PIX37 マニュアル

編集後、コードへ。書籍のオフライン版はpix37.party/docsにあります。

エディタ

PIX37 DEV はエディタです。ここでゲームを描画、記述、再生し、ゲームカートリッジまたはスタンドアロンビルドとしてエクスポートします。また、ゲームカートリッジを再生したり、PIX37.PARTY を開いてゲームカートリッジをダウンロードしたり、カタログを使用したりすることもできます。PIX37 DEV EDUCATIONAL は同じエディタで、無料です。ビルド機能、ゲームカートリッジのエクスポート機能、PIX37.PARTY へのアクセス機能はありません。PIX37 Console はプレイヤーです。.p37 ゲームカートリッジ.p37再生し、PIX37.PARTY を開いてゲームカートリッジをダウンロードしたり、カタログを閲覧したりできます。

プロジェクトとゲームカートリッジ

.pix37ファイルはプロジェクトファイルであり、ソースタブ、スプライトシート、マップ、サウンド、データテーブルなどが.p37れます。.p37ファイルはプレイヤーが実行するゲームカートリッジであり、高密度に圧縮されたバイトコードです。

セーブデータと実績は、アプリデータと同じオーサーフォルダに保存されます。Windows では、セーブデータはC:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.p37s 、実績はC:\\Users\\Ada\\AppData\\Roaming\\PIX37\\Mira\\Pong.achのようになります。Linux では、同じファイルは~/.local/share/PIX37/Mira/Pong.p37sと~/.local/share/PIX37/Mira/Pong.achになります。Ada はコンピュータのアカウントです。Mira はオーサー名で、フォルダ名でもあります。Pong はセーブスロットで、デフォルトではゲームタイトルになります。save save .p37sを書き込みます。最初の実績が解除されると.achが書き込まれます。ファイルが既に存在する場合、ゲームカートリッジが起動するとランナーがそれらを読み取ります。

開発者メニュー

Escキーで開発者メニューが開きます。ゲーム実行中にEscキーを押すとエディターに戻ります。Enterキーまたはaキー（デフォルトはVキー）で確定します。戻るボタンはBキー（デフォルトはCキー）です。

プロジェクトがロードされていない場合、メニューは次のようになります: プロジェクトの作成、プロジェクトのロード、ビルド、ゲームカートリッジのロード、PIX37.PARTYへの接続、設定。

プロジェクトがロードされたら、次のようになります: プロジェクトの保存、プロジェクトのロード、プロジェクトの作成、アセット、ビルド、ゲームカートリッジのロード、PIX37.PARTYへの接続、設定。

Ctrl+Sで開いているプロジェクトを保存します。Ctrl+Rで実行します。

プロジェクトを作成する

プロジェクト作成を開始する前に、「プロジェクトの作成」機能でゲームカートリッジの入力を求められます。

- **ビデオ**— 1ビット（2色）、2ビット（4色）、4ビット（16色）、8ビット（256色）。
- **オーディオ**— シンセサイザーのサンプル深度：16、8、4、2、または1ビット。
- **画面サイズ**— プリセットには、64×64、128×128、160×90、256×256、320×180が含まれます。カスタムサイズでは、1×1から512×512までの任意のサイズを選択できます。
- **パレット**— 色のバンク。最初のバンクは、以下の 16 色セットから始まります。LOAD PALETTE では、組み込みのセット (PICO-8 16、PICO-8 32、Game Boy、1-bit、CGA、Sweetie-16、および Lospec から取得したより長いリスト (Resurrect 64、Apollo、Endesga 32 および 64、CC-29、SLSO8、Pear36、Oil 6、Duel、AAP-64、Lost Century など) が適用されます。パレット ファイルを読み込むか、独自の PNG を読み込んでその画像からパレットを取り出すこともできます。



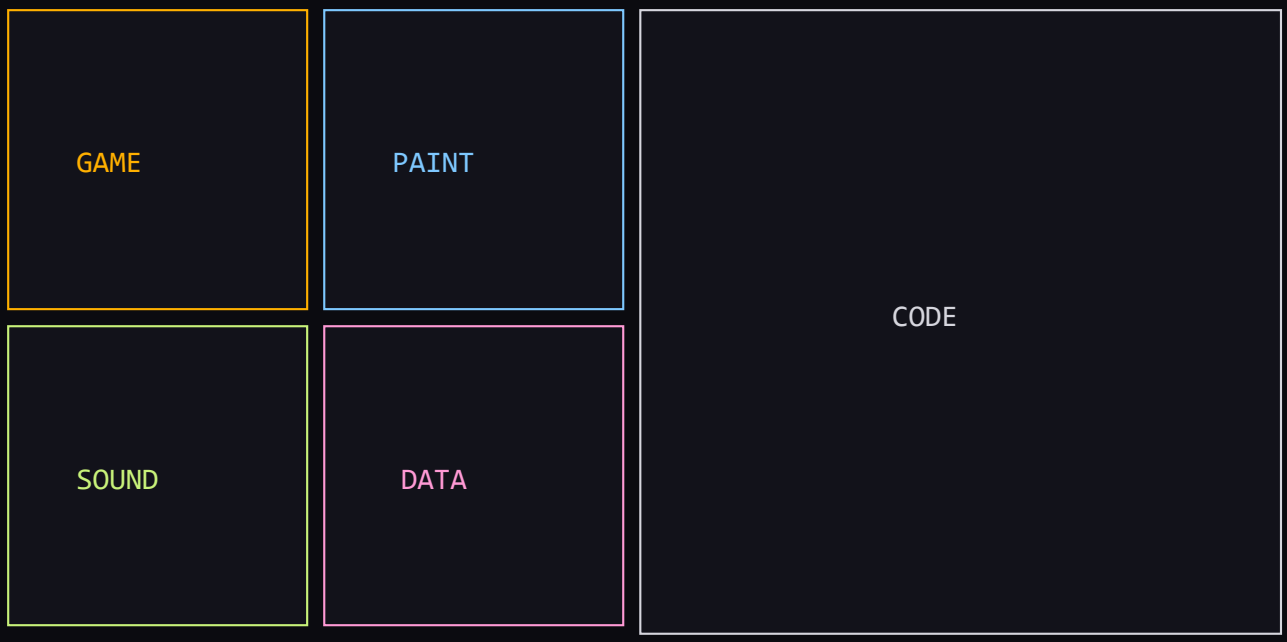
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

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

`cls`、`pset`、`spr`などのコマンドで使用されるカラーインデックス。パレットエディタで任意のインデックスを再描画できます。

5つのパネル

エディタは、GAME、PAINT、CODE、SOUND、DATAの順になっています。マップはPAINTの中にあります。効果音と楽曲はSOUNDの中にあります。パネルをクリックすると、そのパネルで作業できます。CODEがフォーカスされている場合、SOUNDとDATAは文字キーを無視するため、文字を入力してもピアノは演奏されません。



レイアウト全体は1:1:4:1:1順で表記されています。各数字は、そのパネルが行内で占める割合を示しています。GAME、PAINT、CODE、SOUND、DATAの順です。128×128ピクセルのセルの中央に小さな画面が配置されます。128ピクセルより大きい画面では、GAME列の幅が広がります。

設定 → グローバル → レイアウトで配置を切り替えます。

コード	あなたが見るもの
`1:1:4:1:1`	すべて。ゲームとサウンドが縦に並び、ペイントとデータが縦に並び、コードは右側にあります。
`1:1:4:1:1 2X`	同じ配置です。CODEの部分は密度が2倍になっているので、2倍の量のテキストが表示されます。
`1:0:1:0:1`	ゲーム、データ、コード。
`1:0:1:0:1 2X`	同じ構成だが、CODEの密度が高い。
`0:1:1:1:1`	ゲームのプレビューはありません。ペイント、コード、サウンド、データ。
`0:1:1:1:1 2X`	同じ構成だが、CODEの密度が高い。
`0:1:4:1:2`	スタジオ。ワイドなCODE、より背の高いDATAストリップ、PAINT、そしてSOUND。
`0:1:4:1:2 2X`	同じ構成だが、CODEの密度が高い。

`0:0:4:0:4`	コードとデータのみ。どちらも大容量。
`0:0:4:0:4 2X`	同じ構成だが、CODEの密度が高い。
`0:1:4:1:0`	ペイント、サウンド、そして幅広いコード。
`0:1:4:1:0 2X`	同じ構成だが、CODEの密度が高い。
`0:4:0:0:0`	ペイントのみ、256×256ピクセル、用紙の1ピクセルがキャンバスの1ピクセルに対応します。

2X行では、同じパネルが維持されます。CODEパネルは2倍の解像度で描画され、同じセルに縮小されるため、テキスト、タブ、およびガターの高さは半分になり、画面に収まるソースコードの量も2倍になります。

ゲーム

実行中のゲームカートリッジのプレビューです。ゲームパネルをクリックするか、Ctrl+Rを押してプレビューを実行します。設定→ゲームパネルでパネルの設定を行います。

