
PICO-8
v0.2.0h

PICO-8 v0.2.0h

https://www.pico-8.com

Author: Joseph White >> hey@lexaloffle.com

PICO-8 is built with:

SDL2 >> http://www.libsdl.org

Lua 5.2 >> http://www.lua.org

GIFLIB >> http://giflib.sourceforge.net/

WiringPi >> http://wiringpi.com/

libb64 by Chris Venter

miniz by Rich Geldreich

ws281x by jgarff

Welcome to PICO-8!

PICO-8 is a fantasy console for making, sharing and playing tiny games and other computer programs. When you turn it on, the machine greets you with a shell for typing in Lua programs and provides simple built-in tools for creating sprites, maps and sound.

The harsh limitations of PICO-8 are carefully chosen to be fun to work with, encourage small but expressive designs and hopefully to give PICO-8 cartridges their own particular look and feel.

KEYS

Toggle Fullscreen: Alt+Enter

Quit: Alt+F4 or CMD+Q

Reload/Run/Restart cart: Ctrl+R

Quick-Save: Ctrl+S

Mute/Unmute: Ctrl+M

Default Controls

Player 1 defaults: Cursors + ZX / NM / CV

Player 2 defaults: SDFE + tab,Q / shift A Enter or P for pause menu (while running)

Use KEYCONFIG to change the defaults.

Specs

Display: 128x128, fixed 16 colour palette
Input: 6-button controllers
Cartridge size: 32k
Sound: 4 channel, 64 definable chip blerps
Code: Lua (max 8192 tokens of code)
Sprites: Single bank of 128 8x8 sprites (+128 shared)

Map: 128x32 8-bit cells (+128x32 shared)

Hello World

After PICO-8 boots, try typing some of these commands followed by enter:

```
print("hello world")
rectfill(80,80,120,100,12)
circfill(70,90,20,14)
for i=1,4 do print(i) end
```

(Note: PICO-8 only displays upper-case characters -- just type normally without caps-lock [shifted characters produce glyph characters])

You can build up an interactive program by using commands like this in the code editing mode along with two special callback functions _UPDATE and _DRAW. For example, the following program allows you to move a circle around with the cursor keys. Press Esc to switch to the code editor and type or copy & paste the following code:

```
x=64 y=64

function _update()
```

```
  IF (BTN(0)) THEN X=X-1 END
  IF (BTN(1)) THEN X=X+1 END
  IF (BTN(2)) THEN Y=Y-1 END
  IF (BTN(3)) THEN Y=Y+1 END
  END
  FUNCTION _DRAW() CLS(5)
  CIRCFILL(X,Y,7,14)
  END
```

Now press ESC to return to the console and type RUN (or press CTRL-R) to see it in action.

Please refer to the demo cartridges for more complex programs (type INSTALL_DEMOS).

If you want to store your program for later, use the SAVE command:

```
SAVE <NAME>
```

e.g.

```
SAVE PINKCIRC
```

And to load it again:

```
LOAD <NAME>
```

e.g.

```
LOAD PINKCIRC
```

Example Cartridges

These cartridges are included with PICO-8 and can be installed by typing:

```
install_demos
cd demos
ls
```

hello	**Greetings from PICO-8**
api	**Demonstrates most PICO-8 functions**
jelpi	**Platform game demo w/ 2p support**
cast	**2.5D Raycaster demo**
drippy	**Draw a drippy squiggle**
wander	**Simple walking simulator**
collide	**Example wall and actor collisions**

To run a cartridge, open PICO-8 and type:

```
load jelpi
run
```

Press ESC to stop the program, and once more to enter editing mode.

A small collection of BBS carts can also be installed with:

```
install_games
```

File System

These commands can be used to manage files and directories (folders):

LS	**list the current directory**
CD <NAME>	**change directory**
CD ..	**go up a directory**
CD /	**go up to PICO's home directory**
MKDIR <NAME>	**make a directory**
FOLDER	**open the current directory in the host OS's file browser**
LOAD <NAME>	**load a cart from the current directory**
SAVE <NAME>	**save a cart to the current directory**

If you want to move files around, duplicate them or delete them, use the FOLDER command and do it in the host operating system. The default location for PICO-8's drive is:

Windows: C:/Users/Yourname/AppData/Roaming/pico-8/carts

macOS: ~/Library/Application Support/pico-8/carts

Linux: ~/.lexaloffle/pico-8/carts

You can change this and other settings in pico-8/config.txt

> **Tip: The drive directory can be mapped to a cloud drive (provided by Dropbox or similar) in order to create a single disk shared between PICO-8 machines spread across different host machines.**

Loading and Saving

When using LOAD and SAVE, the .P8 extension can be omitted and is added automatically.

Saving to a .p8.png extention will save the cartridge in a special image format that looks like a cartridge.

Use a filename of "@CLIP" to load or save to the clipboard.

Once a cartridge has been loaded or saved, it can also be quick-saved with CTRL-S.

