Fantasy Computer

TIC-80 ビギナーズガイド

TIC-80 とは？

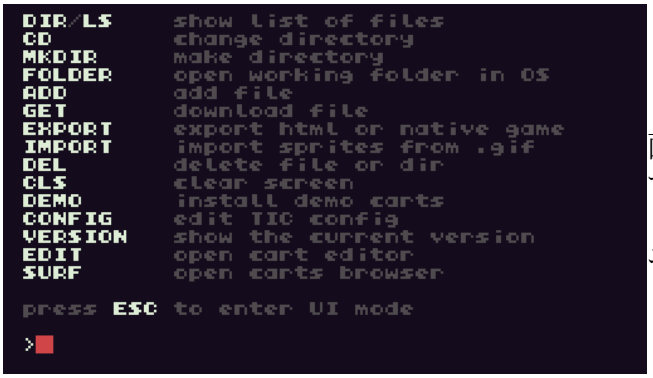
TIC-80 は小さなゲームを作ったり遊んだりするためのキッズパソコンのような架空のコンピュータ(ソフトウェア)です。必要かつ十分な開発用ツールがオールインワン(コマンドコンソール、コードエディタ、スプライトエディタ、マップエディタ、効果音エディタ、音楽エディタ)で組み込まれており、TIC-80 とやる気さえあればすぐにゲームの開発を始めることができます。

TIC-80 は、教育目的、あるいは個人のホビーとして、小規模でレトロスタイルなゲームを作れるように、240×136 ドットサイズの画面、16 色のパレット、4 和音しかない音楽などなど、あえて制約された仕様で設計されています。これは一人あるいは少人数で開発するには理想的なバランスの制限となっています。

そして、何より素晴らしいことに TIC-80 は MIT ライセンス(巻末参照)で公開されており、誰でも無料で自由に利用することができるのです。

TIC-80 の開発用ツール

コマンドコンソール(Console)



TIC-80 を起動すると最初にこのコンソール画面が開きます。作成中のプログラムのセーブやロード、実行などを行ない、SURF というコマンドで Web 公開されている他のユーザの作品を遊ぶこともできます。

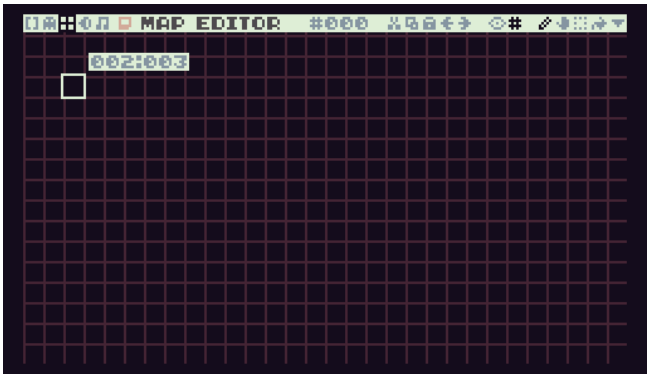
コードエディタ(Code Editor)

Lua というプログラム言語をベースに TIC-80 独自の API 命令を組み合わせてプログラミングします。プログラムコードの最大容量は 64KB(65536 バイト)です。

スプライトエディタ(Sprite Editor)

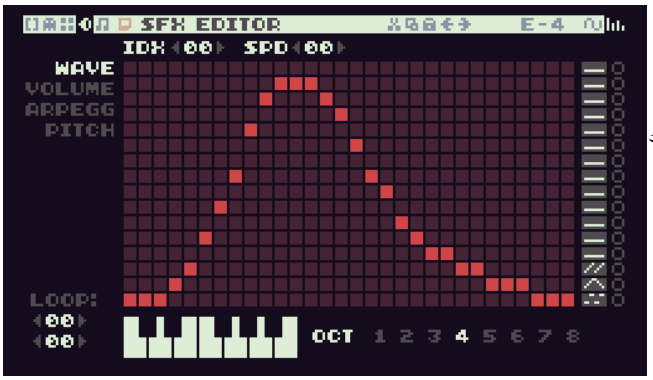
16 色パレットで 8×8、16×16、32×32、64×64 ドットいずれかのサイズでキャラクタを描けるエディタです。8×8 ドットのキャラクタなら 512 個も作成できます。

マップエディタ(Map Editor)



ドットエディタで作成したキャラクタをならべてゲームの背景となるマップを描くことができます。広さは 8×8 ドットのキャラクタなら横に 240 個、縦に 136 個分。

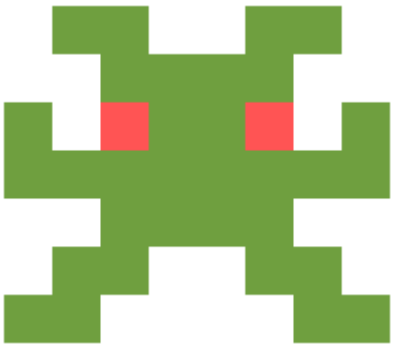
効果音エディタ(SFX Editor)



音の波形、ボリューム、アルペジオ、ピッチなどを編集して色々な効果音を作れます。

音楽エディタ(Music Editor)

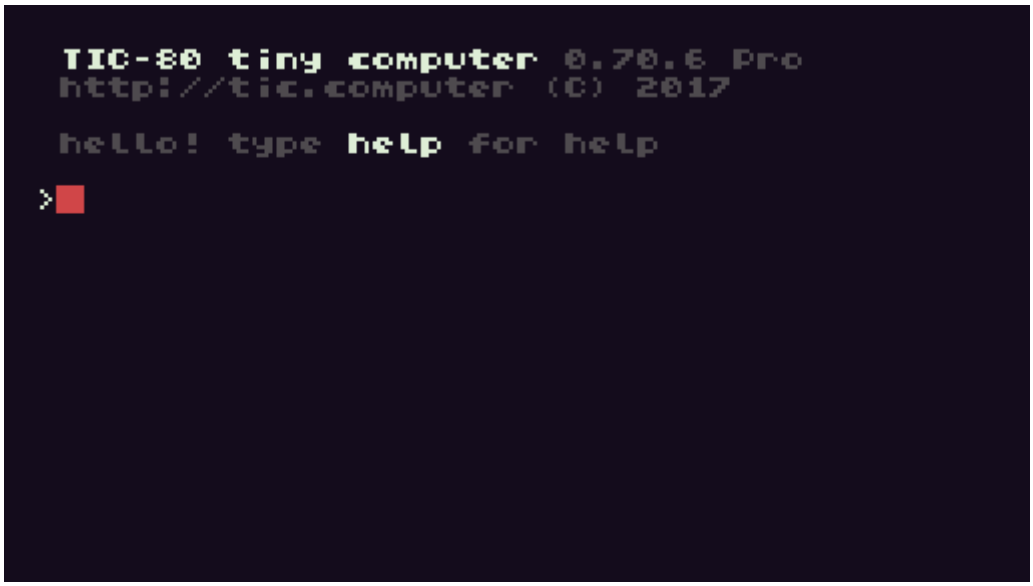
8オクターブ、64種の音色、4つのチャンネル(和音)を使って作曲します。1つのプログラム毎に8トラック分の曲を保存することができます。



