Manual do PIX37

Editor, depois código. Cópia offline do livro em pix37.party/docs.

Editor

O PIX37 DEV é o editor. Nele, desenha, escreve e ouve o jogo, e depois exporta um cartucho ou uma versão independente. Também pode reproduzir um cartucho e aceder ao PIX37. PARTTY para descarregar cartuchos e utilizar o catálogo. O PIX37 DEV EDUCATIONAL é o mesmo editor, mas é gratuito. Não tem versões, não permite exportar tinteiros e não dá acesso ao PIX37.PARTY. O PIX37 Console é o leitor. Reproduz cartuchos `.p37` e permite aceder ao PIX37.PARTY para descarregar cartuchos e navegar pelo catálogo.

Desenhos e cartuchos de jogos

Um ficheiro `.pix37` é o projeto: separadores de código fonte, folhas de sprites, mapas, sons e tabelas de dados. Um ficheiro `.p37` é o cartucho do jogo que o jogador executa: bytecode comprimido.

Os ficheiros de gravação e as conquistas estão na pasta do autor, ao lado dos dados da aplicação. No Windows, o caminho é algo como `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s` para um ficheiro de gravação e `C:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.ach` para as conquistas. No Linux, os mesmos ficheiros são o `~/.local/share/PIX37/Mira/Pong.p37s` e o `~/.local/share/PIX37/Mira/Pong.ach`. Ada é a conta no computador. Mira é o nome do autor, e esse nome corresponde à pasta. Pong é o slot de gravação, que por defeito é o título do jogo. O ficheiro `save` cria o ficheiro `.p37s`. A primeira conquista desbloqueada cria o ficheiro `.ach`. Se os ficheiros já lá estiverem, o programa lê-os quando o cartucho do jogo é iniciado.

Menu de programador

A tecla Escape abre o menu de programador. Premir Escape durante a execução de um jogo retorna ao editor. Confirme com Enter ou com a tecla A (V por defeito). Voltar é feito com a tecla B (C por defeito).

Sem nenhum projeto carregado, o menu é: CRIAR PROJETO, CARREGAR PROJETO, COMPILAR, CARREGAR cartucho de jogo, LIGAR AO PIX37.PARTY, CONFIGURAÇÕES.

Com um projeto carregado, as opções são: GUARDAR PROJETO, CARREGAR PROJETO, CRIAR PROJETO, CARACTERÍSTICAS, COMpilar, CARREGAR cartucho de jogo, LIGAR AO PIX37.PARTY, DEFINIÇÕES.

Ctrl+S guarda o projeto aberto. Ctrl+R executa-o.

Criar um projeto

A opção CRIAR PROJETO solicita o cartucho do jogo antes de começar a criar o projeto.

- **VÍDEO** — 1-BIT (2 cores), 2-BIT (4), 4-BIT (16), 8-BIT (256).
- **ÁUDIO** — profundidade de amostragem do sintetizador: 16, 8, 4, 2 ou 1 bit.
- **ECRÃ** — as predefinições incluem 64×64, 128×128, 160×90, 256×256 e 320×180. O tamanho PERSONALIZADO permite-lhe escolher qualquer tamanho de 1×1 a 512×512.
- **PALETAS** — bancos de cores. O primeiro banco começa com o conjunto de 16 cores abaixo. CARREGAR PALETA aplica um conjunto integrado: PICO-8 16, PICO-8 32, Game Boy, 1-bit, CGA, Sweetie-16 e uma lista maior extraída do Lospec (Resurrect 64, Apollo, Endesga 32 e 64, CC-29, SLSO8, Pear36, Oil 6, Duel, AAP-64, Lost Century e outros). Também pode carregar um ficheiro de paleta ou carregar o seu próprio PNG e extrair a paleta dessa imagem.



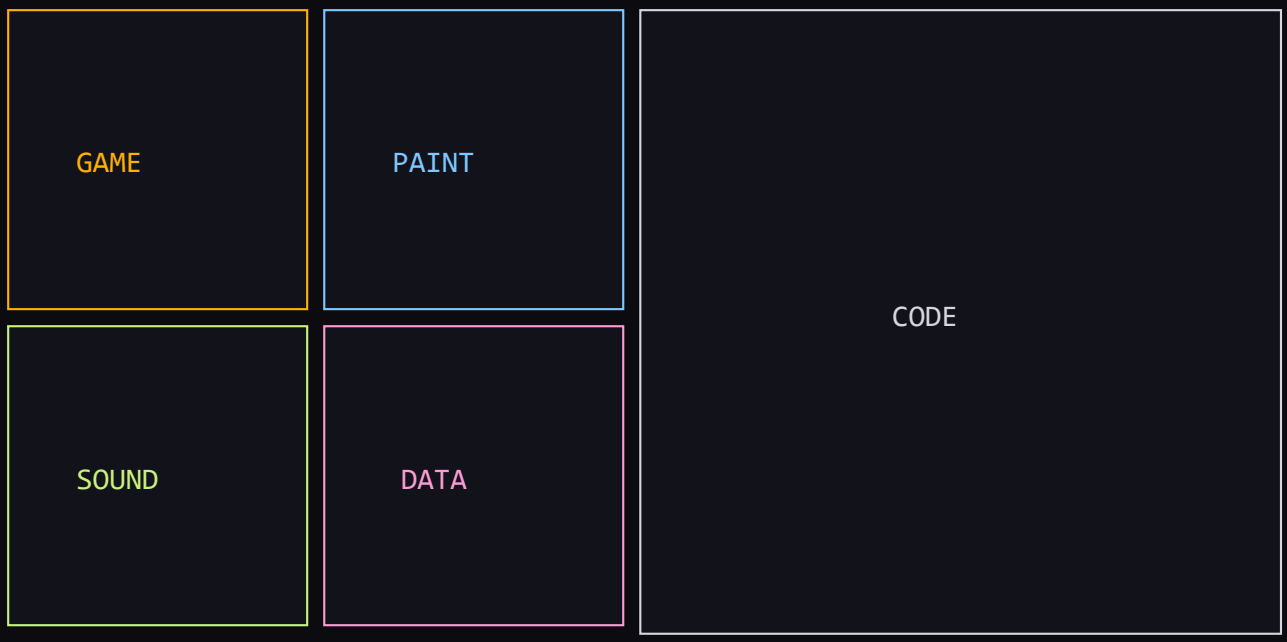
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

```
lette — black, navy, purple, green, brown, gray, silver, white, red, orange, yellow, lime, blue, indigo, p
```

Índices de cor utilizados por `cls`, `pset`, `spr` e os restantes. Pode repintar qualquer índice no editor de paletas.

Os cinco painéis

O editor é composto por JOGO, PINTURA, CÓDIGO, SOM e DADOS, por esta ordem. O mapa fica dentro de PINTURA. Os efeitos sonoros e as músicas estão dentro de SOM. Clique num painel para trabalhar nele. Quando o CÓDIGO está focado, o SOM e os DADOS ignoram as teclas de letras, pelo que a escrita não toca piano ao mesmo tempo.



Layout completo, escrito `1:1:4:1:1`. Cada número indica a percentagem da linha que o painel ocupa: JOGO, PINTURA, CÓDIGO, SOM, DADOS. Um ecrã pequeno fica centralizado numa célula de 128×128. Um ecrã maior que 128 px alarga a coluna JOGO.

DEFINIÇÕES → GLOBAL → LAYOUT alterna as definições:

Código	O que vê
`1:1:4:1:1`	Tudo. JOGO e SOM empilhados, PINTURA e DADOS empilhados, CÓDIGO à direita.
`1:1:4:1:1 2X`	A mesma disposição. O código é desenhado com o dobro da densidade, pelo que cabe o dobro do texto.
`1:0:1:0:1`	JOGO, DADOS, CÓDIGO.
`1:0:1:0:1 2X`	A mesma configuração, mas com o código mais denso.
`0:1:1:1:1`	Sem antevisão do jogo. PINTURA, CÓDIGO, SOM, DADOS.
`0:1:1:1:1 2X`	A mesma configuração, mas com o código mais denso.
`0:1:4:1:2`	Estúdio. CÓDIGO amplo, uma gama de DADOS mais elevada, PINTURA e SOM.
`0:1:4:1:2 2X`	A mesma configuração, mas com o código mais denso.

`0:0:4:0:4`	Apenas CÓDIGO e DADOS, ambos grandes.
`0:0:4:0:4 2X`	A mesma configuração, mas com o código mais denso.
`0:1:4:1:0`	TINTA, SOM e um amplo CÓDIGO.
`0:1:4:1:0 2X`	A mesma configuração, mas com o código mais denso.
`0:4:0:0:0`	Apenas tinta, 256×256, um pixel da folha para um pixel do ecrã.

Uma linha `2X` mantém os mesmos painéis. O painel CODE é desenhado com o dobro da resolução e redimensionado para caber na mesma célula, de modo a que o texto, as tabulações e o espaçamento entre células tenham metade da altura e o dobro do conteúdo caiba no ecrã.