Saving .p8.png carts with a text label and preview image

To generate a label image saved with the cart, run the program first and press CTRL-7 to grab whatever is on the screen. The first two lines of the program starting with '--' are also drawn to the cart's label.

e.g.

```
-- ocean diver legends -- by loopy
```

Code size restrictions for .png format

When saving in .png format, the compressed size of the code must be less than 15360 bytes. To find out the current size of your code, use the INFO command. The compressed size limit is not enforced for saving in .p8 format.

Using an External Text Editor

It is possible to edit .p8 files directly with a separate text editor. Using CTRL-R to run a cartridge will automatically reload the file if:

1. **There are no unsaved changes in the PICO-8 editors, AND**
2. **The file differs in content from the last loaded version**

If there are changes to both the cart on disk and in the editor, a notification is displayed:

didn't reload: unsaved changes

PICO-8 does not fully support upper-case characters, and they are automatically converted to lower-case when viewed in the code editor. Note that this also causes unsaved changes to exist, which means that CTRL-R will stop automatically reloading the version on disk until it is manually LOADed.

Glyph characters (typed with shift-A..Z) are stored in .p8 format and in the host operating system's clipboard as rough Unicode equivalents.

Backups

If you quit without saving changes, or overwrite an existing file, a backup of the cartridge is saved to {appdata}/pico-8/backup. An extra copy of the current cartridge can also be saved to the same folder by typing BACKUP.

Configuration

Config.txt

You can find some settings in config.txt. Edit the file when PICO-8 is not running.

Windows: C:/Users/Yourname/AppData/Roaming/pico-8/config.txt

OSX: /Users/Yourname/Library/Application Support/pico- 8/config.txt

Linux: ~/.lexaloffle/pico-8/config.txt

Use the -home switch (below) to use a different path config.txt and other data. Some settings can be changed while running PICO-8 by

CONFIG SETTING VALUE.

(type CONFIG by itself for a list)

Commandline parameters

note: these override settings found in config.txt

pico-8 [switches] [filename.p8]

-width n	set the window width
-height n	set the window height
-windowed n	set windowed mode off (0) or on (1)

-volume n	**set audio volume (0..256)**
-joystick n	**joystick controls starts at player n (0..7)**
-pixel_perfect n	**1 for unfiltered screen stretching at integer scales (on by default)**
-preblit_scale n	**scale the display by n before blitting to screen (useful with -pixel_perfect 0)**
-draw_rect x,y,w,h	**absolute window coordinates and size to draw pico-8's screen**
-run filename	**load and run a cartridge**
-x filename	**execute a PICO-8 cart headless and then quit (experimental!)**
-export param_str	**run EXPORT command in headless mode and exit (see notes under export)**
-p param_str	**pass a parameter string to the specified cartridge**
-splore	**boot in splore mode**
-home path	**set the path to store config.txt and other user data files**
-root_path path	**set the path to store cartridge files**
-desktop path	**set a location for**

	screenshots and gifs to be saved
-screenshot_scale n	scale of screenshots. default: 3 (368x368 pixels)
-gif_scale n	scale of gif captures. default: 2 (256x256 pixels)
-gif_len n	set the maximum gif length in seconds (1..120)
-gui_theme n	use 1 for a higher contrast editor colour scheme
-timeout n	how many seconds to wait before downloads timeout (default: 30)
-software_blit n	use software blitting mode off (0) or on (1)
-foreground_sleep_ms n	how many milliseconds to sleep between frames.
-background_sleep_ms n	how many milliseconds to sleep between frames when running in background
-accept_future n	use 1 to allow loading cartridges made with future versions of PICO-8

Controller Setup

PICO-8 uses the SDL2 controller configuration scheme. It will detect common controllers on startup and also looks for custom mappings in sdl_controllers.txt in the same directory as config.txt. sdl_controllers.txt has one mapping per line.

To generate a custom mapping string for your controller, use

either the controllermap program that comes with SDL2, or try http://www.generalarcade.com/gamepadtool/

To find out the id of your controller as it is detected by SDL2, search for "joysticks" or "Mapping" in log.txt after running PICO-8. This id may vary under different operating systems. See: https://www.lexaloffle.com/bbs/?tid=32130

To set up which keyboard keys trigger joystick buttons presses, use KEYCONFIG.

Screenshots, Videos and Cartridge Labels

While a cartridge is running use:

CTRL-6	Save a screenshot to desktop
CTRL-7	Capture cartridge label image
CTRL-8	Start recording a video
CTRL-9	Save GIF video to desktop (8 seconds by default)

You can save a video at any time (it is always recording); CTRL-8 simply resets the video starting point. To record more than 8 seconds, use the CONFIG command (maximum: 120):

```
CONFIG GIF_LEN 60
```

If you would like the recording to reset every time (to create a non- overlapping sequence), use:

```
CONFIG GIF_RESET_MODE 1
```

The gif format can not match 30fps exactly, so PICO-8 instead

uses the closest match: 33.3fps.