TIC-80 クイックスタート

TIC-80 の起動

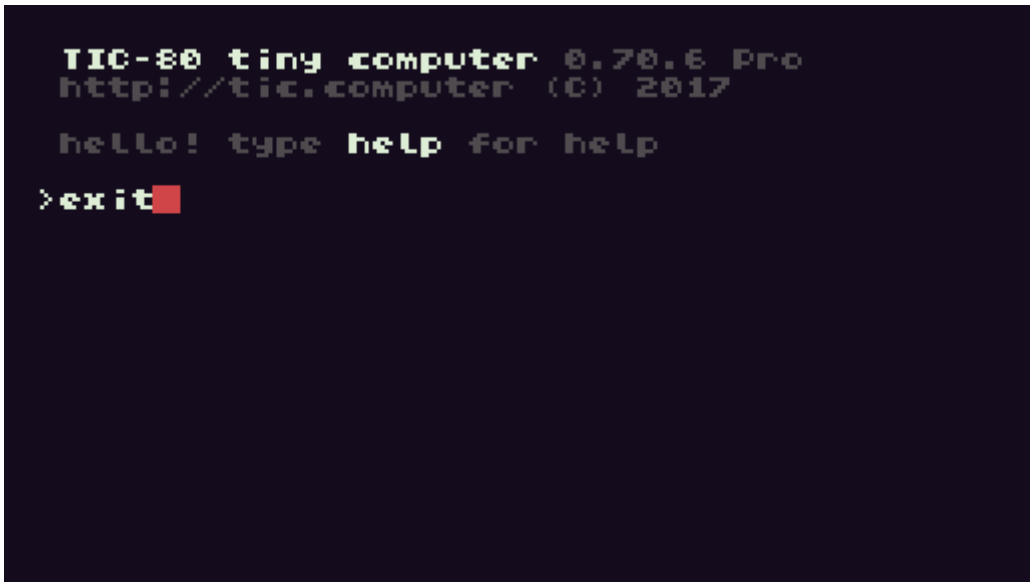
デスクトップのショートカットから TIC-80 アイコンまたは「tic80」実行ファイルをダブルクリックすると最初に次のようなコンソール画面がたちあがります。



[F11]キーでいつでもフルスクリーンモードとウィンドウモードを切り替えることができます。

TIC-80 の終了

コンソール画面で EXIT もしくは QUIT と入力し、[Enter]キーを押すと終了します。ウィンドウ画面右上の[×]ボタンでも同じように終了できます。

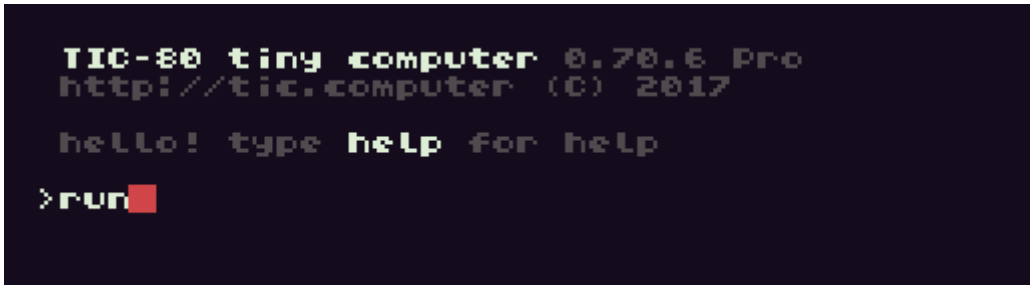


プログラムの実行 - コンソール

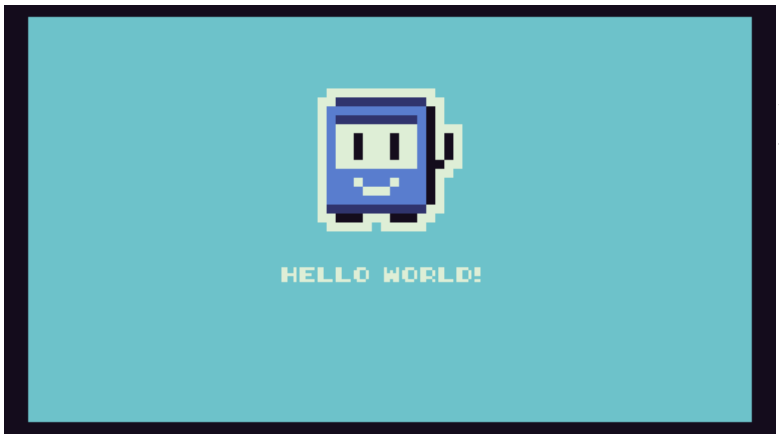
TIC-80 を起動してコンソール画面の状態から[ESC]キーを押すと、プログラムを入力するコードエディタの画面に切り替わります。するとなぜか、最初からコードエディタ上には既にプログラムが書かれています。

実はTIC-80では、起動時には自動的に「HELLO WORLD!」プログラムが読み込まれるようになっているのです。

ここでもう一度[ESC]とし、コンソール画面上でRUNコマンドを打ってみましょう。



そうすると、現在読み込まれている「HELLO WORLD!」プログラムが実行されます。



カーソルキーを押すとTIC-80のアイコンキャラを上下左右に移動できます。[ESC]を押すとプログラムを終了できます。

プログラムの変更 - コードエディタ

もう一度[ESC]キーを押してコードエディタに切り替え、プログラムの一部を変更してみましょう。「HELLO WORLD!」の部分を好きなメッセージに変えてみてください。

例「Welcome to this Happy time!」

[ESC]でコンソールに戻ってRUNコマンドを実行します。プログラムは変更された通りに実行されたでしょうか？

メイン関数「TIC」について

毎秒60回、自動的に呼び出される、TIC-80のプログラム実行の中心になる部分。それがプログラム内に必須の唯一の関数「TIC()」です。TIC関数は function TIC() で始まり end で終わります。

```
function TIC()

    -- ここに好きなようにプログラムを書いてねん

end
```

function TIC() と end の間の行にあなたの書きたいプログラムを入れればそのプログラムが1秒間に60回、自動的に呼び出されて実行されます。

毎回呼び出す必要がないプログラムや自分で定義する別名の関数はTIC関数の外側に書いてかまいません。ただしfunction TIC() と end の間に何も書くことがなくてもTIC関数は一つのプログラム内に必ず書くことになっているので忘れないでください。それがTIC-80のルールです。

カートリッジメタデータ

行の最初にハイフンを2つ「--」付けるとコメント文となります。さて、1〜4行目に書いてあるコメント文は何でしょう？

```
-- title:  game title
-- author: game developer
-- desc:   short description
-- script: lua
```

この4行はメタデータと呼ばれる部分で、先頭に「--」があるようにコメント文なのでプログラムの実行そのものに影響はありません。別に書かなくてもエラーにはならないのですがTIC-80の公式サイトに作品登録(アップロード)する際にこのメタデータの部分が自動的に参照されWebページに紹介されるしくみになっています。なので作品公開を考えている人は書いておくようにしましょう。記述する内容は次の通りです。

```
-- title: ゲームタイトル
-- author: 作者名
-- desc: 短い説明
-- script: 言語の種類
```