JOGO

A pré-visualização do cartucho do jogo em execução. Clique no painel JOGO ou prima Ctrl+R para executar a pré-visualização. DEFINIÇÕES → PAINEL DE JOGO é onde o painel está configurado:

- **Pré-visualização** — cor ociosa, contínua ou recarregamento automático ao editar.
- **Reproduzir numa nova janela** — o jogo será aberto ao lado do editor.
- **Visualizar** — pré-visualização em RGB888 ou RGB565.
- **GAME 64** — um ecrã de 64×64 é centrado 1:1 na célula ou redimensionado 2× para o preencher.
- **Shaders** — CRT e filtros no jogo, separados dos filtros no editor.
- **Cor** — fundo do painel.

PINTAR

Folhas de sprites e, no lado do mapa do mesmo painel, mapas de tiles. As folhas podem ser maiores que a tela. Os tamanhos de pré-visualização dos sprites são 8, 16, 32, 64 e 128 (128 apenas quando a folha tiver pelo menos 128 px). Percorra a roda do rato sobre a pré-visualização do sprite para a ampliar. Os tamanhos dos pincéis são 1, 2, 3 e 4.

Chave	Ferramenta
B	Lápis
M	Seleção retangular
Q	Laço
S	Selo

G	Balde
Eu	Conta-gotas
V	Mover
L / Shift+L	Linha/curva
U / Shift+U	Retângulo oco / elipse oca
F / Shift+F	Retângulo preenchido / elipse preenchida
D	Hether
[ ]	Pincel mais pequeno/maior
Ctrl+Z / Ctrl+Y	Desfazer/refazer. Ctrl+Shift+Z também desfaz a ação.
Ctrl+C / Ctrl+X / Ctrl+V	Copiar, recortar, colar
Ctrl+A	Selecione o ecrã

O mapa utiliza a mesma ideia, mas com menos ferramentas: lápis (B), seleção retangular (M), carimbo (S), mover (V) e Ctrl+Z, Ctrl+Y, Ctrl+C, Ctrl+V. Mantenha a tecla Alt premida para ampliar a textura do mapa. O tamanho do pincel no mapa permanece o mesmo que o das ferramentas de pintura.

Cada sprite possui oito bits de sinalização. O Paint define-os no sprite; `fget` e `fset` lêem-nos no jogo.

CONFIGURAÇÕES → EDITOR DE PINTURA é onde este painel está configurado:

- **Palete** — qual o banco selecionado, além de carregar e guardar.
- **Pré-visualização** — RGB888 ou RGB565.
- **Cursor de pincel** — o ponteiro que utiliza enquanto pinta.
- **Ajuste de pixéis** — como o pincel se ajusta aos pixéis.
- **Grande arbusto** — um arbusto grande quebra a partir do canto superior esquerdo, ou de toda a sua extensão.
- **Pele de cebola** — um fantasma de outros quadros, e para que direção este fantasma está virado.
- **Zoom do mapa** — pixéis reais ou zoom livre.
- **Cores** — as cores da interface de pintura.

CÓDIGO

As abas são ficheiros separados dentro do projeto. Clique no nome de um separador para o renomear; prima Enter para confirmar. Várias abas que definem `_init`, `_update` ou `_draw` são unidas na ordem de tabulação: a primeira aba é executada em primeiro lugar e a última, em último lugar. O código que não está dentro de um

gancho passa a fazer parte de `_init`. Um cartucho de jogo sem qualquer gancho é tratado como um único ficheiro `_draw`.

Chave	Ação
Flechas	Mova o cursor. A tecla Shift amplia a seleção.
Ctrl+Esquerda / Direita	Por palavra.
Início / Fim, Ctrl+Início / Fim	Início ou fim da linha, ou da tabulação.
Aba	Insira dois espaços. Na lista de conclusão, aceite o nome.
Entrar	Nova linha, com recuo semelhante ao da anterior. Aceita uma conclusão.
Ctrl+C / X / V	Copiar, recortar, colar.
Ctrl+Z / Ctrl+Y	Desfazer/refazer.
Ctrl+A	Selecione o separador.
Ctrl+D	Duplique a linha ou a seleção para baixo.
Ctrl+/	Comente ou remova comentários com `--`.
Ctrl+F	Encontre. F3 salta para o próximo resultado. Esc fecha a barra.
Ctrl+G	Vá para um número de linha.
Escapar	Primeiro, fecha a pesquisa, a opção "Ir para" ou a lista de dicas. Uma segunda tecla Escape sai do editor e acede ao menu de programador.

As dicas aparecem enquanto introduz o nome de uma função. Prima Enter ou Tab para introduzir o nome e `(` se a opção COMPLETE () estiver ativada nas definições. DEFINIÇÕES → EDITOR DE CÓDIGO:

- **Pesquisar** — Ctrl+F no separador ativo ou em todos os separadores.
- **Texto alto** — recorte os glifos para 5 px ou mostre o glifo completo.
- **Tamanho da string** — tamanho da fonte dos literais de string.
- **Fonte de comentário** — 3×5, ou a fonte Fusion do idioma da interface de utilizador atual.
- **Complete (** — se a conclusão digita o parêntesis.
- **Dicas de código** — ativadas ou desativadas.
- **Idioma das dicas** — o idioma do menu ou o inglês.
- **Espaço do cursor** — deslocamento extra em linhas vazias.
- **Margem** — números de linha.
- **Cor** — fundo, comentários, tabulações, números, operadores, palavras-chave, cadeias de caracteres, funções do mecanismo, suas funções, variáveis, parêntesis rectos, acento circunflexo, selecção, linha de

erro, aviso.

SOM

O painel tem duas abas. SFX representa um padrão: uma pequena sequência de notas. SOUND representa uma música: coloca estes padrões em oito canais. TIC indica quantos ticks espera um passo. O relógio marca 120 ticks por segundo, e um passo é uma semicolcheia, pelo que 15 ticks equivalem a 120 batimentos por minuto. Um TIC mais elevado indica um padrão mais lento. Dois padrões permanecem sincronizados quando um TIC divide o outro: 15 e 30 permanecem, 15 e 8 não.

O piano toca quando uma nota é selecionada com o rato. As teclas brancas são ZXCVBNM (CDEFGAB). As teclas pretas são SDGHJ (C# D# F# G# A#). Coloque o cursor sobre qualquer valor e introduza um número, ou posicione o cursor sobre o mesmo e faça rolar a roda do rato. Os números de altura no código variam de 0 (C-0) a 63 (D#5).

Num padrão, LOOP e LEN são os mesmos dois números. OUT acima de 0 repete o ciclo de IN até OUT, e o título é LOOP. OUT em 0 e IN acima de 0 é LEN: esta quantidade de notas é tocada desde o início, pelo que o padrão termina, e o título é LEN. Ambos em 0 tocam cada passo.

A quantidade de sintetizadores que pode editar depende da profundidade de bits escolhida para o cartucho do jogo. CONFIGURAÇÕES → EDITOR DE EFEITOS SONOROS define esta profundidade, se uma nota avança automaticamente (desativado, cadeia de notas ou cadeia de campos), se a barra de espaço reinicia a reprodução ou alterna entre iniciar e parar, e as cores do editor.

16 bits

Cada passo possui uma nota, um volume, uma onda e um efeito. O volume varia de 0 a 7. 0 é silêncio.

Ondas:

- **0 triângulo**
- **1 serra inclinada**
- **2 serras**
- **3 quadrados**
- **4 pulsos** — um pulso estreito
- **5 órgãos**
- **6 ruídos**
- **7 faseadores**
- **8 PWM** — um quadrado cuja largura se desloca
- **9 pulsos 25%**
- **10 seno**
- **11 ruído metálico**
- **12 sino FM**

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

Efeitos numa única nota:

- **0 nenhum**
- **1º deslizar** — o tom e o volume deslizam para a nota seguinte ao longo deste degrau.
- **2 vibrato** — oscilações de tom
- **3ª queda** — o tom cai para o silêncio, o habitual pontapé
- **4 fade in**
- **5. Desvanecer**
- **6 arpejos** — percorrem esta nota e as três seguintes, uma de 8 em 8 tempos. No TIC 8 ou inferior, de 4 em 4 tempos.
- **7 arpejos rápidos** — as mesmas quatro notas, de 4 em 4 tempos, ou de 2 em 2 quando o TIC é 8 ou menos.

O cabeçalho do padrão, por cima das notas:

- **TIC** — tiques por passo, de 1 a 255. 15 corresponde a 120 batimentos por minuto. Introduza o número ou aumente em incrementos de um.
- **LOOP IN / OUT** — com OUT acima de 0, o padrão repete-se de IN para OUT. Deixe OUT a 0 e o título passa a ser LEN: IN indica quantas notas são tocadas e 0 significa o padrão completo.
- **RUÍDO** — todo o padrão se transforma em ruído.
- **CRUSH** — uma variante vibrante de cada onda.
- **DTUNE** — 0 desligado, 1 um coro ligeiro, 2 um coro forte.
- **RVERB** — reverberação: 0 desligado, 1 curto, 2 longo.
- **DAMPN** — um filtro passa-baixo: 0 desligado, 1 leve, 2 forte.
- **NÍVEL** — volume de todo o padrão, de 0 a 7.
- **SWING** — arrastar os pés. Os passos ímpares esperam um pouco mais, os passos pares um pouco menos.
- **PASSOS** — quantos passos utiliza o padrão: 32, 24, 16 ou 12.