If you have trouble saving to the desktop, try configuring an alternative desktop path in config.txt

Sharing Cartridges

There are three ways to share carts made in PICO-8:

1. Share the .p8 or .p8.png file directly with other PICO-8 users. Type FOLDER to open the current folder in your host operating system.
2. Post the cart on the Lexaloffe BBS to get a web-playable version: http://www.lexaloffle.com/pico-8.php?page=submit
3. Export the cartridge to a stand-alone html/js or native binary player: (see the exporters section for details)

Exporters & Importers

The EXPORT command can be used to generate .png, .wav files and stand-alone .html and native binary cartridge players. The output format is inferred from the filename extension (e.g. .png).

You are free to distribute and use exported cartridges and data as you please, provided that you have permission from the author and contributors.

Sprite Sheet (.png)

Expects 128x128 png and colour-fits to the pico-8 palette

```
IMPORT BLAH.PNG
```

Use FOLDER() to locate the exported .png

```
EXPORT BLAH.PNG
```

SFX and Music (.wav)

Export music from the current pattern (when editor mode is MUSIC):

```
EXPORT <NAME>.WAV
```

Export the current SFX (when editor mode is SFX):

```
EXPORT <NAME>.WAV
```

Exports all of the SFXs as blah0.wav, blah1.wav .. blah63.wav

```
EXPORT <NAME>%D.WAV
```

HTML Player (.html)

To generate a stand-alone HTML player (foo.html, foo.js):

```
EXPORT FOO.HTML
```

Or just the .js file:

```
EXPORT FOO.JS
```

Use -F to write the files to a folder called foo_html, using index.html instead of foo.html

```
EXPORT -F FOO.HTML
```

Optionally provide a custom HTML template with the -P switch:

```
EXPORT FOO.HTML -P ONE_BUTTON
```

This will use the file {application data}/pico-8/plates/one_button.html as the

HTML shell, replacing a special string, ##js_file##, with the .js filename.

Use -W to export as .wasm + .js: (still experimental!)

```
EXPORT -W FOO.HTML
```

Binary Player (.bin)

To generate stand-alone executables for Windows, Linux (64-bit), Mac and Raspberry Pi:

```
EXPORT FOO.BIN
```

By default, the cartridge label is used as an icon with no transparency. To specify an icon from the sprite sheet, use -i and optionally -s and/or -c to control the size and transparency.

-I N	Icon index N with a default transparent colour of 0 (black).
-S N	Size NxN sprites. Size 3 would be produce a 24x24 icon.
-C N	Treat colour N as transparent. Use 16 for no transparency.

For example, to use a 2x2 sprite starting at index 32 in the spritesheet, using colour 12 as transparent:

```
EXPORT -I 32 -S 2 -C 12 FOO.BIN
```

Technical note: Windows file systems do not support the file metadata needed to create a Linux or Mac executable. PICO-8 works around this by exporting zip files in a way that preserves the file attributes, as long as the zip files are not re-packaged. Otherwise, a Linux user who then downloads the binaries may need to "chmod a+x foo" the file to run it, and Mac user would need to "chmod a+x foo.app/Contents/MacOS/foo". This issue can be avoided by distributing exported .zip files as-is after using the EXPORT command on Windows.

Uploading to itch.io

If you would like to upload your exported cartridge to itch.io as playable html:

1. From inside PICO-8: EXPORT -F FOO.HTML
2. Create a new project from your itch dashboard.
3. Zip up the folder and upload it (set "This file will be played in the browser")
4. Embed in page, with a size of 750px x 680px.
5. Set "Mobile Friendly" on (default orientation) and "Automatically start on page load" on. (No need for the fullscreen button; the default PICO-8 template has its own.)
6. Set the background (BG2) to something dark (e.g. #232323) and the text to something light (#cccccc)

Exporting Multiple Cartridges

Up to 16 cartridges can be bundled together by passing them to EXPORT, when generating stand-alone html or native binary players.

EXPORT FOO.HTML DAT1.P8 GAME2.P8

During runtime, the extra carts can be accessed as if they were local files:
Load spritesheet from DAT1.P8:

```
RELOAD(0,0,0x2000, "DAT1.P8")
```

Load and run another cart:

```
LOAD("GAME2.P8")
```

Running EXPORT from the host operating system

Use the -export switch when launching PICO-8 to run the exporter in headless mode. Output is to the current directory rather than the PICO-8 file system. Parameters to the EXPORT command are passed along as a single (lowercase) string:

```
================================================================

pico8 foo.p8 -export "-i 32 -s 2 -c 12 foo.bin dat0.p8 dat1.p8"

================================================================
```

Limitations of Exported Cartridges

Exported cartridges are unable to load and run BBS cartridges e.g. via LOAD("#FOO")

SPLORE

SPLORE is a built-in utility for browsing and organising

both local and bbs (online)
cartridges. Type SPLORE to launch it, or launch PICO-8 with -splore.

It is possible to control SPLORE entirely with a joystick:
LEFT and RIGHT to navigate lists of cartridges
UP AND DOWN to select items in each list
X,O or MENU to launch the cartridge

While inside a cart, press MENU to favourite a cartridge or exit to splore. If you're using a keyboard, it's also possible to press F to favourite an item while it is selected in the cartridge list view.