ドット絵を描く - スプライトエディタ

コンソールから[ESC]キーを押してエディタ画面に切り替え、[F2]キーを押すか、マウスで左上のスプライトエディタアイコンをクリックします。

キャンバスを16×16ドットに拡大し、選択範囲をTIC-80マスコットキャラにあわせて、ドット絵を描き変えてみてください。

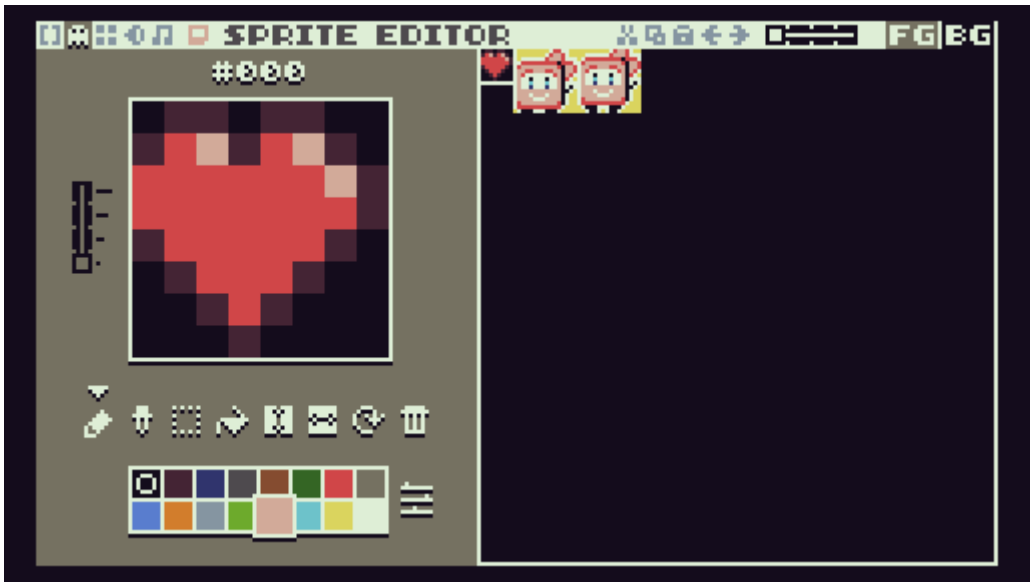
描けたら[ESC]でコンソールに戻り、RUNコマンドで実行してみましょう。

実行画面

プログラムは変更された通りに実行されましたか？

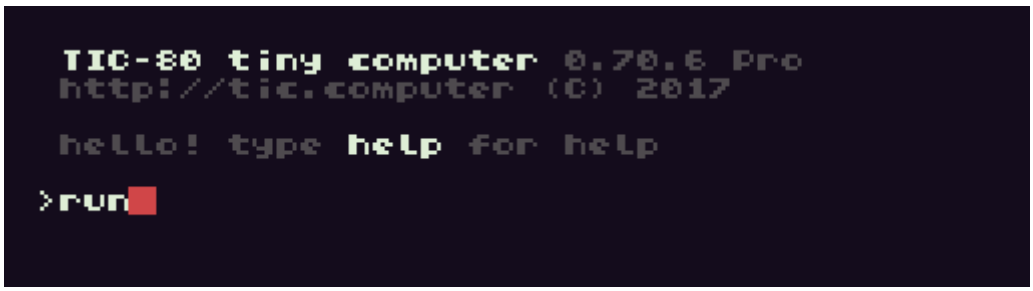
背景マップを表示してみよう - マップエディタ

スプライトエディタで、スプライト番号0の位置に背景にするドット絵を描きます。キャンバスは8×8ドットでいきましょう。



コードエディタで、17行目の「cls(13)」を「map()」に変更します。

[ESC]でコンソールに戻り、RUNコマンドで実行。



プログラムは変更された通りに実行されましたか？

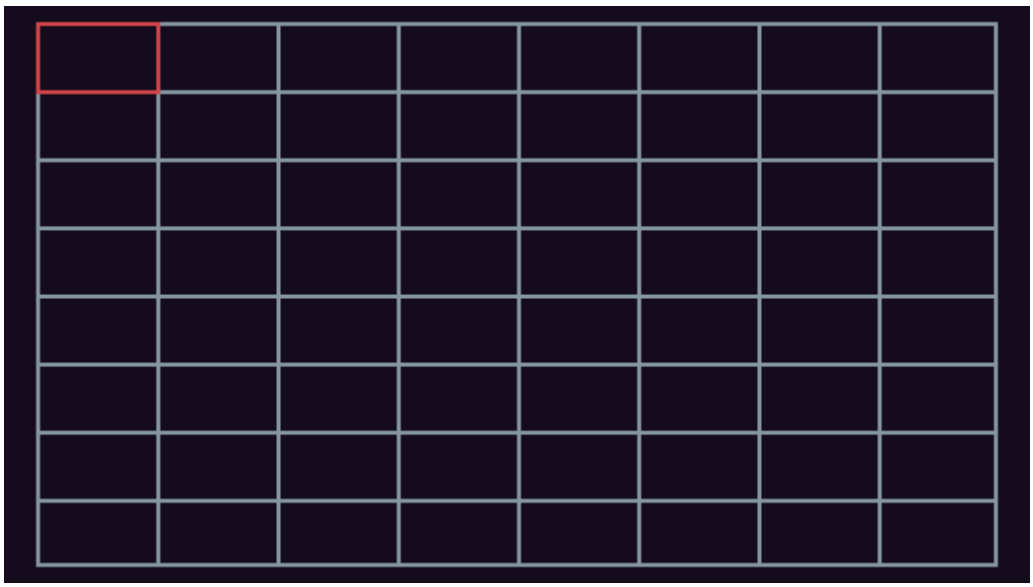
変更前の「cls(13)」はパレット 13 番の色で画面全体をクリアする命令でした。変更後の「map()」は丸カッコ内に何も指定しない場合はマップの座標(0,0)から 1 画面分を表示する初期設定になっています。

[ESC]を押してプログラム終了し、コンソール画面でもう一度[ESC]を押してエディタ画面に切り替えます。そこで[F3]キーを押すか、マウスで左上のマップエディタアイコンをクリックします。

先程、背景に表示されていたのはこの画面。これがマップエディタです。

ここで[TAB]キーを押すか、ワールドマップアイコン(目のアイコン)をクリックしてみてください。

マップエリア全体を見わたすことができます。



左上の囲いが赤くなっているところが現在選択されているエリア。ここでエリアをクリックして選択し、スプライト(ドット絵)を配置していけば、広大なマップを作成することができます。