- **プレビュー**— 待機時の色、連続表示、または編集時の自動再読み込み。
- **新しいウィンドウでプレイする**— ゲームはエディターの横に開きます。
- **表示**— プレビューをRGB888またはRGB565として表示します。
- **ゲーム64** — 64×64の画面は、セルの中央に1:1で配置されるか、セルを埋めるように2倍に拡大されます。
- **シェーダー**とは、ゲーム内のCRTフィルターや各種フィルターのことで、エディター上のフィルターとは別個のものです。
- **色**— パネルの背景色。

ペイント

スプライトシートと、同じパネルのマップ側にはタイルマップが表示されます。シートは画面サイズよりも大きくできます。スプライトプレビューのサイズは8、16、32、64、128です（128はシートのサイズが128ピクセル以上の場合のみ表示されます）。スプライトプレビュー上でマウスホイールをスクロールするとズームできます。ブラシサイズは1、2、3、4です。

鍵	道具
B	鉛筆
M	長方形を選択

Q	ラッソ
S	スタンプ
G	バケツ
私	スポイト
V	動く
L / Shift+L	直線 / 曲線
U / Shift+U	中空の長方形 / 中空の楕円
F / Shift+F	塗りつぶされた長方形 / 塗りつぶされた楕円
D	ディザー
[ ]	小さめのブラシ／大きめのブラシ
Ctrl+Z / Ctrl+Y	元に戻す／やり直し。Ctrl+Shift+Zでもやり直しができます。
Ctrl+C / Ctrl+X / Ctrl+V	コピー、切り取り、貼り付け
Ctrl+A	キャンバスを選択してください

このマップは、ツール数を減らしつつ、同じアイデアに基づいています。Bキーで鉛筆、Mキーで長方形を選択、Sキーでスタンプ、Vキーで移動、そしてCtrl+Z、Ctrl+Y、Ctrl+C、Ctrl+Vキーを使用します。Altキーを押しながらマップのテクスチャをズームします。マップ上のブラシサイズは、ペイントツールに連動します。

各スプライトには8ビットのフラグがあります。ペイント関数はスプライトにフラグを設定し、 `fget`と`fset` ゲーム内でフラグを読み取ります。

設定 → ペイントエディターでそのパネルを設定できます。

- **パレット**— どのバンクが描画されるか、また、読み込みと保存。
- **プレビュー**— RGB888またはRGB565。
- **ブラシカーソル**— ペイント作業中に使用するポインター。
- **ピクセルスナップ**— ブラシがピクセルにスナップする仕組み。
- **大きなブラシ**— 大きなブラシは、左上から、または全体の接地面から折れます。
- **玉ねぎの皮**——他のフレームの残像、そしてその残像がどの方向を向いているか。
- **地図のズーム**— 真のピクセル単位、または自由ズーム。
- **色**— ペイントUIの色。

コード

タブはプロジェクト内の別々のファイルです。タブ名をクリックすると名前を変更できます。Enter キーを押すと変更が確定します。_init、 `_init` 、または`_draw` `_update`する複数のタブは、タブの順序で結合されます。最初のタブが最初に実行され、最後のタブが最後に実行されます。フック内にないコードは`_init`の一部になります。フックがまったくないゲームカートリッジは、1 つの`_draw`として扱われます。

鍵	アクション
矢印	カーソルを移動します。Shiftキーを押すと選択範囲が広がります。
Ctrl+左/右	言葉で。
Home / End、 Ctrl+Home / End	行の先頭または末尾、あるいはタブの先頭または末尾。
タブ	スペースを2つ挿入してください。補完候補リストで、名前を承認してください。
入力	改行。前の行と同じようにインデントされます。補完を受け付けます。
Ctrl+C / X / V	コピー、切り取り、貼り付け。
Ctrl+Z / Ctrl+Y	元に戻す／やり直し。
Ctrl+A	タブを選択してください。
Ctrl+D	行または選択範囲を下方向に複製します。
Ctrl+/	`--`でコメントまたはコメント解除できます。
Ctrl+F	検索。F3キーで次のヒットへジャンプ。Escキーでバーを閉じる。
Ctrl+G	行番号へ移動してください。
逃げる	検索、ジャンプ、またはヒントリストを最初に閉じます。2回目のEscキーを押すと、エディタから開発者メニューに移動します。

関数名を入力するとヒントが表示されます。Enter キーまたは Tab キーを押すと名前が挿入され、設定で「完了 (」がオンになっている場合は`(`が表示されます。設定 → コード エディタ:

- **検索**— アクティブなタブ内、またはすべてのタブでCtrl+Fを押します。
- **テキストの高さ**— グリフを5ピクセルにトリミングするか、グリフ全体を表示します。
- **文字列サイズ**— 文字列リテラルのフォントサイズ。
- **コメントのフォントサイズ**は3×5、または現在のUI言語のFusionフォントを使用してください。
- **完了（** — 補完が括弧をタイプするかどうか）。
- **コードヒントの**オン/オフ。
- **ヒントの言語**— メニューの言語、または英語。
- **キャレットスペース**— 空行に余分なオフセットを追加。
- **ガター（綴じしろ）** —行番号。

- **色**— 背景、コメント、タブ、数字、演算子、キーワード、文字列、エンジン関数、独自の関数、変数、括弧、キャレット、選択範囲、エラー行、警告。

音

パネルには 2 つのタブがあります。SFX は 1 つのパターン、つまり短い音符の連続です。SOUND は曲で、これらのパターンを 8 つのチャンネルに配置します。TIC は、1 つのステップが何ティック待つかを示します。時計は 1 秒間に 120 ティックで、1 つのステップは 16 分音符なので、15 ティックは 1 分間に 120 拍になります。TIC の値が大きいほど、パターンは遅くなります。2 つのパターンは、一方の TIC がもう一方の TIC を割り切る場合に同期します。15 と 30 は同期しますが、15 と 8 は同期しません。

マウスで音符を選択するとピアノが演奏されます。白鍵はZXCVBNM（CDEFGAB）です。黒鍵はSDGHJ（C# D# F# G# A#）です。任意の値にカーソルを合わせて数値を入力するか、カーソルを合わせてホイールをスクロールしてください。音高の数値は0（C-0）から63（D#5）までです。

あるパターンでは、LOOPとLENは同じ2つの数値です。OUTが0より大きい場合、INからOUTまでループし、タイトルにはLOOPと表示されます。OUTが0でINが0より大きい場合、LENになります。つまり、最初からその数の音符が演奏され、その後パターンが終了し、タイトルにはLENと表示されます。両方とも0の場合、すべてのステップが演奏されます。

シンセサイザーの編集範囲は、ゲームカートリッジで選択したビット深度によって異なります。設定 → SFX エディターでは、ビット深度、ノートの自動進行（オフ、ノートチェーン、フィールドチェーン）、スペースキーで再生を再開するか、開始と停止を切り替えるか、エディターの色を設定します。

16ビット

各ステップには、音符、音量、波形、エフェクトがあります。音量は0から7までで、0は無音です。

波:

- **0 三角形**
- **1 傾斜した鋸**
- **2 鋸**
- **3平方メートル**
- **4パルス**— 狭いパルス
- **5つの臓器**
- **6 ノイズ**
- **7フェイザー**
- **8 PWM** — 幅が変化する正方形
- **9パルス25%**
- **10サイン**
- **11 金属音**
- **12 FMベル**

- **13 摘む**
- **14 スーパーソー**
- **15 PWM (C64)**

単音に対する効果:

- **0 なし**
- **1スライド**— このステップを横切って次の音符へピッチとボリュームが滑らかに移動します
- **2. ビブラート**— ピッチの揺れ
- **3ドロップ**- ピッチは静寂に落ち、いつものキック
- **4 フェードイン**
- **5 フェードアウト**
- **6 アルペジオ**— この音符と次の3つの音符を、8ティックごとに1つずつ移動します。TIC 8以下の場合は、4ティックごとに移動します。
- **7 速いアルペジオ**— 同じ 4 つの音符を 4 ティックごと、または TIC が 8 以下の場合は 2 ティックごとに演奏します。

ノートの上にあるパターンヘッダー：

- **TIC** （ティック）は1ステップあたりのティック数で、1から255までです。15は1分間に120拍です。数値を入力するか、1ずつ増やしてください。
- **ループイン/アウト**— OUTが0より大きい場合、パターンはINからOUTまでループします。OUTを0のままにすると、タイトルはLENになります。INは演奏される音符の数、0はパターン全体を意味します。
- **ノイズ**——パターン全体がノイズになる。
- **CRUSH** — 各波の話題性のあるバリエーション。
- **DTUNE** — 0 オフ、1 軽いコーラス、2 強いコーラス。
- **RVERB** — リバーブ：0 オフ、1 ショート、2 ロング。
- **DAMPN** — ローパスフィルター：0 オフ、1 弱、2 強。
- **レベル**— パターン全体のボリューム、0〜7。
- **スイング**— シャッフル。奇数ステップは少し長めに、偶数ステップは少し短めに待つ。
- **ステップ数**— パターンで使用するステップ数：32、24、16、または12。