When viewing a list of BBS carts, use the top list item to re-download a list of cartridges. If you are offline, the last downloaded list is displayed, and it is still possible to play any cartridges you have downloaded.

If you have installed PICO-8 on a machine with no internet access, you can also use INSTALL_GAMES to add a small selection of pre-installed BBS carts to /games

QUIRKS of PICO-8

Common gotchas to watch out for:

- The bottom half of the spritesheet and bottom half of the map occupy the same memory. (Best use only one or the other if you're unsure how this works).
- PICO-8 numbers have limited accuracy and range; the minimum step between numbers is approximately 0.00002 (0x0.0001), with a range of -32768 (-0x8000) to approximately 32767.99999 (0x7fff.ffff)[If you add 1 to a counter each frame, it will overflow after around 18 minutes!]
- Lua arrays are 1-based by default, not 0-based. FOREACH starts at TBL[1], not TBL[0].
- COS() and SIN() take 0..1 instead of 0..PI*2, and SIN() is inverted.
- SGN(0) returns 1.

- Toggle fullscreen: use OPTION-ENTER on macOS (CMD-F is used for searching text).
- When you want to export a .png cartridge, use SAVE, not EXPORT. EXPORT will save only the spritesheet!

Editor Modes

Press ESC to toggle between console and editor.

Click editing mode tabs at top right to switch or press ALT+LEFT/RIGHT (macOS: OPTION+LEFT/RIGHT)

WARNING: The second half of the sprite sheet (banks 2 and 3), and the bottom half of the map share the same cartridge space. It's up to you how you use the data, but be aware that drawing on the second half of the sprite sheet could corrupt data on the map and vice versa.

macOS users: If you see a command with a PINK CTRL replace the CTRL with a CMD (some commands below conflict with macOS's native commands)

Code Editor

Hold shift to select (or click and drag with mouse)
CTRL-X, C, V to cut copy or paste selected
CTRL-Z, Y to undo, redo
CTRL-F to search for text in the current tab
CTRL-G to repeat the last search again
CTRL-L to jump to a line number
CTRL-UP, DOWN to jump to start or end
ALT-UP, DOWN to navigate to the previous, next function
CTRL-LEFT, RIGHT to jump by word
CTRL-W,E to jump to start or end of current line

CTRL-D to duplicate current line
TAB to indent a selection (shift to un-indent)
CTRL-B to comment / uncomment selected block

To enter special characters that represent buttons (and other glyphs), use SHIFT-L,R,U,D,O,X. There are 3 additional font entry modes that can be toggled:

CTRL_J : Hiragana	type romanji equivalents (ka, ki, ku..)
CTRL-K : Katakana	+ shift-0..9 for extra symbols
CTRL-P Puny font	hold shift for the standard font.

Tabs

Click the [+] button at the top to add a new tab.
Navigate tabs by left-clicking, or with CTRL-TAB*, SHIFT-CTRL-TAB*.
To remove the last tab, delete any contents and then moving off it (CTRL-A, BACKSPACE/DELETE, CTRL-TAB*)
When running a cart, a single program is generated by concatenating all tabs in order.

Code Limits

The current number of code tokens is shown at the bottom right. One program can have a maximum of 8192 tokens. Each token is a word (e.g. variable name) or operator. Pairs of brackets, and strings each count as 1 token. Commas, periods, LOCALs, semi-colons, ENDs, and comments are not counted.

Right click to toggle through other stats (character count, compressed size). If a limit is reached, a warning light will flash. This can be disabled by right-clicking.

Sprite Editor

The sprite editor is designed to be used both for sprite-

wise editing and for freeform pixel-level editing. The sprite navigator at the bottom of the screen provides an 8x8-wise view into the sprite-sheet, but it is possible to use freeform tools (pan, select) when dealing with larger or oddly sized areas.

Draw Tool

- Click and drag on the sprite to plot pixels
- Applies to visible area
- Hold CTRL to search and replace a colour
- Use right mouse button to select colour

Stamp Tool

- Click to stamp whatever is in the copy buffer
- Hold LCTRL to treat colour 0 (black) as transparent

Select Tool (shortcut: LSHIFT or S)

- Create a selection
- Enter or click to select none.

If a pixel-wise selection is not present, many operations are instead applied to a sprite-wise selection, or the visible view. To select sprites, shift-drag in the sprite navigator. To select the spritesheet press CTRL-A (repeat to toggle off the bottom half shared with map data)

Pan Tool (shortcut: spacebar)

- View the spritesheet.

Fill Tool

- Fill with the current colour
- Applies to the current selection
- If no selection, applies to visible area

Shape Tools

- Click the tool button to cycle through: circle, rectangle, line
- Hold CTRL to get a filled circle or rectangle

Extra keys

CTRL-Z to undo
CTRL-C to copy selected area or selected sprites
CTRL-V to paste to current sprite location
Q,W to switch to previous/next sprite
1,2 to switch to previous/next colour
Tab to toggle fullscreen view
Mousewheel or < and > to zoom (centred in fullscreen)

Operations on selected area or selected sprites:

F to flip
V to flip vertically
R to rotate (must be square selection)
Cursor keys to move (loops if sprite selection)

Sprite flags

The 8 coloured circles are sprite flags for the current sprite.
Each one can be true (on) or false (off), and are accessed by using the FSET and FGET functions. They are indexed from 0, from the left (0,1,2..7). See FSET() for more information.