スプライトエディタでさらにいくつかのドット絵を描いてマップエディタで1番左上のマップエリアに好きなように配置してください。

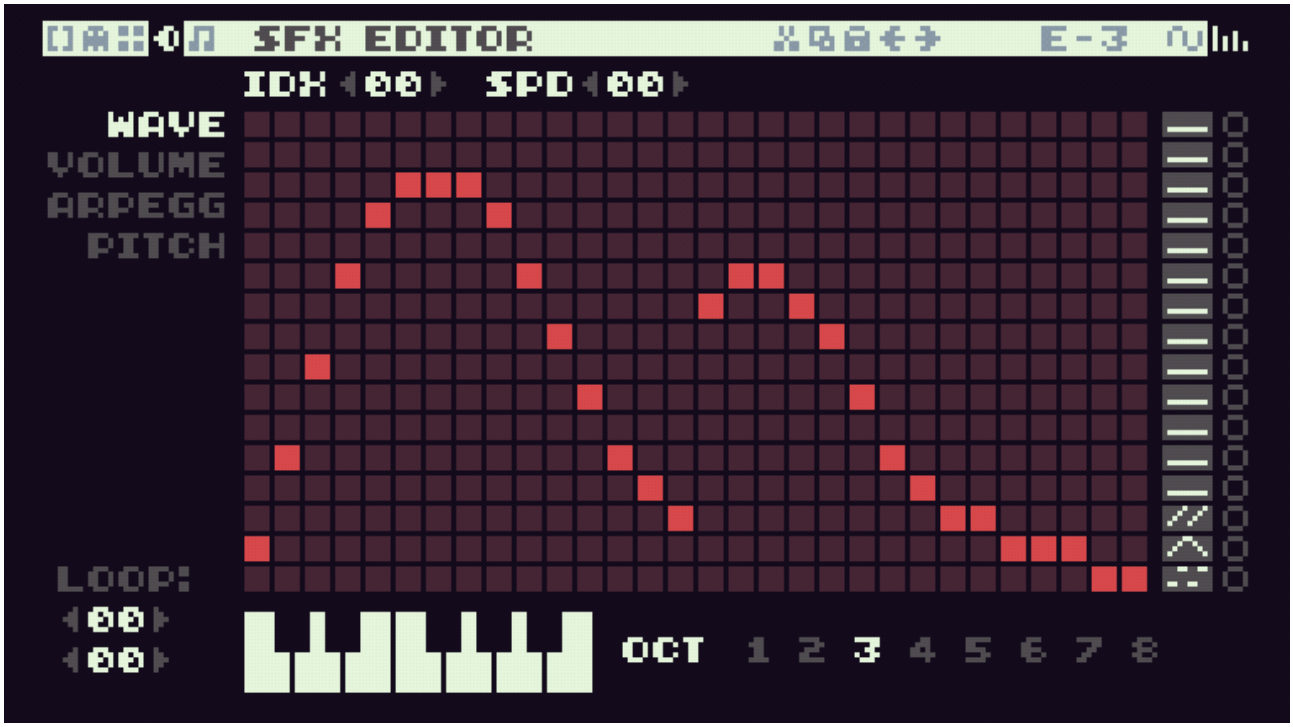
[ESC]でコンソールに戻り、RUNコマンドを実行。

あなたが描いたとおりの背景に変わったでしょうか？

効果音を作成する - SFX エディタ

今度は効果音(SFX)を作ってみましょう。このエディタで設定した音は音楽エディタでも使用されます。

コンソール画面で[ESC]キーを押してエディタ画面に切り替え、[F4]キーを押すか、マウスで左上の SFX エディタアイコンをクリックします。



画面下のほうにピアノの鍵盤があるのでクリックしてみましょう。ドレミの音色が鳴りますよね？鍵盤の横にある「OCT(オクターブ)」の右側に並んだ 1〜8 の数字をクリックするとオクターブを変更できます。

画面上方の「IDX(インデックス)」が効果音 No. となっており、全部で 64 種類(00〜63)まで効果音を作れます。その右隣にある「SPD(スピード)」で再生速度(-4〜03)をコントロールできます。

画面の左側には WAVE(波形)、VOLUME(音量)、ARPEGG(アルペジオ)、PITCH(ピッチ)の 4 つのパラメータがあります。その下方にある「LOOP(ループ)」は効果音を部分的にループさせる機能です。1 番目の数値はループの長さ(0〜15)、2 番目の数値(0〜15)はループの位置を指定します。

画面中央の格子状のエリアをクリックすると線を描くように波形を変えられます。

一つ一つの意味がわからなくてもかまわないので適当にいじって鍵盤で音を鳴らしながら音を作ってみてください。出来たらその効果音をプログラムから鳴らしてみましょう。

コードエディタを開き、16行目に次の1行を追加します。これは「もし[Z]キーが押されたら効果音0番を鳴らせ！」という命令文です。

```
if btnp(4) then sfx(0) end
```

[ESC]でコンソールに戻り、RUNコマンドを実行します。[Z]キーを押すと、あなたが作った効果音が鳴るはずです。

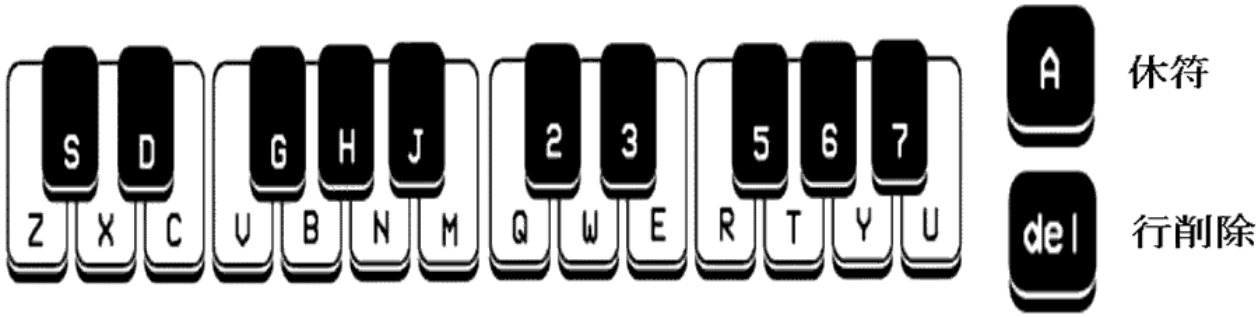
作曲する - 音楽エディタ

音楽エディタは[F5]キーを押すか、音符マークのアイコンをクリックすると切り替わります。作曲そのものは簡単ではないですが、お手軽なDTM環境です。音楽好きの人は楽しみながら作れるんじゃないでしょうか。画面には4つの入力用チャンネルが現れます。それぞれのチャンネルに縦方向に音符を入力していきますが、細かな説明よりもまずは最短で音楽を鳴らしてみることにしましょう。

まずは一番左のチャンネル上にあるパターン番号「00」をすぐ右の[>]をクリックして「01」にします。次にそのチャンネルの音符入力枠の1行目左端をクリックして、その状態からキーボードを[Z][X][C][V][B][N][M]...と打ってみましょう。

音を鳴らしながら音階を入力できたでしょう。

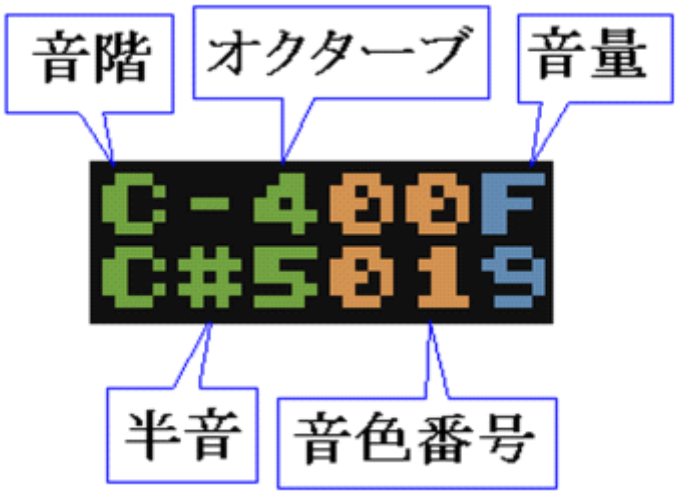
ミュージックエディタでは下図の通りキーボードがピアノの鍵盤に対応されています。



その他にはスペースキーで選択行の音を再生。[Enter]キーで曲全体の再生開始・停止ができます。また、画面右上にあるアイコンボタンでも同様の操作が可能です。

入力された音符コードの説明

左から1番目が音階(CDEFGABがドレミファソラシになっている)。2番目には半音(ピアノの黒い鍵盤の音)をあらわすシャープ「#」が表示されます。3番目は音の高低をあらわすオクターブ。4番目の数字が音色番号(効果音No.と連動)。最後が音量で16進数(0～15を0～9,A～Fであらわす)で表示されています。



カーソルキーで入力枠を移動してキーボードから英数字で直接入力、変更できます。

短めの曲を適当に作ってみましょう。

チャンネル2つ使ってハーモニーしてみました。真っ黒な空白行は休符です。誰もが知ってる名曲のはずですが、再生してみるとその片鱗も感じさせない編曲ぶりです。みなさんはもっとマシな楽曲を作れるよう頑張ってくださいね(ツールの使い方以前に、音楽にはそれなりの才能が必要だとわかります…)

「TRACK(トラック)」の番号を変更すれば、さらに別の曲を入力できます。番号は 00〜07 まで指定でき、1つのカートリッジには8曲までが作成可能です。

プログラムから曲を再生するには「music()」命令を使います。先程「sfx(0)」と書いた部分を入れ替えて試してみましょう。

```
if btnp(4) then music(0,0,0,false) end
```