8ビット

ステップには音符と波形があります。波形は、矩形波、三角波、ノコギリ波、ノイズの4種類です。ヘッダーには、TIC、LOOP IN / OUT、FX、LEVELがあります。TICは1、2、3、4、5、6、8、10、12、15、16、20、24、30、40、60のいずれかです。FXはパターン全体に適用されるエフェクトで、0はなし、1はビブラート、2はアルペジオ、3はフェードアウトです。LEVELは0から3です。

4ビット

ステップとは音符のことです。ヘッダーには、TIC、LOOP（オンまたはオフ）、WAVE（同じ4つの波形）、OCT（音符が属するオクターブ）があります。TICは4、6、8、10、12、15、20、30のいずれかです。

2ビット

ステップは、空いているもの、ドラム、ヒット、またはクラッシュのいずれかです。ヘッダーにはTICとBASEがあります。TICは8、15、16、または30です。BASEはドラムとヒットの高さを設定します。

1ビット

ステップとは音符のことです。TICは8、15、16、30の4種類で、2ビットと同じ速度です。

歌

「サウンド」タブは開いている曲です。8つの列はチャンネルを表します。各セルはパターンIDです。255（空欄）は、その行のチャンネルが無音であることを意味します。X列はコマンドです。continue、stop、loop start、loop endのいずれかを選択できます。Stopは、その行の後で曲を終了します。Loop endは、最後のループ開始位置に戻ります。

行は、LOOP OUTが0の一番左のパターンが終了すると終了します。それより右にあるパターンは行を長くしません。長いパターンは切り捨てられ、短いパターンは最初のステップから再び開始され、行が終了するまで続きます。LOOP OUTが0より大きいパターンは、LOOP INとLOOP OUTの間をループし、行を終了しません。行上のすべてのパターンがループした場合、行はそのままの状態になります。

ショートカット

鍵	アクション
ピアノの鍵盤	マウスで選択した手順についてメモを書いてください。
上 / 下	カーソル位置の値を変更します。
空間	最初の行から、開いている効果音、または開いている曲を再生します。効果音エディターの設定から、開始/停止または再開できます。
Ctrl+Z / Ctrl+Y	元に戻す／やり直し。
Ctrl+C / Ctrl+X / Ctrl+V	選択範囲をコピー、切り取り、貼り付けます。

データ

名前付きテーブル。各テーブルは行と列で構成されます。コード内の`dget`、`dset`、`dstr`、`drows`、および`dcols`はこれらのテーブルを使用します。コード内の行番号と列番号は 0 から始まります。

ヘッダーで切り替えられる列の種類:

- **ブール値**— 1ビット。

- **2** — 2ビット。
- **4** — 4ビット。
- **8** — 1バイト。
- **16** — IDや大きな数値に使用する16ビット。
- **spr** — スプライトID。
- **str** — システムフォントで描画されたテキスト。
- **str8**と**str10** — 同じテキストを縦に高く表示したもの。
- **f16**と**f32**は、それぞれ半精度浮動小数点数と完全浮動小数点数を表します。

列ヘッダーにポインターを置いた状態で、Ctrlキーを押しながらスクロールすると、その列のタイプが変更されます。通常のスクロールでは、設定 → データエディターで選択した方向に応じて、列を横切るか行を下に移動します。Ctrlキーを押すと方向が反転するため、ホイールが列に設定されている場合に、ヘッダーでCtrlキーを押しながらスクロールするとタイプが変更されます。

矢印キーで移動します。Enterキーでセルを編集します。設定 → データエディタ:

- **ホイール**— 列をスクロールするか、行をスクロールします。
- **色**— グリッド、ヘッダー、および選択範囲の色。

資産

ASSETS は、プロジェクト全体ではなく、開いているパーツのみをインポートおよびエクスポートします。スプライトは、開いているシート上の PAINT で開いているスプライトを使用します。マップは、開いているマップです。SFX は、開いているパターンです。ミュージックは、開いている曲です。データは、開いているテーブルです。コードは例外で、すべてのタブが書き込まれ、すべてのタブが読み戻されます。

親切	ファイル	何が動くのか
スプライト	`.png`	開いているスプライト。
地図	`.p37map`	開いた地図。
効果音	`.p37sfx`	オープンパターン。
音楽	`.p37mus`	オープニング曲です。パターン番号はそのままです。効果音が必要な場合は、別途エクスポートしてください。
データ	`.p37dat`	オープンテーブル。
コード	`.lua`	すべてのタブ。

設定

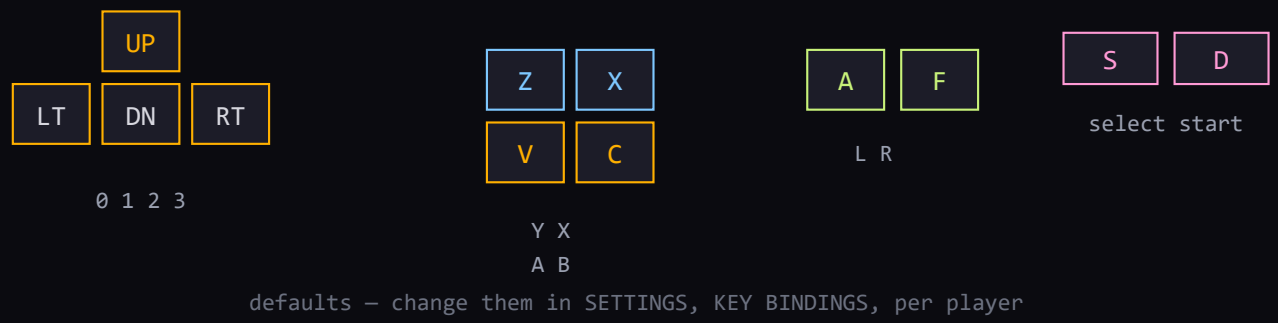
ルートリスト:

ページ	何を保管しているか
プロフィール	すべての設定を保存および読み込みできます。新規作成、名前変更、上書き、削除が可能です。プロファイルをテキストとしてコピーまたは貼り付けできます。
メニュー	ウィンドウ、キー、メニューの色、オプションの背景。
ゲームパネル	プレビュー画面と再生画面。
コードエディタ	ハイライトとヒント。
ペイントエディター	ブラシ、パレット、マップ表示。
特殊効果エディター	シンセサイザーと音符入力。
データエディター	車輪の向きと色。
言語	メニューとヒントの言語。
ライセンス	Fusion PixelフォントおよびSDL3を含むライセンス条項。

グローバル：

- **レイアウト**— パネルの配置（上図参照）。
- **音量**— マスターレベル、0〜10。
- **キーバインド**— 8人のプレイヤーそれぞれに割り当てられたキー。キーを押してキャプチャします。方向キー、ABXYLR、選択キー。
- **表示**— アイドル状態のフレームをスキップするか、すべてのフレームを描画します。
- **フルスクリーン表示**- 枠なし。
- **更新**— ウィンドウのFPS。左右に60、90、120、最大というステップがあります。Enterキーを押すと、1000までの数値を入力します。0は上限なしを意味します。
- **Vsync** — フレーム間の表示を待機します。
- **スケール**— 整数ウィンドウ乗数。
- **シェーダー**（CRTフィルターなど）は、ゲームではなくエディター上で使用されます。
- **色**— メニューのテキストとハイライト。
- **背景**— メニューの背後に表示される、128ピクセルにトリミングされたオプションのPNG画像。クリアすると削除されます。
- **エディタカーソル**— ポインタの形状。
- **マウス**— フローティングポインター、またはピクセルにスナップしたポインター。

- **UI速度**— 押した方向キーがリスト内で繰り返される速さ。
- **エクスプローラー**では、ファイル一覧はコンピューターのルートディレクトリ、または実行ファイル（.exe）のあるフォルダーから始まります。



デフォルトのキーボードです。Aはメニューでの決定、Bは戻るです。ゲームでは、これらはボタン4と5に相当します。セレクト（S）はボタン10であり、メニューのショートカットではありません。

ボタン	役割	デフォルトキー
0	左	左矢印
1	右	右矢印
2	上	上向き矢印
3	下	下向き矢印
4	A、確認	V
5	B、背面	C
6	X	X
7	Y	Z
8	R	F
9	L	A
10	選択	S
11	始める	D