8 bits

Um passo possui uma nota e uma onda. As quatro ondas são quadrada, triangular, dente de serra e ruído. O cabeçalho contém TIC, LOOP IN/OUT, FX e LEVEL. O TIC pode ser um dos seguintes valores: 1, 2, 3, 4, 5, 6, 8, 10, 12, 15, 16, 20, 24, 30, 40 ou 60. O FX define um efeito para todo o padrão: 0 (nenhum), 1 (vibrato), 2 (arpejo) ou 3 (fade out). LEVEL varia de 0 a 3.

4 bits

Um passo é apenas uma nota. O cabeçalho contém TIC, LOOP (ligado ou desligado), WAVE (as mesmas quatro ondas) e OCT (a oitava em que as notas se encontram). TIC pode ser um dos seguintes valores: 4, 6, 8, 10, 12, 15, 20 ou 30.

2 bits

Um degrau pode ser vazio, um tambor, uma batida ou um esmagamento. O cabeçalho contém TIC e BASE. TIC pode ser 8, 15, 16 ou 30. BASE define a altura a que o tambor e a batida estão posicionados.

1 bit

Um passo é uma nota. TIC é de 8, 15, 16 ou 30, as mesmas quatro velocidades de 2 bits.

Músicas

O separador SOM é a música aberta. Oito colunas representam os canais. Cada célula contém o ID de um padrão. O número 255, se estiver vazio, significa que o canal está em silêncio nessa linha. A coluna X contém um comando: continuar, parar, iniciar ciclo ou terminar ciclo. Parar termina a música após a linha. Encerrar loop regressa ao último ponto de início do loop.

Uma carreira termina quando o padrão mais à esquerda com LOOP OUT a 0 termina. Os padrões mais à direita não aumentam o comprimento da carreira: um padrão mais longo é interrompido, um mais curto recomeça no seu primeiro passo até que a carreira termine. Um padrão com LOOP OUT acima de 0 forma um loop entre LOOP IN e LOOP OUT e não termina a carreira. Se todos os padrões da carreira formarem um loop, a carreira permanece inalterada.

Atalhos

Chave	Ação
Teclas de piano	Escreva uma nota sobre a etapa selecionada com o rato.
Para cima / Para baixo	Altere o valor sob o cursor.
Espaço	Reproduza os efeitos sonoros iniciais ou a música inicial da primeira linha. Inicie, pare ou reinicie nas definições do EDITOR DE EFEITOS SONOROS.
Ctrl+Z / Ctrl+Y	Desfazer/refazer.
Ctrl+C / Ctrl+X / Ctrl+V	Copie, recorte e cole a seleção.

DADOS

Tabelas nomeadas. Cada tabela tem linhas e colunas. No código, `dget`, `dset`, `dstr`, `drows` e `dcols` utilizam estas tabelas. A numeração das linhas e colunas no código começa em 0.

Tipos de coluna, apresentados em ciclo no cabeçalho:

- **booleano** — um bit.
- **2** — dois bits.
- **4** — quatro bits.
- **8** — um byte.
- **16** — 16 bits, para identificadores e números superiores.
- **spr** — um identificador de sprite.
- **str** — texto, desenhado com a fonte do sistema.
- **str8** e **str10** — o mesmo texto, desenhado mais alto.
- **f16** e **f32** — ponto flutuante de meia precisão e ponto flutuante completo.

Com o cursor posicionado no cabeçalho de uma coluna, Ctrl+scroll altera o tipo dessa coluna. Um scroll simples move o cursor ao longo das colunas ou para baixo nas linhas, dependendo da selecção feita em DEFINIÇÕES → EDITOR DE DADOS. Ctrl inverte esta direção, e é por isso que Ctrl+scroll num cabeçalho altera o tipo quando a roda de deslocamento está configurada para apresentar colunas.

As teclas de seta movem a célula. A tecla Enter edita a célula. CONFIGURAÇÕES → EDITOR DE DADOS:

- **Roda** — role as colunas ou role as linhas.
- **Cores** — da grelha, do cabeçalho e da seleção.

Ativos

A categoria ASSETS importa e exporta o fragmento que está aberto, e não todo o projeto. A categoria Sprites utiliza o sprite aberto no PAINT, na folha que está aberta. A categoria Mapa é o mapa que está aberto. A categoria SFX é a norma que está aberta. A categoria Música é a música que está aberta. A categoria Data é a tabela que está aberta. A categoria Code é a exceção: cada tabulação é escrita e cada tabulação é lida de volta.

Amabilidade	Ficheiro	O que move
Sprites	`.png`	O sprite aberto.
Mapa	`.p37map`	O mapa aberto.
SFX	`.p37sfx`	O padrão aberto.
Música	`.p37mus`	A música está aberta. Os números dos padrões permanecem como estão. Exporte os efeitos sonoros separadamente, se precisar deles.
Dados	`.p37dat`	A mesa aberta.

Código	`.lua`	Cada separador.

Configurações

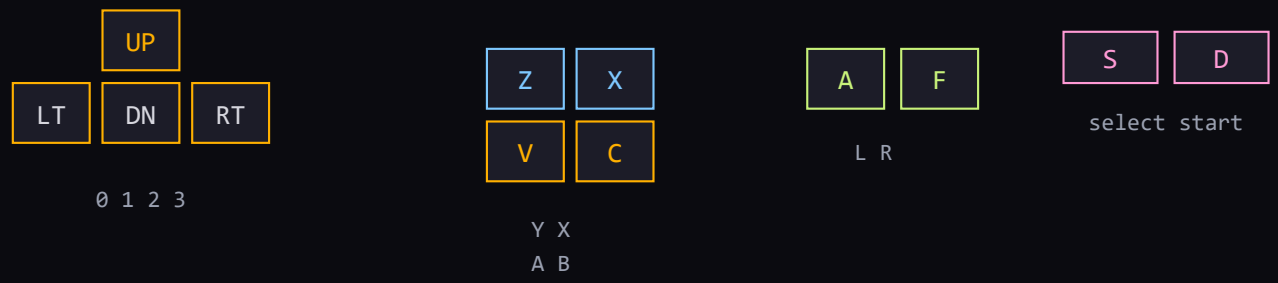
A lista raiz:

Página	O que armazena
PERFIS	Guarde e carregue todas as definições. Crie, renomeie, sobrescreva, apague. Copie ou cole o perfil como texto.
Menu	Cores da janela, teclas, menu e fundo opcional.
PAINEL DE JOGO	Pré-visualização e janela de reprodução.
EDITOR DE CÓDIGO	Destaque e dicas.
EDITOR DE PINTURA	Pincel, paleta, visualização do mapa.
EDITOR DE EFEITOS SONOROS	Sintetizador e entrada de notas.
EDITOR DE DADOS	Direção e cores da roda.
IDIOMA	Idioma do menu e das dicas.
LICENÇAS	Textos de licença, incluindo a fonte Fusion Pixel e o SDL3.

GLOBAL:

- **LAYOUT** — disposição dos painéis, acima.
- **VOLUME** — nível master, de 0 a 10.
- **CONFIGURAÇÕES DE TECLAS** — teclas para cada um dos 8 jogadores. Prima uma tecla para a selecionar. Direções, ABXYLR, seleccionar.
- **Apresentação** — saltar quadros ociosos ou desenhar todos os quadros.
- **Ecrã completo** — sem bordas.
- **Atualizar** — FPS da janela. Para a esquerda e para a direita, defina os valores para 60, 90, 120 e o máximo. Introduza um número até 1000. 0 significa sem limite.
- **Vsync** — aguarda a apresentação entre quadros.
- **Escala** — multiplicador de janela inteira.
- **Shaders** — CRT e filtros no editor, não no jogo.
- **Cor** — texto do menu e destaque.
- **Fundo** — um PNG opcional, recortado para 128 pixéis, localizado atrás do menu. A opção "Limpar" remove o fundo.
- **Cursor do editor** — formato de ponteiro.

- **Rato** — ponteiro flutuante ou alinhado com os pixéis.
- **Velocidade da interface do utilizador** — a rapidez com que uma direção premida se repete numa lista.
- **Explorador de** ficheiros — a lista de ficheiros começa na raiz do computador ou na pasta do ficheiro executável.



Teclado standard. A tecla A confirma nos menus. A tecla B volta. Num jogo, estas teclas correspondem aos botões 4 e 5. A tecla Select (S) corresponde ao botão 10, não sendo um atalho de menu.

Botão	Papel	Chave padrão
0	Esquerda	Seta para a esquerda
1	Certo	Seta para a direita
2	Para cima	Seta para cima
3	Para baixo	Seta para baixo
4	A, confirm	V
5	B, costas	C
6	X	X
7	Y	Z
8	R	F
9	L	UM
10	Selecionar	S
11	Iniciar	D

Construir

Escolha uma plataforma: Windows, Linux, Mac, Android, iOS ou Web.