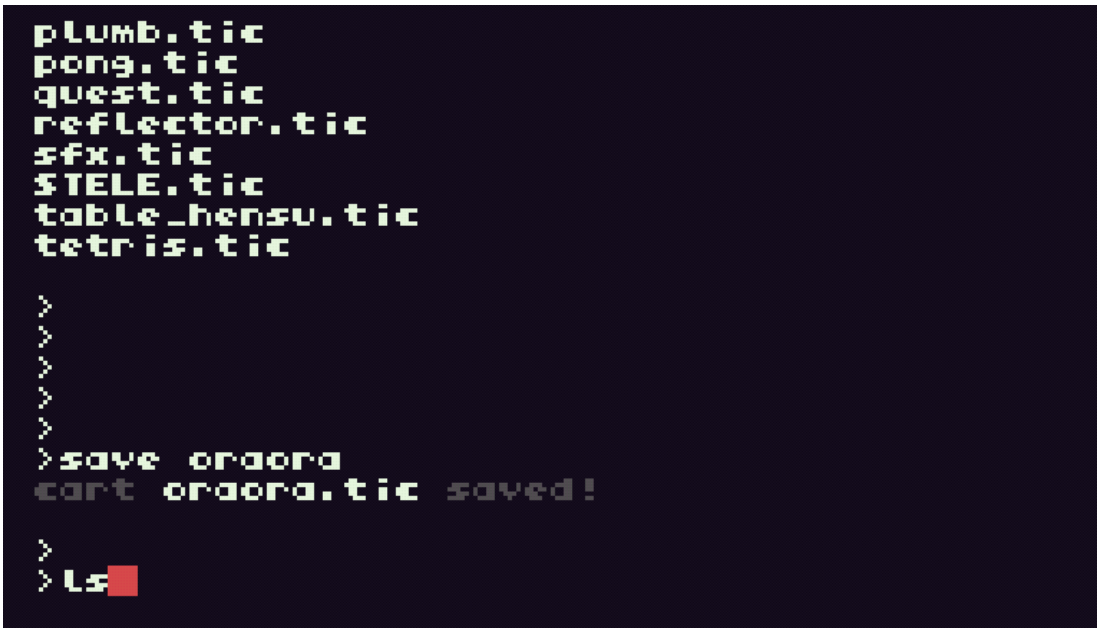
music 命令のカッコ内のパラメータは左から、再生するトラック番号、開始フレーム番号、開始行、リピート(繰り返し)の指定です。最後の「false」を「true」に変えると無限にリピート(繰り返し)再生するようになります。

カートリッジファイルの保存

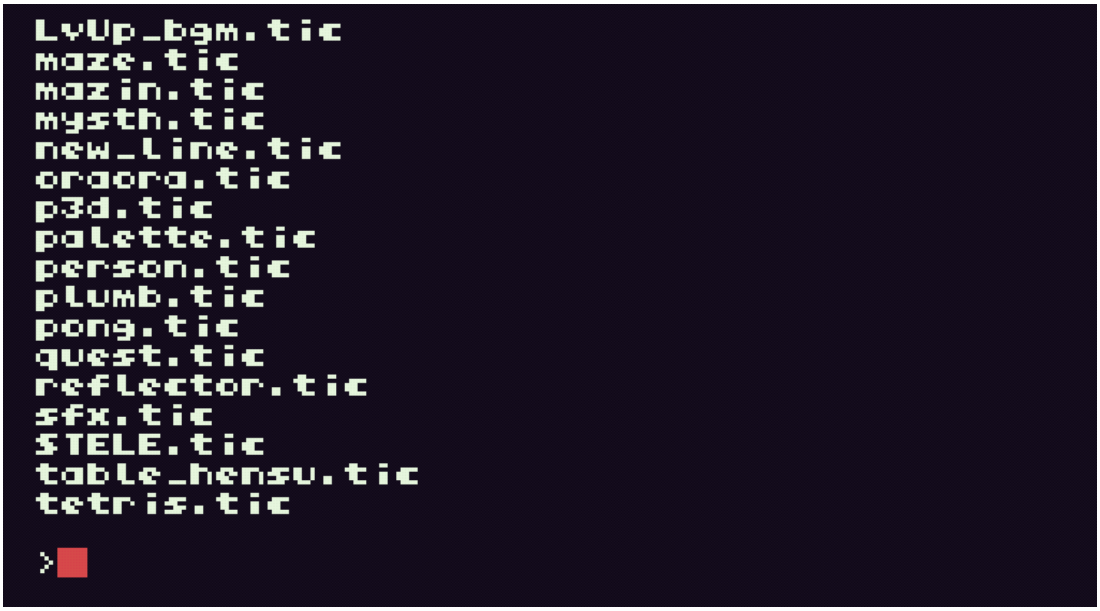
せっかく色々とプログラムを改造したので保存しておきましょう。

コンソール画面に戻ってSAVEコマンドを使います。ファイル名には英数字で任意の名前を付けて保存することができますが、ここでは次のように入力します。

```
save oraora
```



本当に保存できているかどうかはLS(またはDIR)コマンドで確認できます。



拡張子に「.tic」と付いた「oraora.tic」という名前で保存されています。カートリッジファイルにはプログラムのコードだけでなく、ドット絵やマップデータ、効果音、音楽などのすべてのゲームデータが一緒に保存されます。

また、保存したカートリッジファイルを読み込むときは、次のようにLOADコマンドを使います。

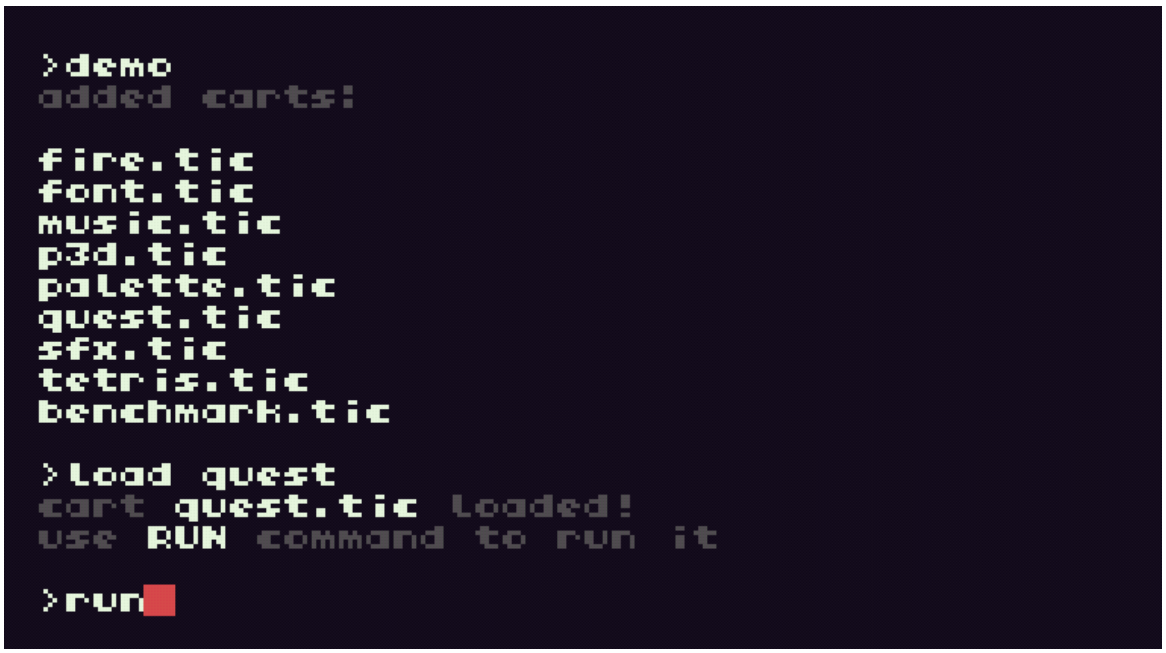
```
load oraora
```

TIC-80 ゲームで遊んでみよう！

さて、このへんでノーミソをちょっとリフレッシュ！お遊びタイムといきましょう。

サンプルデモで遊ぶ

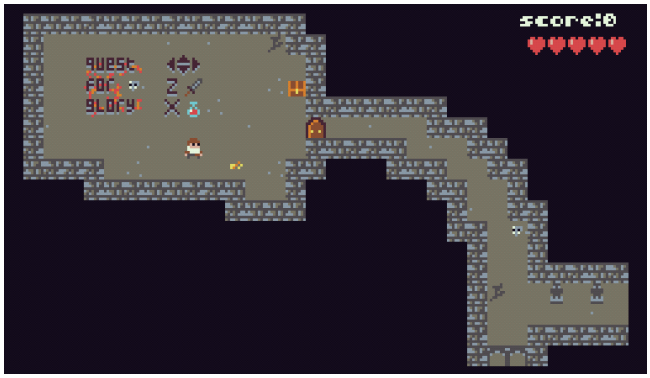
DEMOコマンドを打ち込むと、サンプルプログラムがいくつか自動的に追加されます。