建てる

プラットフォームを選択してください：Windows、Linux、Mac、Android、iOS、またはWeb。

- **名前**— ビルドのファイル名。
- **作者**— WindowsとLinuxに対応。これはセーブデータの保存用フォルダ名です。
- **SteamアプリID** — WindowsおよびLinux用。空欄の場合はSteamworksなしを意味します。IDが入力されている場合は、 `achieve(0) ~ achieve(15)`がSteam実績ACH_0 ~ ACH_15にマッピングされます。
- **ID** — プラットフォームが要求するバンドル識別子。
- **コードを非表示にする機能**— デスクトップおよびモバイルのオーバーレイに表示されます。ゲームカートリッジの`peek`遅くしますが、ロック機能ではありません。
- **フルスクリーン**— エクスポートされたプレーヤーはフルスクリーンで起動します。
- **シェーダー**— プレイヤーに最大3つのレイヤーを焼き付ける。
- **.p37ファイルをダウンロード**— Web版のみ。一時停止メニューにダウンロード項目が表示されます。
- **アイコン**— スプライトシートからのタイル：シート、サイズ、x座標、y座標。または、切り抜いたPNGカバー画像。
- **エクスポートする**には、フォルダーを選択してください。メニューに「ビルド保存済み」または「ビルド失敗」と表示されます。

インターネット接続をオンにしてビルドを作成します。スプラッシュ画面では、このビルドのハッシュが既知かどうかを匿名でサーバーに問い合わせる場合があります。このチェックには 2 つの理由があります。1 つは、PIX37 の作者である私が、配布する権限のないツールのコピーから保護されるためです。もう 1 つは、ゲームを作成した人を保護するためです。他人のゲームを自分のものとしてリストアップすることは、Play (Android) や iOS などの携帯電話市場で一般的になりつつあります。それでもゲームは起動します。マシンがオフラインの場合、イントロにはゲームが接続されていないと表示されます。この行は、コピーが合法的なソースから入手されたものではない可能性があることをプレイヤーに示唆するものでもあります。これは小さなチェックなので、ゲームが盗まれ、同意なしに他人に販売された場合、Play (Android)、Steam、itch.io などのストアでその件を訴えることができます。サーバーは、ハッシュの横に、ハッシュが最初に登録された時刻を保存し、後で変更しません。より早い時刻は元のビルドです。それ以降に登録されたハッシュは、後のコピーです。

このチェックは、ゲーム開発者を保護するために導入したいのです。ゲームエンジンは既に多くの作業を自動化しており、これもその一つです。賛否両論あることは承知しています。もしコミュニティがこのマイクロDRMを廃止する方が良いと判断した場合、私もそうします。

以前のエディタで作成された古いゲームカートリッジは、言語変更後に起動しなくなることがあります。このエディタで`.pix37`プロジェクトを開き、新しい`.p37`をエクスポートしてください。

PIX37.PARTYに接続

PIX37.PARTYへの接続はアプリ内にあります。ブラウザにはログインページはありません。Steamセッションチケットまたはitch.ioを使用して、パッドとマウスでサインインします。サービスには、2〜16文字の固有の表示名が表示されます。ライセンスキーとSteam IDは他のプレイヤーには表示されません。

利用規約に続いて、ハブとなるのは「お気に入り」、「カタログ」、「ランキング」、「プレイヤー」、そしてあなたのプロフィールです。

開発者ライセンスはエディターです。ゲームを作成し、アップロードして、星評価を付けます。ゲーマーライセンスはプレイヤーです。ハートは、ランナーゲームをプレイしている人からの評価です。

- **カタログ**— ゲームを閲覧できます。フィルターは、ジャンル（最大3つ）、画面サイズ、ゲームカートリッジサイズ、アップロードに`.pix37`プロジェクトが含まれているかどうかで選択できます。
- **お気に入り**— マークしたゲーム。
- **アップロード**— 開発者ライセンスが必要です。.p37 ファイルを送信し、 `.p37`プロジェクトを添付できます`.pix37` `.p37`圧縮されたバイトコードです。読み取りが難しく、意図的に難読化されている場合があります`.pix37`は、実際に作業するプロジェクト ファイルです。ソース、画像、サウンドが、他の人が開いて学習できる形式になっています。これを送信することで、他のクリエイターは圧縮されたゲーム カートリッジではなく、ゲームを研究できます。タイトル、作者、ジャンル、カバー、アイコン、実績名も一緒に送信されます。送信される RAM の数値は、ゲーム カートリッジが到達した最大容量です。
- **コメントと実績は**ゲームページに表示されます。ログイン中に実績を解除すると、その情報はアカウントと端末の両方に保存されます。
- **ランキング**— 開発者一覧。星の数またはハートの数で並べ替えられています。各行には両方の数が表示されます。名前をクリックすると、その開発者のゲームがハートの数順に表示されます。
- **プレイヤー**— 解除された実績順に並べられたアカウント。

コード

この言語は、Luaスタイルの小さなスクリプトです。この前半では、空のプロジェクトから始めて、言語の使い方を解説し、各関数の簡単な使用例を示します。

最初のプログラム

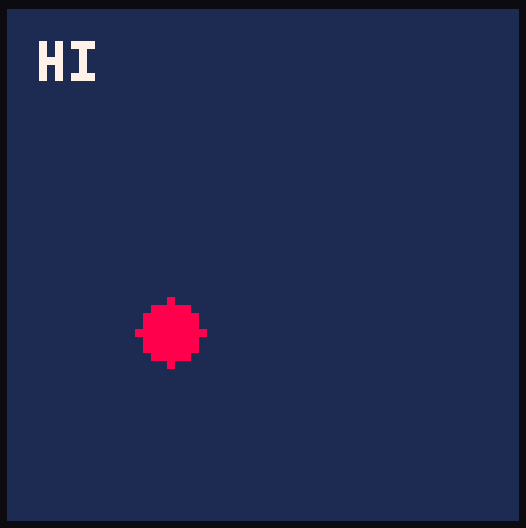
64×64ピクセル、4ビットカラーのプロジェクトを作成します。CODEを開きます。タブを以下のように置き換えます。

```
_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で起動します。矢印キーで円を移動します。Escキーでゲームを停止し、エディタに戻ります。



同じ３つの呼び出し： `cls(1)`は紺色の画面を描画し、 `print` HIを描画し、 `circfill`ピンク色の円を描画します。

何がいつ実行されるか

- `_init`一度だけ実行されます。
- `_update` 60Hzで動作します。ここに動きとボタンを配置してください。
- `_draw`描画します。fps 引数がない場合、コンピュータとモニターの処理速度に応じて描画します。_draw `_draw(30)`描画速度を 30 fps に制限し、ロジックは 60 fps のままにします。
- `_set_fps(n)`は後でその上限を変更します。0 を指定すると上限がクリアされます。

名前は大文字と小文字を区別しません。 `function`は小文字で保存します。function と`fun`は同じキーワードです。

言語

プログラミングの基礎

プログラムとは--ゲームカートリッジが実行する行のリストです。各見出しには、まず行の形式が示され、次に簡単な例が示されています。--で始まる行はコメント、つまりあなたへのメモです。ゲームはこの行をスキップします。

変数

変数とは、値を保持する名前のことです。名前は自分で決めます。後続の行はそれを読み取ることができ、同じ名前に新しい値を格納できます。

形状は`name = value`です。local `local name = value`使用すると、その名前は現在のブロック内に保持されるため、ゲームカートリッジの残りの部分からは見えません。local `local`使用しない場合、名前はグローバルになり、すべてのタブから見ることができます。

```
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`値がないことを意味します。true と`false`は`true`はいといいえを表します。一度も設定していない名前は`nil`と表示されます。

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

表

テーブルは、1つの名前の下に複数の値を保持します。リストは数値を使用します。最初の項目は0ではなく1です。レコードは名前を使用します。テーブルは別のテーブルを保持できます。

リストの形状: `{ value, value }` 。レコードの形状: `{ name = value, name = value }` 。レコードのリストは、項目がテーブルであるテーブルです。

```
-- 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
```

コンディションとボディ

条件とは、 `hp > 0`や`btnd(4)`のように、はいまたはいいえで答えられる質問です。本体とは、ブロックに入ったときに実行される行のグループです。if `if`すべてのループは、条件、本体、 `end`という同じ形式を使用します。repeat `repeat` `end`の代わりに`until`で閉じます。

もし

`if`文は条件が真の場合にのみ実行され、 `elseif`別の条件を試します。else `else`すべての条件が偽の場合に実行されます。

形状: `if condition then` 、本体、 `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
```

ループ

ループは本体を繰り返します。 `while`最初に条件をチェックし、条件が真である限り実行を続けます。`repeat`本体を実行してからチェックするため、本体は常に少なくとも 1 回実行されます。 `for`カウントします。最後の数値も含まれます。 3 番目の数値がない場合、ステップは 1 なので、開始がすでに終了を超えている場合は本体は実行されません。 カウントダウンするには負のステップが必要です。

```
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
```

テーブルを歩く

`all`リストの各項目を最初の項目から順に訪問します。 `pairs`各名前とその名前に格納されている値を訪問します。

```
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
```

機能

関数は名前付きのレシピです。構造は、 `function name(arguments)` 、本体、そして`end`なります。関数を呼び出すには、関数名と括弧を記述します。return `return`値を返しますが、複数の値を返すこともできます。引数は、その関数のローカル変数です。

