nome.

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

nannounce

`nannounce(game)`

Heartbeat in modo che questo host appaia nella lista nlist per la partita. Avvialo con un timer durante l'hosting.

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

nkb

`nkb([on])`

Campo indirizzo. nkb(1) lo apre, nkb(0) lo nasconde, nkb() indica se è aperto. Su e giù cambiano un carattere, da 0 a 9, oppure un punto o due punti. A lo scrive. X lo cancella. La tastiera non digita al suo interno.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

Il testo nel campo indirizzo, quindi il carattere su e giù cambiano. nkbstr("") cancella il testo.

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

_mp

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

Scrivi questa funzione. Viene eseguita quando arriva un messaggio. `from` è l'ID del mittente. `name` è l'evento. Seguono fino a quattro valori.

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

_mp_join

`_mp_join(p)`

Scrivi questa funzione. Un peer ha ricevuto l'ID p. Si attiva anche per te; whoami() è quell'ID.

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

_mp_leave

`_mp_leave(p, why)`

Scrivi questa funzione. Peer p lasciato. Perché 0 è un'uscita normale. Perché 1 è una connessione persa.

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

_mp_update

`_mp_update()`

Scrivi questa funzione. Viene eseguita ogni 15 frame di aggiornamento, circa 4 Hz. Un buon punto per inviare le posizioni.

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

_mp_fail

`_mp_fail()`

Scrivi questa funzione. La sessione è interrotta: l'host è scomparso oppure ncall ha fallito tre volte.

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

Coroutine

cocreate

`cocreate(f)`

Crea una coroutine dalla funzione f. Non viene eseguita fino a quando non viene richiamata la funzione coresume.

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

coresume

`coresume(c, ...)`

Esegue la coroutine c finché non restituisce un valore. In caso di successo: restituisce true e i valori restituiti. In caso di errore: restituisce false e l'errore.

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

costatus

`costatus(c)`

"sospeso", "in esecuzione" o "morto".

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

yield

`yield(...)`

Restituisce il controllo a chi ha chiamato coresume. I valori diventano il risultato di quel coresume.

```
yield(1)
```

Misura

info

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

Disegna l'HUD in tempo reale in _draw: fps (media su 1 secondo), upd, drw, px, max, ram, peak, ramc e net quando una sessione è attiva. info() utilizza il cursore di stampa. info(col) imposta solo il colore. info(x, y) o info(x, y, col) lo posizionano. I numeri usano B, k o m in modo che si adattino a uno schermo piccolo. La chiamata stessa non viene conteggiata in px.

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

showstats

```
showstats([on])
```

Stesso HUD di info, disegnato automaticamente a 0, 0 dopo _draw, ignorando telecamera e clip. showstats(0) lo nasconde. showstats(1) o showstats() lo mostrano.

```
showstats(1)
```

perf

```
perf(n)
```

Un numero HUD, così puoi disegnarlo tu stesso. 0 fps, 1 upd, 2 drw, 3 px, 4 max (frame più pesante in questo secondo), 5 ram, 6 ramc (ram più la cartuccia di gioco compressa), 7 ram di picco dall'avvio della cartuccia di gioco.

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