実際にはどれでもかまいませんが、この中の「quest.tic」をロード(読み込み)して実行してみましょう。次のとおりにコマンド入力します。拡張子は付けなくてもOKです。

```
load quest

run
```



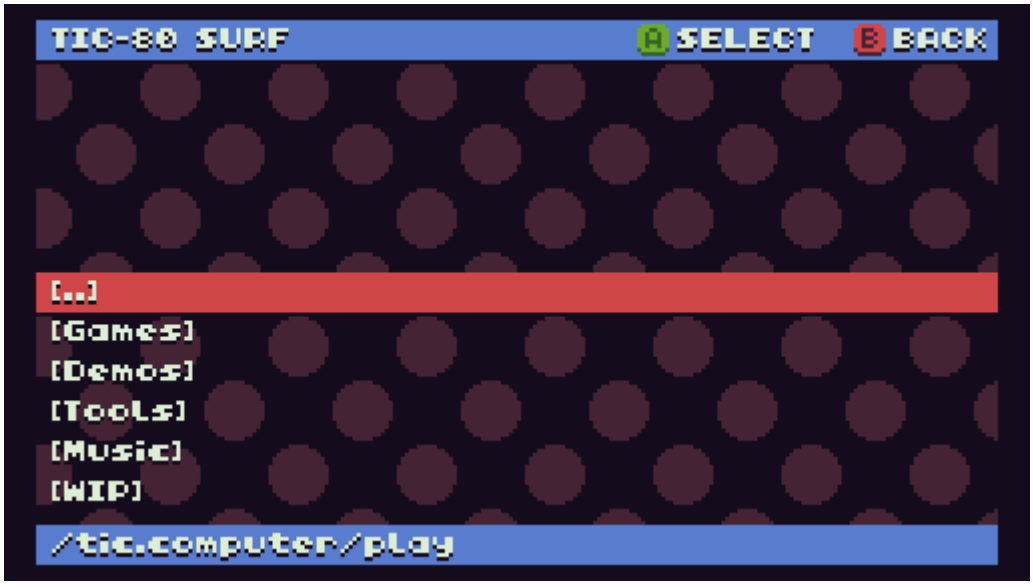
ローグライクRPG「クエスト」が遊べます。[ESC]を押せば終了し、コードエディタでプログラムを見ることもできます。

カートリッジブラウザ

SURF コマンドを打ち込むと、カートリッジブラウザが立ち上がります。

キーボードならカーソルキーと[A][S][Z][X]キーで、ゲームパッドを接続していれば方向キーと[A][B][X][Y]ボタンで操作できます。たくさんのリストが出てくると思いますので色々選んで試してみましょう。

[]カッコで囲まれたメニューはルートメニューになっていて、[A]ボタン(または[Z]キー)でさらに配下のサブメニューへと移動します。例えば1番上の[tic.computer/play]を選ぶと、次のようなカテゴリーのサブメニューが現れます。



ここからはWeb上に登録された多数のユーザ作品を楽しむことができます。

(※インターネットに接続可能な状態である必要があります)

FPS80

[Games]カテゴリーなら

3D-FPS の「FPS80」

ハイセンスなリアルタイム RPG「BALMUNG」

キュートな魔女っ子 STG「WITCHEM UP」

Bad Apple

[Music]カテゴリーなら

超有名な鉄板 MV「Bad Apple」が必見です！

[WIP]カテゴリーは Work in progress（作業中)の略語で「作りかけだけど見てよ～」というユーザの作品がカオス＆フレッシュな感じで公開されているカテゴリーです。

ここでは「FALLSPIRE」を一度は見ておきましょう。

さらに「MICRO-PLATFORMER STARTER KIT」はマリオタイプのジャンプアクションを作りたい人には便利なテンプレートになっている習作です。他にも[Demos]や[Tools]などのカテゴリーがあるので、様々な作品を試遊してみましょう。

ちなみにメニュー選択では[B]ボタン(または[X]キー)を押すか、メニューの[..]を選択すると前画面に戻れます。SURF をやめるときは[ESC]キーでコンソール画面に戻れます。

さらに遊んでいるゲームを中断してエディタに移れば、作品のコードやグラフィック、サウンドなどの素材も全部自由に見れちゃうのが TIC-80 の良いところでもあります。

【巻末附録】

TIC-80 仕様一覧

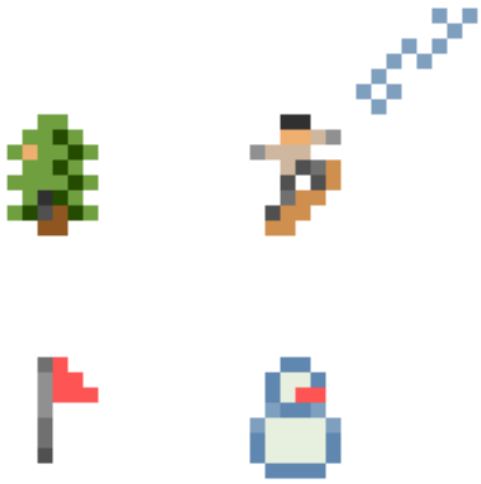
グラフィック表示画面	240×136 ピクセル、16 色パレット
入力デバイス	8 ボタンゲームパッド(4 つまで)／マウス／キーボード
スプライト	8×8 ピクセルを 1 つとして 512 個まで。
マップデータ	240×136 セル(1 セルが 8×8 ピクセルのスプライト)
サウンド	4 チャンネル(編集可能な波形エンベロープ)
プログラムコード	64KB(65536 文字まで)

ホットキー

ESC	コンソール／エディタ間の切り替え
F1/ALT+1	コードエディタを開く
F2/ALT+2	スプライトエディタを開く
F3/ALT+3	マップエディタを開く
F4/ALT+4	SFX エディタを開く
F5/ALT+5	ミュージックエディタを開く
CTRL+PGUP/PGDOWN	各エディタへの前後移動
F7	画面ショットをカートリッジイメージとして保存する
F8	GIF 画像ファイルとして画面ショットを保存する
F9	GIF 動画録画の開始／停止および保存
F11/ALT+ENTER	フルスクリーン／ウィンドウ画面切り替え
CTRL+R	現在のプログラムを実行する
CTRL+X/C/V	エディタ上のカット／コピー／貼り付け
CTRL+S	現在のカートリッジを保存
SHIFT	マップエディタ上でスプライトの BG エリアを表示する
CTRL+Z/Y	元に戻す(UNDO)／やり直し(REDO)
CTRL+F	コードエディタ上で文字検索する
CTRL+G	コードエディタ上で指定行へ移動する
CTRL+O	コードエディタ上で関数行へ移動する
SHIFT+ENTER	ミュージックエディタ上でカーソル位置から曲を再生