```
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`とは、周囲のローカル変数を参照できる関数のことです。これはクロージャの一種です。テーブルはローカル関数を格納でき、ドットを使って呼び出します。

```
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
```

はい、いいえ、そして選択

条件を反転`not` 。 `and`両方の側が真の場合にのみ真となる。 `or`どちらかの側が真の場合に真となる。 `a and b or c` `a`が真の場合に`b`選択し、それ以外の場合は`c`を選択する。 `cond ? a : b`は、同じ考えを 1 行で表したもので、条件が真の場合は`a` 、そうでない場合は`b`である。

```
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)
```

数学とテキスト

`+ - * /`は加算、減算、乗算、除算です。`//` `//`除算して整数を保持します。 `%`は余りです。 `^`はべき乗です`..` は 2 つのテキストを結合します。 `#t`はリスト内の項目の数です。比較は`==` (同じ)、 `~=`または`!=` (異なる)、および`< > <= >=` 。x `x += 1`は x に 1 を加えます。- `-= *= /= ^= %=`にも同じ形式が適用されます。

```
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
```

地元住民が住んでいる場所

`local` 、そのブロックを閉じる`end`まで有効です。if / `end` `if`それ自体がブロックです。ローカル変数を設定する行より上にローカル変数を記述することはできますが、その行が実行されるまで値は`nil`です。

```
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
```

ジャンプ

`::label::`は行を示します。goto `goto label`そこにジャンプしますが、同じ関数内でのみ可能です。

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

メモリ、つまり

`peek`と`poke` 、1 つの 16 ビット アドレスを通してライブ ピクセルを確認します。ゲーム カートリッジの 2 つ目のコピーは保持しません。アドレス`0x5F00`から`0x5FFF`はレジスタです。カメラ、クリップ (読み取り専用)、描画色、塗りつぶしパターン、3 つのパレット リマップ、およびページ番号です。その穴の両側の 65280 バイトは、ピクセル ウィンドウです。

画面が 65280 バイトに収まる場合 (64×64 と 128×128 は収まりますが、256×256 は収まりません)、画面はアドレス 0 から行ごとに開始されます。残りの領域は、 `0x5F41`で選択されたスプライト シート (0 はシート 0) です。シートが残りの領域よりも大きい場合、 `0x5F42`ページを選択し、マップはウィンドウ内に表示されません。シートが収まる場合、末尾は`0x5F43`で選択されたマップのマップ セル (それぞれ 1 バイトまたは 2 バイト) で、末尾に収まらない場合は`0x5F44`マップをページングします。

画面が収まらない場合、ウィンドウ全体が 65280 ピクセルの 1 ページになります。poke `poke(0x5F40, n)`で画面のページが切り替わります。n が 0 以外の`poke(0x5F41, n)`では代わりにシート n が表示され、このモードではシート 0 は表示されません。n が 0 以外の`poke(0x5F43, n)`ではマップ n が表示され、シートよりも優先されます。このモードではマップ 0 は`mget`と`mset`のみ可能です。

ピクセルまたはレジスタに到達しない`peek`は0を返します。そこへの`poke` `fget`何も行いません。スプライト フラグ、SFX、およびDATAテーブルはこのウィンドウには表示されません。fget、 `sfxget` 、および`dget`使用してください。

実用

このページ下部に、PIX37の各機能とその実用的な使用例を一覧表示しています。各例の下にあるリンクをクリックすると、その機能に関する詳細情報が表示されます。



滑らかな動き

上から見た128×128ピクセルのゲームカートリッジ。スプライト1は既に描画済み。矢印はどの方向にも加速し、摩擦によって減速する。スプライトは折り返されており、下端から出る半分が上端に描画され、他の端も同様に描画される。

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



次の手がブロックされているかどうかを確認します

64×64のゲームカートリッジ。床と右側の壁は既にマップ上にあり、スプライト3で描画されています。そのスプライトにはフラグ0が設定されています。矢印はスプライト1を移動させます。上部の文字は、次のステップが実線のタイルと重なる場合は「ブロック」、重ならない場合は「フリー」と表示されます。

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



フリップ付きワイプ

Aボタンを押してください。ワイプは_draw()関数内ではなく、flip()関数によって黒い行が1行ずつ描画されます。_draw()関数はワイプの実行中は邪魔にならないように待機するため、各行が画面上に表示されたままになります。

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



関数一覧表（クラス）

スプライト1は既に描画されています。make_monsterはクラスのように動作します。テーブルを返し、updateとdrawは、そのモンスター自身のx座標とy座標を保持するメソッドです。上部の数値は、モンスターが歩いた距離です。

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



ボタンの読み方4選

Aはボタン4です。長押しするとチャージが増加します。離すとチャージは0に戻ります。長押し中は上下が繰り返されるため、メニュー行が動き続けます。btndは押下の最初のフレームで、このゲームカートリッジでは長押しの開始としてのみ使用されます。

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



マウスで射撃する

スプライト1は既に中央に描画されています。白い点はカーソルです。左クリックすると、その点に向かってピクセルが発射されます。ホイールは速度を変更し、その速度は隅に表示されています。

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



名前を入力して確認してください

Aボタンを押して入力欄を開きます。上下ボタンで点滅している文字を切り替えます。Aで文字を入力し、Xで削除、Yで空白、Bで確定します。すると、入力欄にあなたの名前が表示されます。

`btnd keyon keystr flr time cls print _update _draw`



プレーヤーにカメラを向ける

128×128のゲームカートリッジ。スプライト1はタイルの床の中央から始まります。矢印が動きます。カメラはスプライトを中央に維持します。引数なしのcamera()はHUDのビューを戻すため、隅のxとyはスクロールしません。

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



ビジュアルノベルの数行

テキストはDATA形式で、英語とポーランド語で表示されます。Aは続行、または選択肢が表示されているときに「留まる」を選択します。Bは選択肢が表示されている場合は「去る」を選択し、それ以外の場合は言語を切り替えます。Xは選択肢が表示されている間に言語を切り替えます。このダイアログは8pxフォントを使用しているため、ポーランド語の文字が表示されます。

`dstr dget btnd cls print8 print _update _draw`



覗き込みと突き出しでペイントする

64×64ピクセルのゲームカートリッジなので、アドレス0は左上のピクセルで、次の行は64から始まります。ゲームカートリッジはピクセルを書き込み、行をコピーし、行をクリアし、読み取った値を表示します。

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



追加できるリスト

Aは8までの数字を1つ追加します。Bは最後の数字を削除します。Xは最初の数字を削除します。リストは画面に下向きに表示されます。

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



名前を分解する

ゲームカートリッジは「ada,12,blue」を分割し、名前、その最初の文字、12、そして「blue」という単語を印刷します。

`cls split sub chr ord tonum tostr print _draw`

音を鳴らしてから、それを変える

パターン0は1音で構成され、再生されるたびに音量は同じです。Aは、その音が終わると再び再生します。Bは音程を上げて、その音を再生します。

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

ゲームカートリッジに書き込まれた曲

2つのパターンが0行目でループ再生され、それぞれ最初の2つのチャンネルで1つずつ再生されます。行は0のままです。ch0とch1は、各チャンネルが担当する音符です。ゲームカートリッジでパターン、LEN、LOOPを編集することで、ミュージックパネルでこれらのコントロールがどのように機能するかを確認できます。

`songget songset music musicstat cls print _init _draw`

データ内の3体のモンスター

データパネルには既に名前列とHP列があり、スライム、コウモリ、目が表示されています。Aはスライムにダメージを与えます。右側の数字はスライムのHPです。

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

スコアを覚えておいてください

Aはポイントを加算して保存する。ゲームを終了して再びカートリッジを開くと、その数字はまだ残っている。Bはそれを忘れてしまう。

`load btnd save delsave cls print _init _update _draw`

別のファイルにある第2章

2つのゲームカートリッジがセーブスロットを共有しています。パート1：Aはゲーム終了を示します。最初に開くパート2は、パート1を完了するように指示します。その後、パート2が開き、Bがセーブデータを消去してパート1を再度プレイできるようにします。

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

一つの成果

スプライト8は既に描画されています。Aボタンを10回押してください。トーストメッセージに実績名が表示され、スプライト8はロック解除されているためそのまま表示されます。Bボタンを押すと、ローカルファイルからその実績が消去され、カウントがリセットされます。

achdef btnd achieved achclear cls print spr _init _update _draw achieve



歩き回れる部屋

256×256のゲームカートリッジ。白い点がカーソルです。メニューはホスト、参加、またはサーバーリストです。ホストはwalkという名前の部屋を開き、ゲームを開始します。参加するとアドレス（数字、ドット、コロン）がスクロールし、Bが接続します。サーバーリストは上下に移動し、Aが行に参加します。ゲームでは、矢印がスプライト1を歩きます。コインは1枚共有されます。ホストがコインを動かし、それを拾うと、そのプレイヤーのスコアに1が加算され、次のプレイヤーが出現します。自分のスプライトを含め、すべてのスプライトの横にそのスコアが表示されます。隅には、参加しているプレイヤーの数、LANアドレス、ホストの公開アドレスが表示されます。Bが退出すると、あなたと他のプレイヤーはメニューに戻り、次のホストがスコア0で新しい部屋を開始します。