- **Nome** — nome do ficheiro da compilação.
- **Autor** — no Windows e no Linux. Este é o nome da pasta onde os ficheiros são guardados.
- **ID da aplicação Steam** — no Windows e no Linux. Vazio significa que não há Steamworks. Um ID preenchido mapeia `achieve(0)`…`achieve(15)` para as conquistas Steam ACH_0…ACH_15.
- **ID** — identificador do pacote, quando solicitado pela plataforma.
- **Ocultar código** — em sobreposições para desktop e dispositivos móveis. Isto torna o `peek` do cartucho do jogo mais lento. Não é um bloqueio.
- **Ecrã inteiro** — o leitor exportado inicia em ecrã inteiro.
- **Shaders** — até três camadas integradas no jogador.
- **Transferir . p37** — Apenas na Web. Apresenta uma opção de transferência no menu de pausa.
- **Ícone** — um bloco de uma folha de sprites: folha, tamanho, x, y. Ou uma capa PNG que recorta.
- **Exportar** — selecione a pasta. O menu irá apresentar "COMPILAÇÃO GUARDADA" ou "COMPILAÇÃO FALHOU".

Crie uma versão com a internet ligada. O ecrã inicial pode consultar o servidor, anonimamente, para verificar se o hash desta versão é conhecido. Esta verificação existe por dois motivos. Ela protege-me, o autor do PIX37, de cópias não autorizadas da ferramenta que possam ser distribuídas. Também protege o criador do jogo. Pegar no jogo de outra pessoa e listá-lo como se fosse seu está a tornar-se comum no mercado das aplicações móveis, como a Play Store (Android) ou a App Store (iOS). O jogo ainda inicia. Se o dispositivo estiver offline, a introdução informa que o jogo está desligado. Esta mensagem serve também de alerta para o jogador de que a cópia pode não ter vindo de uma fonte legal. É uma pequena verificação para que, caso o seu jogo tenha sido roubado e vendido sem a sua permissão por outra pessoa, possa apresentar uma reclamação em lojas como a Play Store (Android), Steam ou itch.io. O servidor armazena a hora a que o hash foi registado pela primeira vez, ao lado do próprio hash, e não a altera posteriormente. A hora mais antiga corresponde à versão original. Um hash registado posteriormente corresponde a uma cópia posterior.

Quero essa verificação para proteger os programadores de jogos. O motor gráfico já faz muito do trabalho por eles, e esta é apenas uma das coisas que precisam de ser feitas. Entendo que isto possa ser controverso. Se a comunidade decidir que é melhor remover este micro-DRM, assim o farei.

Um cartucho de jogo antigo de um editor anterior pode recusar-se a arrancar após a alteração do idioma. Abra o projeto `.pix37` neste editor e exporte um novo ficheiro `.p37`.

Ligue-se ao PIX37.PARTY

A opção LIGAR AO PIX37. PARTTY está dentro da aplicação. Não existe página de login no browser. Entra com um bilhete de sessão Steam ou com o itch.io, usando o teclado e o rato. O serviço apresenta um nome de visualização único, de 2 a 16 caracteres. A chave de licença e o ID Steam não são mostrados a outros jogadores.

Após os termos de serviço, a secção principal é composta pelos Favoritos, Catálogo, Ranking, Jogadores e o seu perfil.

A licença de Programador é para o editor. Cria o jogo, publica-o e deixa estrelas. A licença de Jogador é para o jogador. Os corações são dados por pessoas que jogam num jogo de corridas.

- **Catálogo** — navegue pelos jogos. Os filtros abrangem o género (até três), o tamanho do ecrã, o tamanho do cartucho do jogo e se o carregamento inclui um projeto `.pix37`.
- **Favoritos** — jogos que marcou.
- **Envio** — requer uma licença de programador. Envia um ficheiro `.p37` e pode anexar o projeto `.pix37`. Um ficheiro `.p37` é um bytecode comprimido. É difícil de ler e pode ser propositadamente ofuscado. Um ficheiro `.pix37` é o ficheiro de projeto no qual realmente trabalha: o código-fonte, as imagens e o som, num formato que outra pessoa pode abrir e utilizar como referência. O envio permite que outros criadores estudem o jogo em vez de um cartucho comprimido. Título, autor, géneros, capa, ícone e nomes de conquistas acompanham o ficheiro. O número de RAM enviado corresponde ao pico atingido pelo cartucho do jogo.
- **Comentários e conquistas** — na página do jogo. O desbloqueio de uma conquista enquanto está ligado é armazenado nessa conta, bem como no computador.
- **Classificação** — desenvolvedores, ordenados por estrelas ou por corações. Cada linha mostra ambas as contagens. Abra um nome para ver os jogos deste autor, ordenados por corações.
- **Jogadores** — contas ordenadas por conquistas desbloqueadas.

Código

A linguagem é um pequeno script ao estilo Lua. Esta primeira parte começa com um projeto vazio, ensina a linguagem e depois mostra um breve exemplo de cada função.

Um primeiro programa

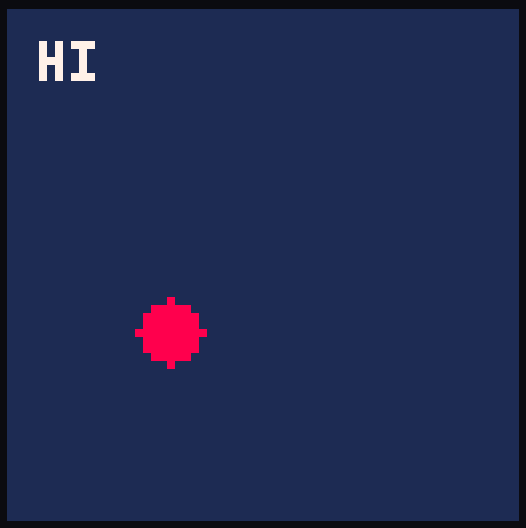
Crie um projeto em 64×64, com cores de 4 bits. Abra o CODE. Substitua a tabulação por:

```
_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 executa o jogo. As setas direcionais movem o círculo. A tecla Esc pára o jogo e regressa ao editor.



As mesmas três chamadas: `cls(1)` desenha a tela azul-marinho, `print` desenha HI e `circfill` desenha o círculo cor-de-rosa.

O que funciona e quando?

- `_init` é executado uma vez.
- `_update` funciona a 60 Hz. Coloque aqui o movimento e os botões.
- `_draw` pinta. Sem argumento de fps, pinta tão rápido quanto o computador e o monitor conseguem. `_draw(30)` limita a pintura a 30 fps e deixa a lógica a 60.
- `_set_fps(n)` altera este limite posteriormente. 0 o limpa.

Os nomes não são sensíveis a maiúsculas e minúsculas; o editor armazena-os em letras minúsculas. `function` e `fun` são a mesma palavra-chave.

Idioma

Noções básicas de programação

Um programa é uma lista de linhas que o cartucho do jogo executa. Cada cabeçalho mostra primeiro o formato da linha e, em seguida, um pequeno exemplo. Uma linha que começa por `--` é um comentário: uma nota para si. O jogo ignora-a.

Variáveis

Uma variável é um nome que armazena um valor. Você escolhe o nome. As linhas posteriores podem lê-lo e armazenar um novo valor com o mesmo nome.

O formato é `name = value`. `local name = value` mantém este nome dentro do bloco atual, de modo a que o resto do cartucho do jogo não o consiga ver. Sem `local`, o nome é global e todos os separadores podem vê-lo.

```
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 não há valor. `true` e `false` significam sim e não, respetivamente. Um nome que nunca definiu será apresentado como `nil`.

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

Tabelas

Uma tabela armazena vários valores sob o mesmo nome. Uma lista utiliza números. O primeiro item é 1, e não 0. Um registo utiliza nomes. Uma tabela pode conter outra tabela.

Formato da lista: `{ value, value }`. Formato do registo: `{ name = value, name = value }`. Uma lista de registos é uma tabela cujos itens são tabelas.

```
-- 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
```

Condição e corpo

Uma condição é uma pergunta com uma resposta sim ou não, como `hp > 0` ou `btnd(4)`. Um corpo é o grupo de linhas que são executadas quando se entra num bloco. `if` e todos os loops utilizam esta mesma estrutura: uma condição, seguida de um corpo e, por fim, `end`. `repeat` é finalizado com `until` em vez de `end`.

Se

`if` executa o seu corpo apenas quando a condição é verdadeira. `elseif` tenta outra condição. O `else` é executado quando todas as condições são falsas.