コンソールコマンド

help - 使用可能コマンドを一覧表示する。
ram - 80K RAM レイアウト(メモリマップ)を表示する。
exit - TIC-80 を終了する。
edit - 画面をコードエディタに切り替える。
new [lua | moon | js] - 新しい「Hello World」カートリッジを作成する
load <cart> [sprites | map | cover | code | sfx | music | palette]
 - カートリッジファイルを読み込みます。拡張子は付けても付けなくても大丈夫。また、データの一部(スプライト、マップ等)のみを読み込むことも可能です。
save <cart> - 作成中のゲームをカートリッジファイルとして保存する。
run - 現在のプログラムを実行する。
resume - 最後に実行されたプログラムを再開する。
dir - 現在のディレクトリにあるファイル一覧を表示する(ls でも可)。
cd - ディレクトリを移動する。
mkdir - ディレクトリを作成する。
folder - カートリッジ保存先のディレクトリをエクスプローラで開く。
add - TIC-80 にカートリッジファイルを追加する。
del <file> - 指定したファイルを削除する。
get <file> - 指定したファイルをローカル PC に保存する。
export [html | native | sprites | cover | map]
 - 読み込んであるカートリッジから実行可能な HTML 形式(html)や対象 OS の実行形式(native)で保存、スプライト(sprites)や画面ショット(cover)を GIF 画像として保存、マップをバイナリデータとして保存する。
import [sprites | cover | map] - export コマンドで出力したデータをインポートする。
cls - 画面をクリアする。
demo - デモ用カートリッジをインストールする。
version - TIC-80 のバージョンを表示する。
config [save | default] - 現在のカートリッジ設定をコードエディタから編集する。
surf - カートリッジブラウザを開く。



TIC-80 命令表

TIC-80 では、TIC-80 独自の API とプログラム言語 Lua の文法を組み合わせてプログラミングできます。以下は TIC-80 の API 一覧です。

TIC()

毎秒 60 回自動で呼び出されるコールバック関数。TIC-80 のゲームループ処理の中心となる必須の関数です。

```
function TIC()
  -- ここに任意の処理を記述
end
```

SCN()

グラフィック描画 1 ライン毎に呼び出される関数。パラメータには描画時のライン位置が入ります。

```
function SCN(line)
  -- ここに任意の処理を記述
end
```

OVR()

フレーム単位で呼び出される関数。SCN 関数で変更したパレットの影響を受けません。

```
function OVR()
  -- ここに任意の処理を記述
end
```

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

画面の描画領域を制限する。パラメータなしで実行すると描画領域がリセットされる。

cls([color])

画面全体をクリアする。

circ(x, y, radius, color)

塗りつぶした円を描く。

circb(x, y, radius, color)

塗りつぶさない円を描く。

btn([id: 0..5 8..13]) -> state

現在のゲームパッドボタンの状態を取得。押されていれば true、押されていなければ false が取得される。

btnp([id: 0..5 8..13, [hold period]]) -> state

現在のゲームパッドボタンの状態を取得。ただし、押された瞬間だけを検知する。

Keyboad	Gamepad	Player-1 ID	Player-2 ID	Player-3 ID	Player-4 ID
↑	↑	0	8	16	24
↓	↓	1	9	17	25
←	←	2	10	18	26
→	→	3	11	19	27
Z	A	4	12	20	28
X	B	5	13	21	29
A	X	6	14	22	30
S	Y	7	15	23	31

exit()

プログラムを中断してコンソールに戻る。

font(text, x, y, colorkey, char_width, char_height, fixed, scale) -> width

スプライトの FG エリアに定義されたフォントで文字列を表示する。

line(x0, y0, x1, y1, color)

指定色で座標 2 点間に直線を描く。

map([x=0, y=0], [w=30, h=17], [sx=0, sy=0], [colorkey=-1], [scale=1], [remap=nil])

画面にマップを描画する。

memcpy(toaddr, fromaddr, len)

RAM 上のデータをコピー元からコピー先のアドレスへ指定バイト数分だけコピーする。

memset(addr, val, len)

指定バイト数分だけ指定アドレスにバイト値を書き込む。

mget(x, y) -> id

マップ座標からそこにセットされているスプライト番号を取得する。

mouse() -> x, y, state

現在のマウスの X,Y 座標およびボタンの状態を取得。押されていれば true、押されていなければ false が取得される。

mset(x, y, id)

マップデータの指定座標にスプライト番号をセットする。

key(keycode)

現在のキーボードボタンの状態を取得する。押されていれば true、押されていなければ false が取得される。

keyp(keycode)

現在のキーボードボタンの状態を取得。ただし、押された瞬間だけを検知する。

KEYCODES		
01 = A	27 = 0	50 = RETURN
02 = B	28 = 1	51 = BACKSPACE
03 = C	29 = 2	52 = DELETE
04 = D	30 = 3	53 = INSERT
05 = E	31 = 4	
06 = F	32 = 5	54 = PAGEUP
07 = G	33 = 6	55 = PAGEDOWN
08 = H	34 = 7	56 = HOME
09 = I	35 = 8	57 = END
10 = J	36 = 9	58 = UP
11 = K		59 = DOWN
12 = L	37 = MINUS	60 = LEFT
13 = M	38 = EQUALS	61 = RIGHT
14 = N	39 = LEFTBRACKET	
15 = O	40 = RIGHTBRACKET	62 = CAPSLOCK
16 = P	41 = BACKSLASH	63 = CTRL
17 = Q	42 = SEMICOLON	64 = SHIFT
18 = R	43 = APOSTROPHE	65 = ALT
19 = S	44 = GRAVE	
20 = T	45 = COMMA	
21 = U	46 = PERIOD	
22 = V	47 = SLASH	
23 = W		
24 = X	48 = SPACE	
25 = Y	49 = TAB	
26 = Z		

music([track=-1], [frame=-1], [row=-1], [loop=true])

指定トラック番号の音楽を再生する。

peek(addr) -> val

RAM から 1 バイト読み込む。

peek4(addr4) -> val4

RAM から半バイト(4 ビット分)の値を読み込む。指定するアドレスは peek 命令の倍の値となる点に注意。

poke(addr, val)

RAM に 1 バイト書き込む。

poke4(addr4, val)

RAMに半バイト(4ビット分)の値を書き込む。指定するアドレスはpoke命令の倍の値となる点に注意。

pix(x, y, [color]) -> color

指定色で画面上に点(ドット)を描く。または画面上の1点から色を取得する。

pmem(index:0..6, [val]) -> val

値をスロットメモリに保存、または読み出します。プログラム終了後もハイスコアデータなどを残すことができる永続メモリ領域を読み書きする。

print(text, [x=0, y=0], [color=15], [fixed=false], [scale=1]) -> width

文字列を表示する。

rect(x, y, w, h, color)

塗りつぶした矩形(四角形)を描画する。

rectb(x, y, w, h, color)

塗りつぶさない矩形(四角形)を描画する。

reset()

カートリッジの実行をリセットして初期状態に戻す。

sfx(id, [note], [duration=-1], [channel=0], [volume=15], [speed=0])

指定番号の効果音を再生する。

spr(id, x, y, [colorkey=-1], [scale=1], [flip=0], [rotate=0], [w=1, h=1])

指定番号のスプライトを表示。拡大、回転、反転などの指定も可能です。

sync([toCart=true])

実行中に変更されたスプライトやマップデータをカートリッジに同期(コピー)する。

time() -> ticks

ゲーム実行開始からの経過時間をミリ秒(1/1000秒)単位で取得する。

trace(msg, [color])

プログラム実行中の文字表示をコンソールにも出力する。

tri(x1, y1, x2, y2, x3, y3, color)