Map Editor

The PICO-8 map is a 128x32 (or 128x64 using shared space) block of 8-bit values. Each value is shown in the editor as a reference to a sprite (0..255), but you can, of course, use the data to represent whatever you like.

The tools are similar to the ones used in sprite editing mode. Select a sprite and click and drag to paint values into the map.

To draw multiple sprites, select from sprite navigator with SHIFT+drag.

To copy a block of values, use the selection (LSHIFT or S) tool and then stamp tool to paste.

To pan around the map, use the pan tool or hold space.

Press Q,W to switch to the previous/next sprite.

Mousewheel or < and > to zoom (centred in fullscreen)

To move sprites in the spritesheet without breaking references to them in the map:

1. Back up your cartridge!
2. Optionally select the area of the map you would like to alter; it defaults to the top half.(Press CTRL-A twice for whole map including shared memory.)
3. Select the sprites you would like to move (while still in map view), and press CTRL-X
4. Select the destination sprite (also while still in map view) and press CTRL-V

SFX Editor

There are 64 SFX ("sound effects") in a cartridge, used for both sound and music.

Each SFX has 32 notes, and each note has:

A frequency	(C0..C5)
An instrument	(0..7)
A volume	(0..7)
An effect	(0..7)

Each SFX also has these properties:

A play speed (SPD) : the number of 'ticks' to play each note for. (1 is fastest, 3 is 3x as slow, etc.)

Loop start and end : this is the note index to loop back and to (Looping is turned off when the start index ≥ end index)

There are 2 modes for editing/viewing a SFX: Pitch mode (more suitable for sound effects) and tracker mode (more

suitable for music). The mode can be changed using the top-left buttons, or toggled with TAB.

1. Pitch Mode

Click and drag on the pitch area to set the frequency for each note, using the currently selected instrument (indicated by colour).

Hold SHIFT to apply only the selected instrument
Hold CTRL to snap entered notes to the C minor pentatonic scale

2. Tracker Mode

Each note shows: frequency octave instrument volume effect
To enter a note, use

|q-2-w-3-e-r-5-t-6-y-7-u-i|
|z-s-x-d-c-v-g-b-h-n-j-m|

(piano-like layout)
Hold SHIFT when entering a note to transpose -1 octave .. +1 octave.
New notes are given the selected instrument/effect values.
To delete a note, use BACKSPACE or set the volume to 0.

Click and then SHIFT-click to select a range that can be copied(CTRL-C) and pasted (CTRL-V). Note that only the selected attributes are copied. Double-click to select all attributes of a single note.

Navigation:
- **PAGEUP/DOWN or CTRL-UP/DOWN to skip up or down 4 notes**
- **HOME/END to jump to the first or last note**

CTRL-LEFT/RIGHT to jump across columns

3. Controls for both modes

- + to navigate the current SFX.
< > to change the speed.
SPACE to play/stop.
SHIFT-SPACE to play from the current SFX quarter (group of 8 notes)
A to release a looping sample
Left click or right click to increase / decrease the SPD or LOOP values (Hold shift when clicking to increase / decrease by 4, alternatively, click and drag left/right or up/down)
SHIFT-click an instrument, effect, or volume to apply to all notes.

Effects

0 none	
1 slide	**Slide to the next note and volume**
2 vibrato	**Rapidly vary the pitch within one quarter-tone**
3 drop	**Rapidly drop the frequency to very low values**
4 fade in	**Ramp the volume up from 0**
5 fade out	**Ramp the volume down to 0**
6 arpeggio fast	**Iterate over groups of 4 notes at speed of 4**
7 arpeggio slow	**Iterate over groups of 4 notes at speed of 8**
	If the SFX speed is ≤ 8, arpeggio speeds are halved to 2, 4

Music Editor

Music in PICO-8 is controlled by a sequence of 'patterns'. Each pattern is a list of 4 numbers indicating which SFX will be played on that channel.

Flow control

Playback flow can be controlled using the 3 buttons at the top right.

When a pattern has finished playing, the next pattern is played unless:

- there is no data left to play (music stops)
- a STOP command is set on that pattern (the third button)
- a LOOP BACK command is set (the 2nd button), in which case the music player searches back for a pattern with the LOOP START command set (the first button) or returns to pattern 0 if none is found.

When a pattern has SFXes with different speeds, the pattern finishes playing when the left-most non-looping channel has finished playing. This can be used to set up time signatures that don't divide into 32, double-time drum beats, etc.

Copying music between or within cartridges

To select a range of patterns: click once on the first pattern in the pattern navigator, then SHIFT-click on the last pattern. Selected patterns can be copied and pasted with CTRL-C and CTRL-V. When pasting into another cartridge, the SFX that each pattern points to

will also be pasted (possibly with a different index) if it does not already exist.