Forma: `if condition then`, depois o corpo, depois `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
```

Laços

Um loop repete um corpo. `while` verifica primeiro a condição e continua enquanto for verdadeira. `repeat` executa o corpo e depois verifica, de modo a que o corpo seja sempre executado pelo menos uma vez. `for` conta. O último número é incluído. Sem um terceiro número, o incremento é 1, pelo que, se o início já tiver passado o fim, o corpo não é executado. A contagem decrescente requer um incremento 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
```

Caminhando sobre uma mesa

`all` visita cada item de uma lista, a partir do primeiro. `pairs` visita cada nome e o valor armazenado sob esse 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
```

Funções

Uma função é uma receita nomeada. O formato é `function name(arguments)`, seguido de um corpo e, por fim, `end`. Chama-a escrevendo o nome e parênteses. `return` devolve um valor, podendo devolver mais do que um. Os argumentos são variáveis locais desta função.

```
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` é uma função que pode ver as variáveis locais à sua volta. Trata-se de um fecho. Uma tabela pode conter um fecho, e chama-se com um ponto.

```
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
```

Sim, não e escolher.

`not` inverte uma condição. `and` é verdadeiro apenas quando ambas as condições são verdadeiras. `or` é verdadeiro quando qualquer uma das condições é verdadeira. `a and b or c` escolhe `b` quando `a` é verdadeiro, caso contrário escolhe `c`. `cond ? a : b` representa a mesma ideia numa única linha: `a` quando a condição é verdadeira, `b` quando não é.

```
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)
```

Matemática e texto

`+ - * /` representa a adição, subtração, multiplicação e divisão. `//` divide mantendo o número inteiro. `%` representa o resto da divisão. `^` representa a potência. `..` une dois textos. `#t` indica a quantidade de artigos numa lista. As comparações são representadas por `==` (igual), `~=` ou `!=` (diferente) e `< > <= >=`. `x += 1` soma 1 a x. A mesma fórmula se aplica ao `-= *= /= ^= %=`.

```
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
```

Onde vive um morador local

`local` dura até que `end` feche o seu bloco. `if` / `end` é um bloco independente. Pode mencionar uma variável local acima da linha que define o valor, mas até que essa linha seja executada, o 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
```

Saltando

`::label::` marca uma linha. `goto label` salta para lá, mas apenas dentro da mesma função.

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

Memória, em síntese

Os endereços `peek` e `poke` visualizam os pixéis ativos através de um endereço de 16 bits. Não guardam uma segunda cópia do cartucho do jogo. Os endereços `0x5F00` a `0x5FFF` são registos: câmara, clip (apenas leitura), desenho de cor, preenchimento de padrão, os três remapeamentos de paleta e os números de página. Os bytes de ambos os lados deste orifício, 65280 no total, constituem a janela de pixéis.

Quando o ecrã cabe nestes 65280 bytes (64×64 e 128×128 cabem, mas 256×256 não), o ecrã começa no endereço 0, linha a linha. O que resta é a folha de sprites selecionada por `0x5F41` (0 é a folha 0). Se esta folha for maior do que o espaço restante, `0x5F42` selecciona a página e o mapa não é apresentado na janela. Se a folha couber, a parte final contém células do mapa seleccionadas por `0x5F43`, com um ou dois bytes cada, e `0x5F44` pagina o mapa quando este não couber na parte final.

Quando o ecrã não cabe, toda a janela ocupa uma página de 65280 pixels. `poke(0x5F40, n)` vira a página do ecrã. `poke(0x5F41, n)` com n diferente de 0 apresenta a folha n, e a folha 0 não é apresentada neste modo. `poke(0x5F43, n)` com n diferente de 0 exibe o mapa n, e este prevalece sobre a folha. O mapa 0 neste modo é apenas apresentado por `mget` e `mset`.

Um `peek` que não atinja um pixel ou um registo retorna 0. Um `poke` neste caso não faz nada. Os flags de sprite, efeitos sonoros e tabelas de dados não estão nesta janela. Utilize `fget`, `sfxget` e `dget`.

Uso prático

Todas as funções estão listadas na parte inferior desta página, juntamente com exemplos práticos de utilização das funções do PIX37. Os links abaixo de cada exemplo fornecem mais informações sobre as

funções.



Movimento suave

Um cartucho de jogo de 128×128, visto de cima. O sprite 1 já está desenhado. As setas aceleram em qualquer direção e o atrito abranda o movimento. O sprite repete-se: metade da imagem que sai da parte inferior é desenhada na parte superior, e o mesmo acontece com as outras margens.

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



Verificando se o próximo movimento está bloqueado.

Um cartucho de jogo 64x64. O chão e a parede da direita já estão no mapa, pintados com o sprite 3, e a flag 0 está definida nesse sprite. As setas movem o sprite 1. A palavra no topo indica "bloqueado" quando o passo seguinte sobrepõe um bloco sólido e "livre" quando não sobrepõe.

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



Um lenço humedecido com flip

Prima A. A transição é desenhada por flip(), uma linha preta de cada vez, em vez de dentro de _draw(). _draw permanece fora do caminho enquanto este é executado, pelo que cada linha permanece no ecrã.

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



Uma tabela de funções (Classe)

O sprite 1 já está desenhado. `make_monster` funciona como uma classe. Devolve uma tabela, e `update` e `draw` são métodos que armazenam as coordenadas x e y do monstro. O número no topo indica a distância percorrida pelo monstro.

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



Quatro formas de ler um botão

A é o botão 4. Mantenha-o pressionado e a carga aumenta. Solte-o e a carga volta a 0. Os botões para cima e para baixo repetem-se enquanto estão pressionados, fazendo com que a linha do menu continue a mover-se. btnd é o primeiro frame de um pressionamento, que este cartucho de jogo utiliza apenas como início desse pressionamento.

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



Disparar com o rato

O sprite 1 já está desenhado no meio. Um ponto branco representa o cursor. O clique esquerdo dispara um pixel na sua direção. A roda de scroll altera a velocidade, indicada no canto.

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



Introduza um nome e confirme-o.

Prima A para abrir o campo. As setas para cima e para baixo alteram uma letra intermitente. A tecla A escreve, X apaga, Y dá espaço e B confirma. Em seguida, a linha apresenta o seu nome.

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



Câmara no jogador

Um cartucho de jogo de 128×128. O sprite 1 começa no meio de um piso de azulejos. As setas movem-se. A câmara mantém o sprite centrado. Os eixos x e y nos cantos não rolam, porque a função `camera()` sem argumentos devolve a vista para o HUD.

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



Algumas linhas de uma visual novel

As falas estão em DATA, em inglês e polaco. A opção "Continua" ou seleciona "Ficar" quando uma escolha está no ecrã. B seleciona "Sair" quando existe uma opção e, caso contrário, alterna o idioma. X alterna o idioma enquanto uma escolha estiver a ser apresentada. O diálogo utiliza a fonte de 8px, pelo que as letras em polaco são apresentadas corretamente.

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



Pinte com a técnica de espiar e picar.

Um cartucho de jogo de 64×64, pelo que o endereço 0 é o pixel superior esquerdo e a linha seguinte começa em 64. O cartucho de jogo escreve pixels, copia uma linha, limpa uma linha e imprime os valores que leu.

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



Uma lista à qual pode adicionar

A acrescenta um número, até 8. B retira o último. X remove o primeiro. A lista é impressa no ecrã.

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



Um nome desmembrado

O cartucho do jogo divide "ada", "12", "blue" e imprime o nome, a sua primeira letra, "12" e a palavra "blue".

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

 Toque uma nota e depois mude-a.

O padrão 0 tem a duração de uma nota, e cada repetição tem a mesma intensidade. O músico A volta a tocar quando essa nota termina. O músico B aumenta o tom e toca essa nota.

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

 Uma música escrita no cartucho do jogo

Dois padrões repetem-se na linha 0, um em cada um dos dois primeiros canais. A linha permanece em 0. ch0 e ch1 representam a nota em cada canal. A edição do padrão, LEN e LOOP no cartucho do jogo permite visualizar o que estes controlos fazem no painel de música.

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

 Três monstros em DADOS

O painel DADOS já possui uma coluna de nome e uma coluna de HP, com os nomes slime, bat e eye. O símbolo A magoa o slime. Os números à direita representam os seus pontos de vida (HP).

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

 Lembre-se da pontuação

A acrescenta um ponto e guarda-o. Sai e abre novamente o cartucho do jogo: o número ainda lá está. B esquece.

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

  Um segundo capítulo noutro ficheiro

Dois cartuchos de jogo partilham a missão de salvar o jogo. Parte 1: A marca o fim do jogo. A Parte 2, aberta primeiro, diz para terminar a Parte 1. Depois disso, a Parte 2 é aberta e B apaga o jogo guardado para que a Parte 1 possa ser jogada novamente.

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

 Uma conquista

O sprite 8 já está desenhado. Prima A dez vezes. A mensagem apresentada mostra a conquista, e o sprite 8 permanece porque está desbloqueado. B remove esta conquista do ficheiro local e reinicia a contagem.

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

 Um quarto em que pode entrar.

Um cartucho de jogo de 256x256. Um ponto branco representa o cursor. O menu oferece as opções de alojar, entrar ou lista de servidores. Hospedar abre uma sala chamada "walk" e inicia o jogo. A introdução apresenta um endereço (dígitos, um ponto, dois pontos) e o jogador B liga-se. A lista de servidores move-se para cima e para baixo, e o jogador A entra na linha. No jogo, as setas movem o sprite 1. Uma moeda é partilhada: o anfitrião move-a, e apanhá-la adiciona 1 à pontuação desse jogador e gera o próximo. Cada sprite, incluindo o teu, mostra a pontuação ao lado. O canto mostra quantos jogadores estão na sala, o endereço da LAN e o endereço público do anfitrião. O jogador B sai: tu e os outros regressam ao menu, e o próximo anfitrião inicia uma nova sala com uma pontuação 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 tecla A mostra a linha seguinte. A função pausa em cada `yield` até que pressione novamente A.

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

 Aguarde meio segundo

A palavra muda de "..." para "oi" passado cerca de meio segundo. O ecrã continua a desenhar enquanto a corrotina está pausada.

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

 Círculos que chegam um a um

Cinco círculos gotejam do topo, separados por dois quadros, e depois caem.

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

 Dois quadros, do relógio

O sprite 1 e o sprite 2 são dois fotogramas da mesma personagem. A imagem alterna quatro vezes por segundo.

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

Todas as funções

Os argumentos entre parêntesis rectos são opcionais. Os índices de tabelas e de strings começam em 1, exceto as linhas e colunas de DADOS, que começam em 0. As cores são índices de paleta.

Índice

_desenhar
_iniciar
_mp
_mp_fail
_mp_juntar-se
_mp_sair
_mp_update
_set_fps
_atualizar
abdómen
claro
achdef
alcançar
alcançado
adicionar
tudo
afirmar
atan2
banda
não
bor
btn
btnd
btnp
btnu
bxor
câmera
teto
cr
círculo
circular
clique
clique
clique
clipe
cls
cocriar
cor
recomeçar
porque
Costatus
contar
cursor
dcols
del
delicatessen
delsave
dget
drows
dset
dstr
fget
preencher
virar
flor
foreach
conjunto
informações
pares
desligar
chave
keystat
chave
linha
carregar
registo
lshr
mapa
máx.
memcpy
conjunto de memórias
obter
meio
min
rato
mset
música
estatística de música
anunciar
ligar
próximo
host
juntar-se
obrigado
nkbstr
nlan
sair
nlista
nouvir
nota
jogadores
enviar
atordoar
ordem
oval
preenchimento oval
pacote
pares
amigo
palrgb
pálido
chamada
espiar
espiar2
espiar4
desempenho
página
cutucar
poke2
poke4
imprimir
imprimir10
imprimir8
conjunto
correto
preenchimento rectificado
segundo
rotl
rotr
correto

preencher
salvar
slot de salvamento
tela
rolar
selecione
mapa de configuração
setpalette
conjuntospr
settex
setturn
efeitos especiais
sfxget
sfxhdr
conjunto de efeitos
sfxstat
sget
sinal
shl
estatísticas de exibição
Shr
pecado
canção
conjunto de músicas
dividir
primavera
quadrado
areia
definir
sspr
estatística
substituto
hora
linha
tom
tostr
desempacotar
uau
rendimento

Vida útil

_init

`_init() ... end`

Executa uma vez quando o cartucho do jogo é iniciado. O código que está fora de qualquer gancho também é incorporado no _init.

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

_update

`_update() ... end`

Lógica do jogo. Chamada a 60 Hz, mesmo quando _draw está limitado ou o host é mais lento. Leia aqui os botões.

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

_draw

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

Desenha um quadro. Sem argumento, ou com 0, é executado com a frequência que o host apresentar. Passe n (1..999) para limitar o desenho a n fotogramas por segundo. _update permanece nos 60 Hz.

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

_set_fps

`_set_fps(n)`

Altera o limite de sorteio (_draw cap) durante a execução do jogo. 0 significa ilimitado. O valor de n está limitado entre 0 e 999.

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

flip

`flip()`

Apresenta o ecrã e aguarda até ao próximo quadro. Utilize-o apenas se estiver a desenhar o seu próprio loop de quadros. Não utilize `busy-wait` após o mesmo.

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

time

`time()`

Segundos desde o arranque.

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

Entrada

btn

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

1 enquanto o botão b estiver premido, caso contrário, 0. p é o jogador, 0..7. Sem argumentos, devolve uma máscara de bits do jogador 0. Botões: 0 esquerda, 1 direita, 2 cima, 3 baixo, 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])`

Verdadeiro apenas no frame _update quando o botão b é premido.

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

btnp

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

Verdadeiro ao pressionar, e novamente enquanto mantido pressionado: 30 ticks de atraso, depois a cada 8 ticks. Sem qualquer botão premido, retorna uma máscara.

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

btnu

`btnu(b, [p])`

Verdadeiro apenas no quadro em que o botão é libertado.

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

mouse

`mouse()`

Devolve as coordenadas x e y do cursor em pixéis do jogo.

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

clickl

`clickl([p])`

Botão esquerdo do rato. clickl() é mantido pressionado. clickl(1) é a aresta de pressão.

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

clickr

`clickr([p])`

Botão direito do rato. Mesma aresta de 1 argumento que o clickl.

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

clickm

`clickm([p])`

Botão do meio ou um clique da roda de scroll.

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

scroll

`scroll()`

Neste frame, é realizado o movimento da roda do rato. Os valores positivo e negativo dependem da direção da roda.

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

keyon

`keyon()`

Abre um campo de texto controlado por botões. As setas para cima e para baixo selecionam uma letra de A a Z. A tecla A escreve a letra. A tecla X apaga-se. A tecla Y abre o espaço. A tecla B confirma e fecha o campo. O teclado não digita neste campo.

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

keyoff

`keyoff()`

Oculta o campo de texto.