塗りつぶされた三角形を描画する。

textri(x1, y1, x2, y2, x3, y3, u1, v1, u2, v2, u3, v3, [use_map=false], [chroma=-1])

マップに描かれたテクスチャで塗りつぶされた三角形を描画する。

Lua 基本文法

マルチステートメント

複数の命令文をセミコロン「;」で区切ることで１行内に記述可能です。

```
a=8;b=5;c=a+b;print("a+b="..c,0,0)
```

コメント

ハイフン２つ「--」の後続をコメントとします。複数行の場合は --[[と]] で囲まれた範囲をコメントとします。

```
-- comment
```

```
--[[
In case of multiple lines,
The part enclosed in square brackets becomes a comment
]]
```

制御文

```
if 条件式 then
          処理
elseif 条件式 then
          処理
else
          処理
end
```

```
while 条件式 do
          処理
end
```

```
repeat
          処理
until 条件式
```

```
for 変数=開始値,終了値 do
          処理
end
```

```
break
```

while、repeat、for 命令文のループ処理から脱出します。

```
goto ラベル名
```

ラベルへジャンプします。

```
::ラベル名::
```

ラベル名はコロン２つ「::」で囲んで記述します。

Lua 標準ライブラリ一覧

最後に Lua の標準ライブラリの主な機能を書式付で列挙します。

【基本機能】

_G	グローバル環境を保持するグローバル変数(関数ではない)
_VERSION	バージョン文字列を保持するグローバル変数(関数ではない)
assert(v, [message])	引数 v が偽であれば error を呼ぶ
collectgarbage([opt, [arg]])	ガベージコレクタの様々な機能を実行する
dofile([filename])	指定された Lua ファイルを開き、実行する
error(message, [level])	エラーとして終了する(この関数は戻りません)
getmetatable(object)	指定されたオブジェクトのメタテーブルを返す
ipairs(t)	イテレータ関数、table、0 の三つ組みを返す
load(chunk, [chunkname, [mode, [env]]])	チャンクをロードする
loadfile([filename, [mode, [env]]])	チャンクをファイル filename から取得
next(table, [index])	テーブルのすべてのフィールドを巡回する
pairs(t)	テーブル t のすべてのキーと値のペアを巡回する
pcall(f, [arg1, ...])	指定された引数を渡して関数 f を保護モードで呼ぶ
rawequal(v1, v2)	v1 と v2 が等しいかどうか判定しブーリアンで返す
rawget(table, index)	table[index]の値を取得
rawlen(v)	オブジェクト v の長さを返す
rawset(table, index, value)	table[index]の値を value に設定
select(index, ...)	index 番目以降の引数をすべて返す
setmetatable(table, metatable)	指定されたテーブルのメタテーブルを設定
tonumber(e, [base])	引数を数値に変換して返す
tostring(v)	引数を文字列に変換して返す
type(v)	引数の型を文字列として返す
xpcall(f, msgh, [arg1, ...])	新しいメッセージハンドラに msgh を設定

【コルーチン操作】

coroutine.create(f)	関数 f を本体に持つ新しいコルーチンを作成
coroutine.isyieldable()	実行中のコルーチンが yield 可能であれば真を返す
coroutine.resume(co, [val1, ...])	コルーチン co の実行を開始または続行する
coroutine.running()	実行中のコルーチンとブーリアンを返す
coroutine.status(co)	コルーチン co の状態を文字列で返す
coroutine.wrap(f)	関数 f を本体に持つ新しいコルーチンを作成する
coroutine.yield(...)	呼び出し元のコルーチンの実行を中断する

【数学関数】

math.abs(x)	x の絶対値を返す
math.acos(x)	x の逆余弦を(ラジアンで)返す
math.asin(x)	x の逆正弦を(ラジアンで)返す
math.atan(y, [x])	y/x の逆正接を(ラジアンで)返す
math.ceil(x)	x より大きいまたは等しい最小の整数値を返す
math.cos(x)	x(ラジアン)の余弦を返す
math.deg(x)	角度 x をラジアンから度に変換する
math.exp(x)	e^x の値を返す(e は自然対数の底)
math.floor(x)	x より小さいまたは等しい最大の整数を返す
math.fmod(x)	x を y で割った商を丸めた余りを返す
math.huge	浮動小数点数の最大値 HUGE_VAL を返す

math.log(x, [base])	指定された base を底とする x の対数を返す
math.max(x, ...)	Lua の演算子"<"に従って最大の値を持つ引数を返す
math.maxinteger	整数の最大値を返す
math.min(x, ...)	Lua の演算子">"に従って最小の値を持つ引数を返す
math.mininteger	整数の最小値を返す
math.modf(x)	x の整数部と x の小数部を返す
math.pi	円周率 π の値
math.rad(x)	角度 x を度からラジアンに変換する
math.random([m, [n]])	浮動小数点数の擬似乱数を返す
math.randomseed(x)	x を擬似乱数生成器の種として設定する
math.sin(x)	x(ラジアン)の正弦を返す
math.sqrt(x)	x の平方根を返す
math.tan(x)	x(ラジアン)の正接を返す
math.tointeger(x)	x を整数に変換して返す
math.type(x)	x の数値型を文字列として返す
math.ult(m, n)	符号無しで比較し m が n より下であれば真を返す

【文字列操作】

string.byte(s, [i, [j]])	文字 s の文字コード値を返す
string.char(...)	文字コード値から文字列を生成する
string.dump(function, [strip])	指定された関数のバイナリ文字列を返す
string.find(s, pattern, [init, [plain]])	文字列 s からマッチする pattern を検索する
string.format(formatstring, ...)	指定された書式に従った文字列を返す
string.gmatch(s, pattern)	マッチした pattern 順にイテレータ関数を返す
string.gsub(s, pattern, repl, [n])	マッチした pattern を指定文字列で置換する
string.len(s)	文字列の長さを返す
string.lower(s)	文字列中の大文字をすべて小文字に変換する
string.match(s, pattern, [init])	文字列 s からマッチする pattern を検索する
string.pack(fmt, v1, v2, ...)	書式 fmt に従いバイナリ文字列にして返す
string.packsize(fmt)	string.pack で返される文字列のサイズを返す
string.rep(s, n, [sep])	文字列 s を sep で区切って n 個連結した文字列を返す
string.reverse(s)	反転文字列を返す
string.sub(s, i, [j])	文字列 s の i 文字目から j 文字目までの部分を返す
string.unpack(fmt, s, [pos])	書式 fmt に従って文字列 s にパックされた値を返す
string.upper(s)	文字列中の小文字をすべて大文字に変換する

【テーブル操作】

table.concat(list, [sep, [i, [j]]])	テーブルの各要素を sep で結合して返す
table.insert(list, [pos], value)	list の位置 pos に要素 value を挿入する
table.move(a1, f, e, t, [a2])	テーブル a1 の要素をテーブル a2 に移動する
table.pack(...)	指定した引数をすべて格納した新しいテーブルを返す
table.remove(list, [pos])	list から位置 pos の要素を切り出して返す
table.sort(list, [comp])	list の要素をソートする
table.unpack(list, [i, [j]])	指定されたリストの要素を返す

各命令の詳細説明は以下 Lua 関連サイトをご参照ください。

Lua 公式サイト

https://www.lua.org

Lua リファレンスマニュアル(日本語訳)

http://milkpot.sakura.ne.jp/lua/

MIT ライセンスについて

TIC-80 の著作権／ライセンス表示は下記の通り。簡単に言うと「著作権と MIT ライセンスである旨を表示すれば、誰でも無償で無制限にあつかって良いけど、作者(著作権者)はソフトウェアに関して何があっても責任を負いませんよ」というライセンスです。