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



待つ対話

Aキーを押すと次の行が表示されます。関数は各yield地点で一時停止し、Aキーをもう一度押すまで停止しません。

yield cocreate btnd coresume cls print _init _update _draw



0.5秒待ってください

約0.5秒後、文字が「...」から「hi」に変わります。コルーチンが一時停止している間も、画面の描画は続きます。

yield cocreate costatus coresume cls print _init _update _draw



一つずつ到着する円

5つの円が上から2フレーム間隔で滴り落ち、そして落下する。

yield cocreate coresume all cls circfill _init _update _draw add



時計から切り取った2枚の画像

スプライト1とスプライト2は、同じキャラクターの2つのフレームです。画像は1秒間に4回切り替わります。

cls flr time spr _draw

すべての機能

角括弧内の引数は省略可能です。テーブルと文字列のインデックスは1から始まりますが、DATA行とDATA列は0から始まります。色はパレットのインデックスです。

索引

_draw
_init
_mp
_mp_fail
_mp_join
_mp_leave
_mp_update
_set_fps
_更新
腹筋
アククリア
アクデフ
達成する
達成した
追加する
すべて
主張する
あたん2
バンド
ない
ボル
ボタン
結び
btnp
ところで
ボクサー
カメラ
天井
chr
周回
サーフィル
カチッ
クリックしてください
クリッカー
クリップ
CLS
共創する
色
コアレジューム
コス
肋骨
数える
カーソル
dcols
デル
デリ
デルセーブ
ダイジェット
眠くなる
dset
dstr
ゲット
フィルプ
フリップ
フロリダ
フォーリーチ
fset
情報
アイペア
キーオフ
キーヨン
キー統計
キー文字列
ライン
負荷をかける
ログ
LSHR
地図
最大
memcpy
メムセット
ムゲット
半ば
分
マウス
ミリセット
音楽
ミュージックスタット
発表する
電話をかけない
次へ
nhost
参加する
NKB
nkbstr
んらん
残す
nlist
聞いてください
メモ
プレイヤー
送信する
スタン
命令
楕円形
オーバルフィル
パックする
ペア
友達
パルギブ
パルト
pcall
覗く
ピーク2
ピーク4

パフォーマンス
プゲット
突く
ポケ2
ポケ4
印刷する
プリント10
プリント8
プセット
長方形
レクトフィル
ランド
ロートル
ローター
正しい
正しいフィル
保存する
セーブスロット
画面
スクロール
選択する
セットマップ
セットパレット
セットPR
セテックス
セットターン
特撮
sfxget
sfxhdr
sfxset
sfxstat
sget
サイン
シュル
統計の表示
シュル
罪
ソングゲット
ソングセット
分割
春
平方メートル
スランド
セット
spr
ステータス
サブ
時間
トライン
トナム
トーストラ
開梱する
おいおい
収量

ライフサイクル

_init

`_init() ... end`

ゲームカートリッジの起動時に一度だけ実行されます。フックの外側にあるコードも_initに組み込まれます。

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

_update

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

ゲームロジック。_drawが制限されている場合やホストの処理速度が遅い場合でも、60Hzで呼び出されます。ボタンの読み取りはここで行います。

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

_draw

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

1フレームを描画します。引数なし、または0を指定すると、ホストが表示する頻度で実行されます。n（1〜999）を渡すと、描画を1秒あたりnフレームに制限できます。_updateは60Hzで動作します。

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

_set_fps

```
_set_fps(n)
```

ゲーム実行中の_drawの上限を変更します。0は上限なしを意味します。nは0〜999に制限されます。

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

flip

```
flip()
```

画面を表示し、次のフレームまで処理を一時停止します。独自のフレームループを描画する場合にのみ使用してください。この関数の後にビジーウェイトをしないでください。

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

time

```
time()
```

_initからの秒数。

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

入力

btn

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

ボタン b が押されている間は 1、それ以外の場合は 0。p はプレイヤーで、0〜7 です。引数がない場合、プレイヤー 0 のビット マスクを返します。ボタン: 0 左、1 右、2 上、3 下、4 A、5 B、6 X、7 Y、8 R、9 L、10 セレクト、11 スタート。

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

btnd

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

ボタンbが押されたときの_updateフレームでのみ真となります。

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

btnp

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

ボタンを押すと真になり、押したままにすると、30ティックの遅延の後、8ティックごとに真になります。ボタンが押されていない場合はマスクが返されます。

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

btnu

`btnu(b, [p])`

ボタンが解除されるのはフレーム上のみである。

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

mouse

`mouse()`

マウスカーソルのx座標とy座標をゲーム内のピクセル単位で返します。

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

clickl

`clickl([p])`

左マウスボタン。clickl() が押されています。clickl(1) は押下端です。

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

clickr

`clickr([p])`

マウスの右ボタン。clicklと同じ1引数のedge。

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

clickm

`clickm([p])`

中央のボタン、またはスクロールホイールのクリック。

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

scroll

`scroll()`

このフレームでは、マウスホイールの動きが反映されます。正負はホイールの回転方向によって決まります。

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

keyon

`keyon()`

ボタンで操作するテキストフィールドを開きます。上下キーでA～Zの文字を切り替えます。Aキーでその文字を入力します。Xキーで削除します。Yキーはスペースです。Bキーで確定してフィールドを閉じます。キーボードからの文字入力はできません。

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

keyoff

keyoff()

テキストフィールドを非表示にします。

```
keyoff()
```

keystr

keystr([s])

フィールドにテキストを入力し、フレームBの「1」で確認した後、文字の上下が変わります。フィールドが閉じると、その文字は空になります。keystr("") はテキストをクリアします。

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

keystat

keystat()

テキストフィールドが開いている間は1、それ以外の場合は0。

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

描画

screen

screen()

現在のゲームカートリッジの画面サイズ（ピクセル単位）。幅、次に高さを返します。これはゲームの解像度であり、エディタウィンドウの解像度ではありません。

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

cls

`cls([col])`

画面をクリアします。デフォルトの色は0です。

```
cls(1)
```

color

`color(col)`

後続の呼び出しでcolが省略された場合に使用される描画色を設定します。引数がない場合は、現在の色を返します。

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

camera

`camera([x, y])`

描画呼び出しごとにオフセットします。camera() は 0, 0 にリセットします。

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

clip

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

描画範囲を矩形に制限します。5番目の引数がゼロ以外の場合、現在のクリップ領域と交差します。clip() 関数は画面全体を復元します。クリップレジスタのピークは読み取り専用です。

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

cursor

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

座標なしの次の出力先と、オプションで色を設定します。 cursor() は x、y を返します。

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

pset

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

画面上のピクセルを1つ設定します。ピクセル統計にカウントされます。

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

pget

`pget(x, y)`

画面上のx、y座標にあるピクセルの色。

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

print

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

3×5のテキスト（UTF-8、Fusionフォントから切り抜いたもの）を描画します。x、y座標が指定されていない場合は、カーソル位置に印刷し、次の行に移動します。

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

log

`log(...)`

引数をスペースで区切ってテキストバッファに追加します。pix37_run はそのバッファの内容を標準出力に表示します。描画は行いません。

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

print8

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

印刷物と同じで、8ピクセルのFusionフォントを使用しています。

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

print10

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

印刷物と同じで、10ピクセルのFusionフォントを使用しています。

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

line

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

x0,y0からx1,y1への線。2つの引数は前の終点から継続します。line()はチェーンをリセットします。

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

circ

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

円の輪郭。半径はピクセル単位です。

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

circfill

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

塗りつぶされた円。

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

oval

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

境界ボックスの内側に楕円の輪郭が描かれている。角も含まれる。

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

ovalfill

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

境界ボックス内に塗りつぶされた楕円。

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

rect

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

長方形の輪郭。角も含む。

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

rectfill

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

塗りつぶされた長方形。

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

rrect

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

角丸長方形のアウトライン。x、yは左上隅の位置です。wとhはピクセル単位のサイズで、どちらも0より大きい値でなければなりません。rは角の半径で、デフォルト値は0です。短い方の辺の半分に制限されます。

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

rrectfill

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

塗りつぶされた角丸長方形。引数はrrectと同じです。

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

fillp

`fillp([p])`

後で塗りつぶす図形のための16ビット塗りつぶしパターン。0は塗りつぶしです。fillp()は現在のパターンを返します。ビットは4×4のパターンに沿って移動します。

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

pal

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

色 c0 を c1 に再マッピングします。モード 0 は描画パレット、1 は表示パレット、2 は塗りつぶしパターンパレットです。pal() はリセットします。

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

palrgb

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

パレットエントリを1つRGBに設定し、各チャンネルを0〜255に設定します。

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

palt

`palt([col, t])`

透明度。palt(col) または palt(col, true) は col を透明にします。palt(col, false) は不透明にします。palt() はすべての色を不透明にします。