```
keyoff()
```

keystr

`keystr([s])`

O texto no campo é introduzido, seguido da confirmação com o caractere 1 no quadro B. De seguida, as letras para cima e para baixo são alteradas. Esta letra fica vazia quando o campo é fechado. A tecla keystr("") limpa o texto.

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

keystat

`keystat()`

1 enquanto o campo de texto estiver aberto, caso contrário, 0.

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

Desenho

screen

`screen()`

Tamanho atual do ecrã do cartucho do jogo em pixéis. Devolve a largura e, em seguida, a altura. Esta é a resolução do jogo, não da janela do editor.

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

cls

`cls([col])`

Limpa o ecrã. A cor padrão é 0.

```
cls(1)
```

color

`color(col)`

Define a cor de desenho utilizada quando uma chamada posterior omite a coluna. Sem argumentos, devolve a cor atual.

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

camera

`camera([x, y])`

Desloca-se a cada chamada de desenho. A função camera() redefine para 0, 0.

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

clip

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

Limita o desenho a um retângulo. Um quinto argumento diferente de zero intersecta o recorte atual. `clip()` restaura o ecrã inteiro. A visualização dos registos de recorte é de leitura apenas.

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

cursor

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

Define onde será feita a próxima impressão sem coordenadas e, opcionalmente, a cor. cursor() devolve x, y.

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

pset

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

Define um pixel do ecrã. Conta para a estatística de pixéis.

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

pget

`pget(x, y)`

Cor do pixel do ecrã nas coordenadas x e y.

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

print

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

Desenha um texto de 3×5 (UTF-8, recortado da fonte Fusion). Sem os parâmetros x e y, imprime na posição do cursor e passa para a linha seguinte.

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

log

`log(...)`

Anexa os argumentos, separados por espaços, a um buffer de texto. A função `pix37_run` imprime este buffer na saída padrão (stdout). Ela não desenha nada.

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

print8

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

Igual à versão impressa, com a fonte Fusion de 8 px.

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

print10

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

Igual à versão impressa, com a fonte Fusion de 10 px.

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

line

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

Linha de x0,y0 a x1,y1. Dois argumentos continuam do final anterior. line() reinicia essa cadeia.

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

circ

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

Contorno circular. O raio está em pixéis.

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

circfill

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

Círculo preenchido.

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

oval

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

Contorno elíptico dentro da caixa delimitadora. Os cantos estão incluídos.

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

ovalfill

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

Elipse preenchida dentro de uma caixa delimitadora.

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

rect

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

Contorno retangular. Os cantos estão incluídos.

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

rectfill

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

Retângulo preenchido.

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

rrect

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

Contorno de retângulo com cantos arredondados. x e y representam o canto superior esquerdo. w e h são as dimensões em pixéis; ambos devem ser maiores que 0. r é o raio do canto e tem como valor por defeito 0. Está limitado a metade do lado menor.

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

rrectfill

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

Retângulo arredondado preenchido. Os mesmos argumentos que o rrect.

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

fillp

`fillp([p])`

Padrão de preenchimento de 16 bits para formas preenchidas posteriormente. 0 é sólido. fillp() devolve o padrão atual. Os bits percorrem o padrão 4×4.

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

pal

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

Remapeia a cor c0 para c1. O modo 0 é a paleta de desenho, 1 a paleta de visualização e 2 a paleta de preenchimento de padrões. pal() redefine.

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

palrgb

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

Define uma entrada da paleta para RGB, com cada canal entre 0 e 255.

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

palt

`palt([col, t])`

Transparência. palt(col) ou palt(col, true) torna col transparente. palt(col, false) torna-a sólida. palt() torna todas as cores opacas.

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

setpalette

`setpalette(n)`

Seleciona o banco de paletas n. setpalette() retorna o banco atual.