SFX Instruments

In addition to the 8 built-in instruments, custom instruments can be defined using the first 8 SFX. Use the toggle button to the right of the instruments to select an index, which will show up in the instrument channel as green instead of pink.

When an SFX instrument note is played, it essentially triggers that SFX, but alters the note's attributes:

- Pitch is added relative to C2
- Volume is multiplied
- Effects are applied on top of the SFX instrument's effects

For example, a simple tremolo effect could be implemented by defining an instrument in SFX 0 that rapidly alternates between volume 5 and 2. When using this instrument to play a note, the volume can further be altered as usual (via the volume channel or using the fade in/out effects). In this way, SFX instruments can be used to control combinations of detailed changes in volume, pitch and texture.

SFX instruments are only re-triggered when the pitch changes, or the previous note has zero volume. This is useful for instruments that change more slowly over time.

For example: a bell that gradually fades out. To invert this behaviour, effect 3 (normally 'drop') can be used when triggering the note. All other effect values have their usual meaning when triggering SFX instruments.

Lua Syntax Primer

PICO-8 programs are written using Lua syntax, but do not use the standard Lua library. The following is a brief summary of essential Lua syntax. For more details, or to find out about proper Lua, see www.lua.org.

Comments

```
-- use two hyphens like this to ignore everything until the end of the line
--[[ multi-line
comments ]]
```

Types and assignment

Types in Lua are numbers, strings, booleans and tables:

```
num = 12/100
s = "this is a string"
b = false
t = {1,2,3}
```

Numbers in PICO-8 are all 16:16 fixed point. They range from -32768.0 to 32767.99999

Hexadecimal notation with optional fractional parts can be used:

```
0x11            17
0x11.4000       17.25
```

Numbers written in decimal are rounded to the closest fixed point value. To see the 32-bit hexadecimal representation, use print(tostr(val,true)):

```
?tostr(-32768,true)             0x8000.0000
?tostr(32767.99999,true)        0x7fff.ffff
```

Dividing by zero evaluates to 0x7fff.ffff if positive, or -0x7fff.ffff if negative.

Conditionals

```
IF NOT B THEN
    PRINT("B IS FALSE")
ELSE
    PRINT("B IS NOT FALSE")
END

-- WITH ELSEIF

IF X == 0 THEN
    PRINT("X IS 0")
ELSEIF X < 0 THEN
    PRINT("X IS NEGATIVE")
ELSEIF X > 0 THEN
    PRINT("X IS POSITIVE")
ELSE
    PRINT("THIS IS LINE IS NEVER REACHED")
END

IF (4 == 4) THEN PRINT("EQUAL") END
IF (4 ~= 3) THEN PRINT("NOT EQUAL") END
IF (4 <= 4) THEN PRINT("LESS THAN OR EQUAL") END
IF (4 > 3) THEN PRINT("MORE THAN") END
```

Loops

```
FOR X=1,5 DO
    PRINT(X)
END
-- PRINTS 1,2,3,4,5

X = 1
WHILE(X <= 5) DO
    PRINT(X)
    X = X + 1
END

FOR X=1,10,3 DO PRINT(X) END   -- 1,4,7,10

FOR X=5,1,-2 DO PRINT(X) END  -- 5,3,1
```

Functions and Local Variables

```
y=0
function plusone(x)
    local y = x+1
    return y
end
print(plusone(2)) -- 3
print(y)          -- 0
```

Tables

In Lua, tables are a collection of key-value pairs where the key and value types can both be mixed. They can be used as arrays by indexing them with integers.

```
a={} -- create an empty table
a[1] = "blah"
a[2] = 42
a["foo"] = {1,2,3}

-- arrays use 1-based indexing by default

a = {11,12,13,14}
print(a[2]) -- 12

-- the size of a table indexed with sequential 1-based integers:

print(#a)   -- 4

-- indexes that are strings can be written using dot notation

player = {}
player.x = 2 -- is equivalent to player["x"]
player.y = 3

-- see also the tables section in the api reference below.
```

PICO-8 Shorthand

PICO-8 also allows several non-standard, shorter ways to write common patterns.

1. IF THEN END statements, and WHILE THEN END can be written on a single line with:

```
IF (NOT B) I=1 J=2

-- IS EQUIVALENT TO: IF NOT B THEN I=1 J=2 END
-- NOTE THAT BRACKETS AROUND THE SHORT-HAND CONDITION
   ARE REQUIRED.
```

2. Assignment operators

Shorthand assignment operators can also be used if the whole statement is on one line. They can be constructed by appending a '=' to any binary operator, including arithemtic (+=, -= ..), bitwise (&=, |= ..) or the string concatenation operator (..=):

```
A += 2   -- EQUIVALENT TO: A = A + 2
```

3. != operator

Not shorthand, but pico-8 also accepts != instead of ~= for "not equal to"

API

PICO-8 is built on the Lua programming language, but does not include the Lua standard library. Instead, a small API is offered in keeping with PICO-8's minimal design and limited screen space. For an example program that uses most of the API functions, see /DEMOS/API.P8

Functions are written here as:

```
FUNCTION_NAME PARAMETER [OPTIONAL_PARAMETER]
```

System functions called from commandline can omit the usual brackets and string quotes:

```
LOAD BLAH.P8 --> LOAD("BLAH.P8")
```

System

```
LOAD FILENAME [BREADCRUMB [PARAM_STR]]
SAVE FILENAME
```

Load or save a cartridge.