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

setpalette

`setpalette(n)`

パレットバンクnを選択します。setpalette()は現在のバンクを返します。

```
setpalette(1)
```

tline

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

マップ位置 mx、my からマップピクセルをサンプリングしながら、画面上に線を描画します。mdx、mdy は、線に沿ってマップサンプルをステップします。

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

スプライトとマップ

spr

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

x、y の位置にスプライト n を描画します。w と h はピクセル単位のサイズで、デフォルトは 8 です。fx と fy は、0 以外の場合に水平方向と垂直方向に反転します。tex はスプライト シートのタブです。rot は回転角度 (0 ~ 1) で、sin と cos と同じ方向です。0 を指定すると、スプライトは回転しません。

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

sspr

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

スプライトシートから長方形を画面にコピーし、縦横比dw×横dhに拡大縮小します。デフォルトでは元のサイズが維持されます。

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

sget

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

スプライトシート上の1テクセルの色。

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

sset

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

スプライトシートに1テクセルを書き込みます。

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

fget

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

スプライトフラグ。fget(n)はフラグバイトを返します。fget(n, f)は、フラグビットf（0〜7）がセットされている場合にtrueになります。

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

fset

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

スプライトフラグを設定します。fset(n, v) はバイト全体を書き込みます。fset(n, f, v) は1ビットを設定します。

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

setspr

`setspr(n)`

アクティブなスプライトシートを選択します。setspr() は現在のシートを返します。settex と同じバンクです。

```
setspr(1)
```

settex

```
settex(n)
```

setsprの別名。

```
settex(0)
```

map

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

マップを描画します。マップのセル cx、cy は画面 sx、sy に描画されます。cw、ch が 0 の場合はマップ全体を意味します。layer は受け入れられ、無視されます。tex はタイルセットシートです。

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

mget

```
mget(cx, cy)
```

マップセルcx、cy内のスプライトID。

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

mset

```
mset(cx, cy, n)
```

スプライト n をマップセル cx、cy に書き込みます。

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

setmap

`setmap(n)`

マップnを選択します。setmap()は現在のマップを返します。マップ0は実際のマップです。

```
setmap(1)
```

メモリ

peek

`peek(addr)`

64 KB ウィンドウの 1 バイトを読み取ります。ウィンドウはライブ画面、次にスプライト ピクセル、画面が 65280 バイトに収まる場合はマップ セルです。より大きな画面は poke(0x5F40, page) でページングされます。マップされていないアドレスは 0 を返します。また、@addr にも書き込まれます。

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

poke

`poke(addr, ...)`

アドレス addr にバイト列を書き込みます。追加引数を指定すると、次のバイト列が書き込まれます。マッピングされていないアドレスは無視されます。また、$addr、val にも書き込まれます。画面への書き込みはピクセル書き込みとしてカウントされます。

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

peek2

`peek2(addr)`

アドレスにある符号なし16ビット値、リトルエンディアン。

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

poke2

poke2(addr, val)

符号なし16ビット値をリトルエンディアンで書き込みます。

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

peek4

peek4(addr)

符号なし32ビット値、リトルエンディアン。

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

poke4

poke4(addr, val)

符号なし32ビット値をリトルエンディアンで書き込みます。

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

memcpy

memcpy(dst, src, len)

同じ64KBのウィンドウ内にlenバイトをコピーします。

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

memset

memset(dest, val, len)

dest から始まる len バイトを val で埋めます。

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

数学

abs

`abs(x)`

絶対値。

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

flr

`flr(x)`

マイナス無限大の方向に四捨五入します。

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

ceil

`ceil(x)`

プラス無限大の方向に丸めます。

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

max

`max(a, b)`

2つの値のうち大きい方。

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

min

`min(a, b)`

2つの値のうち小さい方。

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

mid

`mid(a, b, c)`

3つの値の中間値。クランプとして便利です。

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

sgn

`sgn(x)`

xの符号：-1、0、または1。

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

sqrt

`sqrt(x)`

平方根。負の引数は数ではありません。

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

sin

`sin(x)`

サイン。xは回転数で表され、0、0.25、0.5、0.75が円を一周します。ラジアンではありません。

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

cos

`cos(x)`

コサインは、サインと同じ慣習に従う。

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

atan2

`atan2(dx, dy)`

ベクトルdxの角度dyを0から1まで変化させたもの。

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

rnd

`rnd([x])`

[0, x) の範囲の乱数。デフォルトの x は 1 なので、rnd() は 0..1 になります。サイコロの場合は、flr(rnd(6))+1 です。

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

srand

`srand(seed)`

乱数発生器のシード値を設定することで、後続の乱数シーケンスを繰り返すことができるようにします。

```
srand(1)
```

band

`band(a, b)`

ビットごとのAND演算。a & b とも表記されます。

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

bor

`bor(a, b)`

ビットごとの論理和。a | b とも表記されます。

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

bxor

`bxor(a, b)`

ビットごとの XOR。^^ b とも表記されます。

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

bnot

`bnot(a)`

ビットごとの否定。~a とも表記されます。

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

shl

`shl(a, n)`

左にシフトします。a << n とも表記されます。

shr

`shr(a, n)`

右シフト演算。a >> n とも表記される。

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

lshr

`lshr(a, n)`

論理シフト右。>>> n とも表記される。

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

rotl

`rotl(a, n)`

ビットを左に回転させます。また、<<> n とも表記されます。

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

rotr

`rotr(a, n)`

ビットを右に回転させます。>>< n とも表記されます。

表と値

add

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

テーブルtにvを追加します。iが指定されている場合は、その1から始まるインデックスに挿入します。

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

del

`del(t, v)`

vと等しい最初のエントリを削除します。

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

deli

`deli(t, [i])`

インデックス i を削除し、その値を返します。i が指定されていない場合は、最後のエントリを削除します。

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

all

`all(t)`

配列の各部分を順番に走査するイテレータ。1から始まる配列の終端と同じように、穴があると走査が停止します。

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

foreach

`foreach(t, fn)`

配列の各要素に対してfn(v)を呼び出します。

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

pairs

`pairs(t)`

名前付きフィールドを含む、すべてのキーと値を反復処理します。

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

ipairs

`ipairs(t)`

1から始まるインデックスを使用して配列部分を反復処理します。

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

next

`next(t, [k])`

k の次のキーと値を返します。next(t) は先頭から始まります。ペアで使用されます。

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

count

`count(t, [v])`

配列部分の長さ。vが指定されている場合は、vに等しいエントリの数をカウントします。

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

pack

`pack(...)`

引数を新しいテーブルにまとめます。

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

unpack

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

t[i]からt[j]までを個別の値として返します。デフォルト値は配列全体を対象としています。

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

select

`select(i, ...)`

select("#", ...) は、続く引数の数を返します。select(i, ...) は、i 以降の引数を返します。

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

assert

`assert(c, [msg])`

cがfalseまたはnilの場合、エラーで停止します。msgはデフォルトのテキストを置き換えます。

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

pcall

`pcall(f, ...)`

f関数を呼び出します。成功した場合はtrueと結果を返します。エラーが発生した場合はfalseとエラーメッセージを返します。

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

弦楽器

chr

`chr(n)`

バイトコード0〜255の1文字。

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

ord

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

インデックス i から j までの s のバイトコード。インデックスは 1 から始まります。デフォルトは最初の文字です。

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

sub

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

iからjまでの部分文字列を1から始まる順序で取得する。末尾を取得する場合はjを省略する。

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

split

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

s をテーブルに分割します。sep のデフォルトはカンマです。数値 sep はチャンクの幅を表します。conv が true の場合、tonum を使用してピースを変換します。

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

tonum

`tonum(s, [base])`

数値を解析します。基数は2〜36です。テキストが数値でない場合はnilを返します。

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

tostr

`tostr(v, [digits])`

値をテキストに変換します。digits は数値の小数点以下の桁数を設定します。

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

オーディオ

sfx

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

パターン n を再生し、チャンネルを返します。ch を省略するか、-1 を渡すと、最初に空いているチャンネルが選択されます。ch -2 は、すべてのチャンネルでパターン n を停止します。n -1 は、1 つのチャンネルを停止します。ch を省略すると、すべてのチャンネルが停止します。n -2 は、ループするパターンをループせずに終了させます。引数なしの sfx() は -1 を返します。

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

sfxget

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

1 つのパターン ステップのテーブルを返します: [1] ピッチ、[2] 波形、[3] ボリューム、[4] エフェクト。ピッチは 0 〜 63 (C-0 から D#5) です。波形は 0 〜 15 です。ボリュームは 0 〜 7 で、0 は無音です。エフェクトは 0 〜 7 です。