```
setpalette(1)
```

tline

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

Desenha uma linha no ecrã enquanto amostra os pixéis do mapa a partir da posição do mapa mx, my. mdx, mdy desloca a amostra do mapa ao longo da linha.

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

Sprites e mapas

spr

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

Desenha o sprite n nas coordenadas x e y. w e h representam o tamanho em pixéis, com um valor por defeito de 8. fx e fy invertem horizontal e verticalmente quando diferentes de zero. tex indica o separador da folha de sprites. rot representa o ângulo em rotações (0..1), na mesma direcção que o seno e o coseno. O valor 0 deixa o sprite sem rotação.

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

sspr

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

Copia um retângulo da folha de sprites para o ecrã, dimensionado para dw por dh. As definições padrão mantêm o tamanho original.

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

sget

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

Cor de um texel na folha de sprites.

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

sset

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

Escreve um texel na folha de sprites.

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

fget

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

Flags de sprite. fget(n) devolve o byte de flag. fget(n, f) é verdadeira quando o bit de flag f (0..7) está a 1.

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

fset

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

Define os sinalizadores do sprite. fset(n, v) escreve o byte inteiro. fset(n, f, v) coloca um bit a 1.

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

setspr

`setspr(n)`

Seleciona a folha de sprites ativa. setspr() devolve a folha atual. Mesmo banco que o settex.

```
setspr(1)
```

settex

`settex(n)`

Alias de setspr.

```
settex(0)
```

map

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

Desenha o mapa. As células cx e cy do mapa são desenhadas nos ecrãs sx e sy. cw e ch iguais a 0 representam todo o mapa. A camada é aceite ou ignorada. tex é a folha do conjunto de tiles.

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

mget

`mget(cx, cy)`

ID do sprite na célula do mapa cx, cy.

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

mset

`mset(cx, cy, n)`

Escreve o sprite n na célula cx, cy do mapa.

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

setmap

`setmap(n)`

Seleciona o mapa n. setmap() devolve o mapa atual. O mapa 0 é um mapa real.

```
setmap(1)
```

Memória

peek

`peek(addr)`

Lê um byte da janela de 64 KB. A janela é o ecrã ativo, depois os pixéis do sprite e, por fim, as células do mapa quando o ecrã cabe em 65280 bytes. Os ecrãs maiores são paginados com poke(0x5F40, page). Os endereços que não estão mapeados devolvem 0. Também escrito em @addr.

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

poke

`poke(addr, ...)`

Escreve bytes no endereço especificado. Argumentos extra escrevem os bytes seguintes. Os endereços não mapeados são ignorados. Escreve também $addr, val. Tocar no ecrã conta como uma escrita de pixel.

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

peek2

`peek2(addr)`

Valor de 16 bits sem sinal, little-endian, no endereço.

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

poke2

`poke2(addr, val)`

Escreve um valor de 16 bits sem sinal, em formato little-endian.

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

peek4

`peek4(addr)`

Valor de 32 bits sem sinal, formato little-endian.

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

poke4

`poke4(addr, val)`

Escreve um valor de 32 bits sem sinal, em formato little-endian.

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

memcpy

`memcpy(dst, src, len)`

Copia len bytes dentro da mesma janela de 64 KB.

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

memset

`memset(dest, val, len)`

Preenche len bytes a partir de dest com val.

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

Matemática

abs

`abs(x)`

Valor absoluto.

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

flr

`flr(x)`

Arredonda para menos infinito.

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

ceil

`ceil(x)`

Arredonda para o infinito positivo.

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

max

`max(a, b)`

O maior dos dois valores.

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

min

`min(a, b)`

O menor dos dois valores.

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

mid

`mid(a, b, c)`

Valor intermédio entre os três. Útil como limitador.

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

sgn

`sgn(x)`

Sinal de x: -1, 0 ou 1.

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

sqrt

`sqrt(x)`

Raiz quadrada. Um argumento negativo não é um número.

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

sin

`sin(x)`

O seno de x assume os seguintes valores: 0, 0,25, 0,5, 0,75, completando uma volta no círculo. Não em radianos.

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

cos

`cos(x)`

O coseno, por sua vez, segue a mesma convenção do seno.

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

atan2

`atan2(dx, dy)`

O ângulo do vector dx, dy, varia de 0 a 1.

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

rnd

`rnd([x])`

Número aleatório em [0, x). O valor por defeito de x é 1, logo rnd() é 0..1. Para um dado, flr(rnd(6))+1.

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

srand

`srand(seed)`

Inicializa o gerador de números aleatórios para que uma sequência aleatória posterior possa ser repetida.