When loading from a running cartridge, the loaded cartridge is immediately run with parameter string param_str, and a menu item is inserted and named breadcrumb, that returns the user to the loading cartridge.

Filenames that start with '#' are taken to be a BBS cart followed by its ID:

```
LOAD("#1234") -- DOWNLOAD [AND RUN] CART NUMBER 1234
```

If the ID is the cart's parent post, or a revision number is not specified, then the latest version is

fetched. BBS carts can be loaded from other BBS carts or local carts, but not from exported carts.

FOLDER

Open the cart's folder in the host OS.

LS (also aliased as DIR)

List files in the current directory. When called from a running program, returns a list of all .p8 and .p8.png files in the same directory.

RUN

Run from the start of the program
Can be called from inside a program to reset program.

STOP [MESSAGE]

Stop the cart and optionally print a message.

RESUME

Resume the program. Use R for short.

Use a single "." from the commandline to advance a single frame. This enters frame-by-frame mode, that can be read with STAT(110). While frame-by-frame mode is active, entering an empty command (by pressing enter) advances one frames.

REBOOT

Reboot the machine
Useful for starting a new project

INFO

Print out some information about the cartridge:
Code size, tokens, compressed size

Also displayed:

UNSAVED CHANGES	When the cartridge in memory differs to the one on disk
EXTERNAL CHANGES	When the cartridge on disk has changed since it was loaded (e.g. by editing the program using a separate text editor)

FLIP

Flip the back buffer to screen and wait for next frame (30fps)
Don't normally need to do this _DRAW() calls it for you.

If your program does not call flip before a frame is up, and a _DRAW() callback is not in progress, the current contents of the back buffer are copied to screen.

PRINTH STR [FILENAME] [OVERWRITE]

Print a string to host operating system's console for debugging.

If filename is set, append the string to a file on the

host operating system (in the current directory -- use FOLDER to view) Setting overwrite to true causes that file to be overwritten rather than appended. Use a filename of "@CLIP" to write to the host's clipboard. (use STAT(4) to read the clipboard, but the contents of the clipboard are only available after pressing CTRL-V during runtime (for security reasons)).

TIME / T

Returns the number of seconds elapsed since the cartridge was run.

This is not the real-world time, but is calculated by counting the number of times _UPDATE or _UPDATE(60) is called. Multiple calls of TIME() from the same frame return the same result.

STAT X

Get system status where X is:

0	Memory usage (0..2048)
1	CPU used since last flip (1.0 == 100% CPU at 30fps)
4	Clipboard contents (after user has pressed CTRL-V)
6	Parameter string
7	Current framerate
16..19	Index of currently playing SFX on channels 0..3
20..23	Note number (0..31) on channel 0..3
24	Currently playing pattern index
25	Total patterns played
26	Ticks played on current pattern
80..85	UTC time: year, month, day, hour, minute, second
90..95	Local time

100	Current breadcrumb label, or nil
110	Returns true when in frame-by-frame mode

EXTCMD X

Special system command, where X is a string:

"PAUSE"	request the pause menu be opened
"RESET"	request a cart reset
"GO_BACK"	follow the current breadcrumb (if there is one)
"LABEL"	set cart label
"SCREEN"	save a screenshot
"REC"	set video start point
"VIDEO"	save a .gif to desktop
"AUDIO_REC"	start recording audio
"AUDIO_END"	save recorded audio to desktop
"SHUTDOWN"	quit cartridge (from exported binary)

Program Structure

There are 3 special functions that, if defined by the user, are called during program execution:

_UPDATE()

Called once per update at 30fps

_DRAW()

Called once per visible frame

_INIT()

Called once on program startup

_DRAW() is normally called at 30fps, but if it can not complete in time, PICO-8 will attempt to run at 15fps and call _UPDATE() twice per visible frame to compensate.

Running PICO-8 at 60fps

If _update60() is defined instead of _update(), PICO-8 will run in 60fps mode:

- both _update60() and _draw() are called at 60fps
- half the PICO-8 CPU is available per frame before dropping down to 30fps

Note that not all host machines are capable of running at 60fps. Older machines, and / or web versions might also request PICO-8 to run at 30 fps (or 15 fps), even when the PICO-8 CPU is not over capacity. In this case, multiple _update60() calls are made for every _draw() call in the same way.

#include

Source code can be injected into a program at cartridge boot (but not during runtime),
using "#include filename", where filename is either a plaintext file (containing Lua code), a tab from another cartridge, or all tabs from another cartridge:

```
#include somecode.lua
#include onetab.p8:1
#include alltabs.p8
```

When the cartridge is run, the contents of each included file is treated as if it had been pasted into the editor in place of that line.