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

sfxset

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

パターン手順を1つ記述します。6つの引数はすべて必須です。

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

sfxhdr

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

パターン ヘッダー。sfxhdr(n) はヘッダー テーブルを返します。sfxhdr(n, key, val) は、speed、loop_in、loop_out、level、noise、crush、dtune、rverb、dampn、swing、steps の 1 つのフィールドを設定します。数値 0～10 は、同じフィールドをこの順序で選択します。speed は TIC、つまりステップあたりのティック数です。値が大きいほど遅くなります。15 は 120 拍/分です。16 ビットは 1〜255 を受け入れます。より狭いシンセは、TIC リストにスナップします。

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

sfxstat

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

パターンIDはチャンネルchで再生されます。指定しない場合は-1を指定します。デフォルトのチャンネルは0です。

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

note

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

パターンなしで1つの音を再生します。ピッチ0〜63、波形0～15、ボリューム0〜7、エフェクト0〜7。ティックは、TICと同じ120Hzのティックで音を保持する時間です。ティックを省略すると、デフォルトの1ステップである15が持続します。エフェクトはその時間実行され、その後チャンネルは静かになります。

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

music

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

曲 n を最初から再生します。music(-1) で停止します。既に再生中の ID で music を呼び出すと、再生が継続されます。music() は再生中の曲 ID または -1 を返します。fade はミリ秒です。曲がフェードインするか、n が -1 の場合はフェードアウトします。設定されている mask ビットは、曲に使用されるチャンネルです。0 は 8 個すべてを意味します。その他のチャンネルは効果音用に空いています。

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

musicstat

`musicstat()`

再生中の曲のテーブル。song と row は曲 ID と行番号、または何も再生されていない場合は -1 です。step と pos は 8 つの数値です。step[1] はチャンネル 0、step[8] はチャンネル 7 です。step は音符のインデックス、またはそのチャンネルが無音の場合は -1 です。pos はその音符が 0 から 1 までどれだけ進んだかです。tick はこの行が既に再生されたティックの数です。played は再生中の行を含め、music() 以降に開始された行の数です。

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

songget

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

パターンIDは曲のセルに格納されます。chは0〜7です。255はセルが空であることを意味します。これは実際の呼び出しです。コードヒントテキストには、古い引数の順序が表示されています。

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

songset

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

パターンIDを楽曲セルに書き込みます。セルをクリアするには255を使用してください。

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

stat

`stat(n)`

音声のみ。16〜23はチャンネルn-16のsfx IDです。24〜31はチャンネルn-24の音楽パターンです。その他のnは0を返します。フレームタイミングとRAMはパフォーマンスと情報であり、統計情報ではありません。

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

データテーブル

dget

`dget(t, row, col)`

データパネルからデータテーブルtのセルを読み取ります。行と列は0から始まります。

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

dset

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

セルに書き込みます。テキスト列には文字列を使用してください。

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

dstr

`dstr(t, row, col)`

セルはテキストとして表示されます。数値は変換されます。

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

drows

`drows(t)`

テーブルtには何行ありますか。

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

dcols

`dcols(t)`

テーブルtにはいくつの列がありますか。

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

セーブと実績

save

`save(k, v)`

保存ファイル内のキー k の下に v を格納します。v は数値、文字列、またはそれらの組み合わせのテーブルです。ファイルは圧縮されます。

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

load

`load(k)`

キーkを読み取ります。キーが存在しない場合はnilを返します。

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

delsave

`delsave(k)`

セーブデータからキー「k」を削除します。

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

saveslot

`saveslot([name])`

作者フォルダ内のセーブファイルを選択します。名前が指定されていない場合は、ゲームタイトルが該当します。ファイルが既に存在する場合は1を返します。

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

achieve

`achieve(i)`

実績i（0〜15）を解除します。ローカルに保存されます。SteamビルドではACH_iも設定されます。カタログセッションは、それをプレイヤーのアカウントに報告します。

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

achieved

`achieved(i)`

1. 実績 i が既に解除されている場合。

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

achdef

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

実績 i のトーストテキストが画面下部から 15 秒間表示されます。spr はオプションの 8×8 タイルで、そのシートを tex します。

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

achclear

`achclear(i)`

ローカルファイル内の実績iを削除します。既に設定されているSteamのACH_iはそのまま維持されます。

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

ネットワーク

whoami

`whoami()`

セッション中は、プレイヤーID（0〜15）が設定されます。オフライン時は-1になります。

```
id = whoami()
```

nhost

`nhost()`

このマシンがホストの場合は1、それ以外の場合は0。

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

nplayers

`nplayers()`

0〜15の範囲にいくつの有効なIDが存在しますか。

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

nlisten

`nlisten()`

UDPのホスティングを開始し、LANアドレスをip:portの形式で返します。

```
addr = nlisten()
```

njoin

`njoin(addr)`

IPアドレスまたはIPアドレスとポート番号の組み合わせで参加します。

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

nleave

`nleave()`

セッションから退出します。ピアは理由0とともに_mp_leaveを確認します。

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

nsend

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

情報が失われるメッセージを送信します。名前はイベントです。最大4つの追加値が続きます。p（0〜15）を指定すると、そのピアにささやき送信します。pを省略すると、全員に送信します。

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

ncall

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

nsendと同じ形式ですが、メッセージは確認応答があるまで再試行されます。3回連続で失敗するとセッションが終了します。

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

nlan

`nlan()`

nlisten実行後の現在のLAN IPアドレスとポート番号。

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

nstun

`nstun()`

STUNから取得したあなたのパブリックIPアドレスとポート番号です。別のネットワーク上のゲストが接続を試みることができます。

```
pub = nstun()
```

setturn

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

直接パスが失敗した場合のフォールバックとして TURN を使用します。引数を空にして setturn() を実行すると、この機能は無効になります。アドレスは host:port の形式です。

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

nlist

`nlist(game)`

そのゲーム名のオープンセッション一覧表。各項目にはアドレス、プレイヤー数、ゲーム名が記載されています。

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

nannounce

`nannounce(game)`

ハートビート機能により、このホストがゲームのnlistに表示されるようになります。ホスト中はタイマーで呼び出してください。

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

nkb

`nkb([on])`

アドレスフィールド。nkb(1)で開き、nkb(0)で閉じます。nkb()で開いているかどうかを確認できます。上下キーで文字、0～9、ドット、コロンを変更できます。Aで書き込み、Xで削除します。キーボードから文字を入力することはできません。

```
nkb(1)
```

nkbstr

`nkbstr([s])`

アドレス欄のテキストは、文字の上下が変化する。nkbstr("") はテキストをクリアする。

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

_mp

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

この関数を記述します。メッセージが到着すると実行されます。from は送信者 ID、name はイベントです。その後に最大 4 つの値が続きます。

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

_mp_join

`_mp_join(p)`

この関数を記述します。ピアはID pを受け取ります。この関数はあなたにも適用されます。whoami()はそのIDです。

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

_mp_leave

`_mp_leave(p, why)`

この関数を作成します。ピア p が退出します。0 は正常な退出です。1 は接続が切断されたことを意味します。

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

_mp_update

`_mp_update()`

この関数はあなたが作成します。15回の_updateフレームごと、つまり約4Hzの間隔で実行されます。位置情報を送信するのに適した場所です。

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

_mp_fail

`_mp_fail()`

この関数を記述します。セッションは終了しています。ホストが消失したか、ncallが3回失敗したかのいずれかです。

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

コルーチン

cocreate

`cocreate(f)`

関数fからコルーチンを作成します。コルーチンはcoresumeが実行されるまで実行されません。

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

coresume

`coresume(c, ...)`

コルーチンcがyieldまたはreturnするまで実行します。成功した場合：trueとyieldされた値を返します。エラーが発生した場合：falseとエラーメッセージを返します。

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

costatus

`costatus(c)`

「停止中」、「走行中」、または「死亡中」。

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

yield

`yield(...)`

coresumeを呼び出したユーザーに制御権を戻します。これらの値は、そのcoresumeの実行結果となります。

```
yield(1)
```

測定

info

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

_draw でライブ HUD を描画します: fps (1 秒平均)、upd、drw、px、max、ram、peak、ramc、およびセッションが起動しているときの net。info() はプリント カーソルを使用します。info(col) は色のみを設定します。info(x, y) または info(x, y, col) はそれを配置します。数値は、小さな画面に収まるように B、k、または m を使用します。呼び出し自体は px にはカウントされません。

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

showstats

`showstats([on])`

infoと同じHUDで、_drawの後に0, 0の位置に自動的に描画され、カメラとクリップは無視されます。showstats(0)で非表示になり、showstats(1)またはshowstats()で表示されます。

```
showstats(1)
```

perf

`perf(n)`

HUDの数値は1つなので、自分で描画できます。0 fps、1 upd、2 drow、3 px、4 max（この秒で最も重いフレーム）、5 ram、6 ramc（RAMとパックされたゲームカートリッジ）、7 ゲームカートリッジ開始からのピークRAM。

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

PIX37のマニュアルです。カミル・スタニスワフスキー。