```
srand(1)
```

band

`band(a, b)`

AND bit a bit. Também escrito como a&b.

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

bor

`bor(a, b)`

OU bit a bit. Também escrito como a | b.

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

bxor

`bxor(a, b)`

XOR bit a bit. Também escrito como ^^ b.

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

bnot

`bnot(a)`

NOT bit a bit. Também escrito ~a.

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

shl

`shl(a, n)`

Deslocar para a esquerda. Também escrito como << n.

shr

`shr(a, n)`

Deslocamento aritmético para a direita. Também escrito como >> n.

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

lshr

`lshr(a, n)`

Deslocamento lógico para a direita. Também escrito como >>> n.

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

rotl

`rotl(a, n)`

Rode os bits para a esquerda. Também escrito como <<> n.

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

rotr

`rotr(a, n)`

Rode os bits para a direita. Também escrito como >>< n.

Tabelas e valores

add

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

Anexa v à tabela t. Se for fornecido i, insere no índice correspondente, com base 1.

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

del

`del(t, v)`

Remove a primeira entrada igual a v.

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

deli

`deli(t, [i])`

Remove o índice i e devolve esse valor. Sem i, remove a última entrada.

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

all

`all(t)`

Iterador sobre a parte do array, por ordem. Os espaços vazios interrompem o percurso da mesma forma que termina um array de base 1.

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

foreach

`foreach(t, fn)`

Chama fn(v) para cada elemento da matriz.

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

pairs

`pairs(t)`

Iterador sobre cada chave e valor, incluindo campos nomeados.

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

ipairs

`ipairs(t)`

Iterador sobre a parte do array com índice baseado em 1.

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

next

`next(t, [k])`

Retorna a próxima chave e valor após k. next(t) começa no início. Usado por pares.

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

count

`count(t, [v])`

Comprimento da peça da matriz. Se for fornecido v, conta quantas entradas são iguais a v.

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

pack

`pack(...)`

Agrupa os argumentos numa nova tabela.

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

unpack

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

Retorna t[i] até t[j] como valores separados. Os valores por defeito abrangem toda a parte do array.

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

select

`select(i, ...)`

select("#", ...) devolve quantos argumentos se seguem. select(i, ...) devolve os argumentos a partir de i.

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

assert

`assert(c, [msg])`

A execução é interrompida com um erro quando c é falso ou nulo. msg substitui o texto padrão.

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

pcall

`pcall(f, ...)`

Chama a função f. Em caso de sucesso, devolve verdadeiro e os resultados. Em caso de erro, devolve falso e o texto do erro.

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

Cordas

chr

`chr(n)`

Um caractere de um código de byte 0..255.

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

ord

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

Códigos de bytes de s do índice i ao j. Os índices começam em 1. O predefinido é o primeiro caractere.

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

sub

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

Substring de i a j, começando em 1. Omita j para obter a cauda.

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

split

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

Divide s numa tabela. O parâmetro sep utiliza a vírgula como padrão. Um valor numérico para sep representa a largura do bloco. Quando verdadeiro, o conv converte os blocos usando tonum.

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

tonum

`tonum(s, [base])`

Analisa um número. A base pode ser de 2 a 36. Retorna nulo quando o texto não é um número.

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

tostr

`tostr(v, [digits])`

Converte um valor em texto. `digits` define as casas decimais para números.

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

Áudio

sfx

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

Reproduz o padrão n e devolve o canal. Omita ch ou passe -1 e o primeiro canal livre será escolhido. ch -2 interrompe o padrão n em todos os canais. n -1 interrompe um canal ou todos os canais quando ch é omitido. n -2 permite que um padrão em loop termine em vez de continuar em loop. sfx() sem argumentos retorna -1.

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

sfxget

`sfxget(n, step)`

Devolve uma tabela de um passo do padrão: [1] altura, [2] onda, [3] volume, [4] efeito. A altura é de 0 a 63 (Dó-0 a Ré#5). A onda é de 0 a 15. O volume é de 0 a 7, e 0 é silencioso. O efeito é de 0 a 7.

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

sfxset

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

Escreve um passo de padrão. Todos os seis argumentos são obrigatórios.

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

sfxhdr

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

Cabeçalho do padrão. sfxhdr(n) devolve a tabela de cabeçalho. sfxhdr(n, key, val) define um campo: speed, loop_in, loop_out, level, noise, crush, dtune, rverb, dampn, swing, steps. Os números de 0 a 10 seleccionam os mesmos campos por esta ordem. velocidade é TIC, ticks por passo. Quanto maior o valor, mais lento. 15 corresponde a 120 batimentos por minuto. Os sintetizadores de 16 bits aceitam valores de 1 a 255. Os sintetizadores com menor largura de banda ajustam-se à sua lista de TIC.

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

sfxstat

`sfxstat([ch])`

O ID do padrão está a ser reproduzido no canal ch ou -1. O canal predefinido é 0.

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

note

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

Toca uma nota sem padrão. Afinação 0..63, onda 0..15, volume 0..7, efeito 0..7. "ticks" indica a duração, nos mesmos intervalos de 120 Hz que o "TIC". Omitir "ticks" resulta numa duração de 15, um passo padrão. O efeito é reproduzido durante este tempo e, em seguida, o canal fica silencioso.

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

music

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

Inicia a música n do início. `music(-1)` interrompe a reprodução. Chamar `music` com o ID de uma música que já está a tocar mantém-na em execução. `music()` devolve o ID da música em reprodução ou -1. `fade` é um valor em milissegundos: a música aumenta gradualmente de intensidade (fade in) ou diminui gradualmente de intensidade (fade out) quando n é -1. Os bits da máscara (mask) que estão definidos representam os canais utilizados para a música. 0 significa todos os oito canais. Os outros canais permanecem livres para efeitos sonoros.

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

musicstat

`musicstat()`

Tabela da música em reprodução. `song` e `row` são o ID da música e a linha, ou -1 quando não está a ser reproduzido nada. `step` e `pos` têm oito números. `step[1]` é o canal 0, `step[8]` é o canal 7. `step` é o índice da nota, ou -1 quando este canal está em silêncio. `pos` indica o quanto esta nota avançou, de 0 a 1. `tick` indica quantos ticks esta linha já reproduziu. `played` indica quantas linhas foram iniciadas desde `music()`, incluindo a que está a ser reproduzida.

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

songget

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

O ID do padrão está armazenado numa célula de música. O valor de `ch` é de 0 a 7. O número 255 significa que a célula está vazia. Esta é a chamada em tempo real; a dica de código ainda mostra uma ordem de argumentos antiga.

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

songset

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

Escreve um ID de padrão numa célula de música. Utilize 255 para limpar a célula.

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

stat

`stat(n)`

Apenas áudio. 16..23 é o ID do efeito sonoro no canal n-16. 24..31 é o padrão musical no canal n-24. Qualquer outro valor de n devolve 0. O tempo de fotograma e a RAM são informações de desempenho, não estatísticas.

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

Tabelas de dados

dget

`dget(t, row, col)`

Lê uma célula da tabela de dados t do painel DADOS. A linha e a coluna começam em 0.

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

dset

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

Escreve numa célula. Utilize uma string para uma coluna de texto.

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

dstr

`dstr(t, row, col)`

A célula como texto. Os números são convertidos.

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

drows

`drows(t)`

Quantas linhas tem a tabela t?

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

dcols

`dcols(t)`

Quantas colunas tem a tabela t?

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

Guardar e conquistas

save

`save(k, v)`

Armazena v sob a chave k no ficheiro de gravação. v pode ser um número, uma string ou uma tabela que contenha esses elementos. O ficheiro é compactado.

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

load

`load(k)`

Lê a chave k. Retorna nil quando a chave está em falta.

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

delsave

`delsave(k)`

Remove a tecla k do ficheiro de gravação.

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

saveslot

`saveslot([name])`

Seleciona o ficheiro de gravação dentro da pasta do autor. Sem nome, o slot é o título do jogo. Retorna 1 se o ficheiro já existir.

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

achieve

`achieve(i)`

Desbloqueia a conquista i, de 0 a 15. Armazenada localmente. Uma versão do Steam também define o ACH_i. Uma sessão de catálogo reporta isso à conta do jogador.

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

achieved

`achieved(i)`

1 quando a conquista i já estiver desbloqueada.

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

achdef

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

Mensagem de notificação para a conquista i, apresentada durante 15 segundos na parte inferior do ecrã. spr é um bloco opcional de 8×8, tex é a sua folha.

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

achclear

`achclear(i)`

Limpa a conquista i no ficheiro local. Uma conquista ACH_i do Steam que já estava definida permanece definida.

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

Rede

whoami

`whoami()`

O seu ID de jogador, de 0 a 15, enquanto a sessão estiver ativa. Offline, é -1.

```
id = whoami()
```

nhost

`nhost()`

1 se esta máquina for o host, caso contrário 0.

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

nplayers

`nplayers()`

Quantos ID ativos existem no intervalo de 0 a 15?

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

nlisten

`nlisten()`

Inicia o alojamento de UDP e devolve o endereço da LAN como ip:porta.

```
addr = nlisten()
```

njoin

`njoin(addr)`

Liga-se via IP ou IP:porta.

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

nleave

`nleave()`

Sai da sessão. Os pares vêem _mp_leave com o motivo 0.

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

nsend

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

Envia uma mensagem com perda de dados. O nome é o evento. Seguem-se até quatro valores adicionais. O valor de p, de 0 a 15, envia uma mensagem privada para esse participante; omita p para enviar a todos.

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

ncall

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

Tem o mesmo formato que o nsend, mas a mensagem é reenviada até ser confirmada. Três falhas consecutivas terminam a sessão.

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

nlan

`nlan()`

IP e porta da LAN atual após o nlisten.

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

nstun

`nstun()`

O seu endereço IP público:porta do STUN. Um convidado noutra rede pode tentar ligar-se a ela.

```
pub = nstun()
```

setturn

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

O TURN é uma funcionalidade de fallback que é utilizada quando um caminho direto falha. A função setturn(), sem argumentos, desativa o TURN. O endereço é host:porta.

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

nlist

`nlist(game)`

Tabela de sessões abertas para esse nome de jogo. Cada entrada contém o endereço, os jogadores e o nome.

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

nannounce

`nannounce(game)`

O sinal de pulsação faz com que o host apareça na lista de jogos (nlist). Inicie o jogo com um temporizador enquanto estiver a alojar.

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

nkb

`nkb([on])`

Campo de endereço. nkb(1) abre, nkb(0) fecha, nkb() informa se está aberto. As teclas para cima e para baixo alteram um caractere, de 0 a 9, um ponto ou dois pontos. A tecla A escreve. X apaga. O teclado não digita neste campo.

```
nkb(1)
```

nkbstr

`nkbstr([s])`

O texto no campo de endereço, então o caractere para cima e para baixo está a mudar. nkbstr("") limpa o texto.

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

_mp

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

Você escreve esta função. É executada quando chega uma mensagem. `from` é o ID do remetente. `nome` é o evento. Até quatro valores podem ser enviados a seguir.

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

_mp_join

`_mp_join(p)`

Você escreve esta função. Um nó recebeu o ID p. Ela também é executada para si; whoami() é esse ID.

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

_mp_leave

`_mp_leave(p, why)`

Você escreve esta função. O utilizador p saiu. Porque é que 0 representa uma saída normal? Porque é que 1 representa uma ligação perdida?

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

_mp_update

`_mp_update()`

Você escreve esta função. É executado a cada 15 fotogramas de atualização, aproximadamente 4 Hz. Um bom local para enviar posições via nsend.

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

_mp_fail

`_mp_fail()`

Você escreve esta função. A sessão é terminada: o host desapareceu ou o ncall falhou três vezes.

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

Corrotinas

cocreate

`cocreate(f)`

Cria uma corrotina a partir da função f. Não é executada até que o coresume seja iniciado.

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

coresume

`coresume(c, ...)`

Executa a corrotina c até que esta produza um resultado ou devolva. Em caso de sucesso: devolve verdadeiro e os valores produzidos. Em caso de erro: retorna falso e o erro.

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

costatus

`costatus(c)`

"Suspenso", "em execução" ou "morto".

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

yield

`yield(...)`

Devolve o controlo a quem chamou o coresume. Os valores passam a ser os resultados desse coresume.

```
yield(1)
```

Medir

info

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

Desenha o HUD em tempo real em `_draw`: fps (média de 1 segundo), upd, drw, px, max, ram, peak, ramc e net quando uma sessão está activa. `info()` utiliza o cursor de impressão. `info(col)` apenas define a cor. `info(x, y)` ou `info(x, y, col)` posicionam o cursor. Os números usam B, k ou m para se ajustarem a um ecrã pequeno. A chamada em si não é contabilizada em px.

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

showstats

`showstats([on])`

O mesmo HUD que a informação, desenhado automaticamente a 0, 0 após _draw, ignorando a câmara e o recorte. showstats(0) oculta-o. showstats(1) ou showstats() exibe-o.

```
showstats(1)
```

perf

`perf(n)`

Um número no HUD, para que possa desenhá-lo você mesmo. 0 fps, 1 upd, 2 drw, 3 px, 4 max (frame mais pesado neste segundo), 5 ram, 6 ramc (ram mais o cartucho do jogo), 7 pico de ram desde que o cartucho do jogo foi iniciado.

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