- Filenames are relative to the current cartridge (so, need to save first)
- Includes are not performed recursively.
- Normal character count and token limits apply.

When a cartridge is saved as .p8.png, or exported to a binary, any included files are flattened and saved with the cartridge so that there are no external dependencies.

#include can be used for things like:

- **Sharing code between cartridge (libraries or common multi-cart code)**
- **Using an external code editor without needing to edit the .p8 file directly.**
- **Treating a cartridge as a data file that loads a PICO-8 editing tool to modify it.**
- **Loading and storing data generated by an external (non-PICO-8) tool.**

Graphics

PICO-8 has a fixed capacity of 128 8x8 sprites, plus another 128 that overlap with the bottom half of the map data ("shared data"). These 256 sprites are collectively called the sprite sheet, and can be thought of as a 128x128 pixel image.

All of PICO-8's drawing operations are subject to the current draw state. The draw state includes a camera position (for adding an offset to all coordinates), palette mapping (for recolouring sprites), clipping rectangle, a drawing colour, and a fill pattern.

The draw state is reset each time a program is run. This is equivalent to calling:

```
clip() camera() pal() color()
```

Colours indexes: (use American spellings)

0 black	1 dark_blue	2 dark_purple	3 dark_green
4 brown	5 dark_gray	6 light_gray	7 white
8 red	9 orange	10 yellow	11 green
12 blue	13 indigo	14 pink	15 peach

CLIP [X Y W H]

Sets the screen's clipping region in pixels
CLIP() to reset

PGET X Y
PSET X Y [C]

Get or set the colour (C) of a pixel at X, Y.

SGET X Y
SSET X Y [C]

Get or set the colour (C) of a spritesheet pixel.

SGET N [F]
SSET N [F] V

Get or set the value (V) of a sprite's flag
F is the flag index 0..7
V is boolean and can be true or false

The initial state of flags 0..7 are settable in the sprite editor, using the line of little colourful buttons.
The meaning of sprite flags is up to the user, or can be used to indicate which group ('layer') of sprites should be drawn by map.

If the flag index is omitted, all flags are retrieved/ set as a bitfield:

```
FSET(2, 1+2+8)   -- SETS BITS 0,1 AND 3
FSET(2, 4, TRUE) -- SETS BIT 4
PRINT(FGET(2))   -- 27 (1+2+8+16)
```

PRINT STR X Y [COL]

PRINT STR [COL]

Print a string and optionally set the colour to col.

If x and y are not supplied, the text will automatically wrap and cause carriage returns / vertical scrolling similar to the command prompt.

CURSOR X Y [COL]

Set the cursor position and carriage return margin
If col is specified, also set the current colour.

COLOR [COL]

Set the current colour to be used by drawing functions
If col is not specified, the current colour is set to 6

CLS [COL]

Clear the screen and reset the clipping rectangle.
col defaults to 0 (black)

CAMERA [X Y]

Set a screen offset of -x, -y for all drawing operations CAMERA() to reset

CIRC X Y R [COL]
CIRCFILL X Y R [COL]

Draw a circle or filled circle at X,Y with radius R
If R is negative, the circle is not drawn

LINE X0 Y0 [X1 Y1] [COL]

Draw line, if x1,y1 are not given the end of the last drawn line is used.

```
RECT x0 y0 x1 y1 [col]
RECTFILL x0 y0 x1 y1 [col]
```

Draw a rectangle or filled rectangle.

```
PAL c0 c1 [p]
```

Swap palette colour c0 for c1, either in subsequent draw calls, or screen-wide.

PICO-8's draw state has two palettes. p specifies which one to modify, and defaults to 0:

0 draw palette: colours are remapped on draw (e.g. to re-colour sprites)
1 screen palette: colours are remapped on display (e.g. for fades or screen-wide effects)

For example, to swap colour 12 (blue) for colour 8 (red) when drawing a sprite:

```
PAL(8,12)
SPR(1,60,60)
```

PAL() to reset to system defaults (including transparency values and fill pattern)

```
PAL tbl [p]
```

When the first parameter of pal is a table, colours are assigned for each entry.

For example, to re-map colour 12 and 14 to red:

```
PAL{[12]=9, [14]=8}
```

Or to re-colour the whole screen shades of grey (including everything that is already drawn):

```
PAL{1,1,5,5,5,6,7,13,6,7,7,6,13,6,7}, 1)
```

PALT C [T]

Set transparency for colour index to T (boolean)
Transparency is observed by SPR(), SSPR(), MAP() and TLINE().

```
PALT [8, TRUE]
```

red pixels not drawn in subsequent sprite/TLINE draw calls. PALT[] resets to default: all colours opaque except colour 0.

When c is the only parameter, it is treated as a bitfield used to set all 16 values. For example: to set colours 0 and 1 as transparent:

```
PALT(0B1100000000000000)
```

SPR N X Y [W H] [FLIP_X] [FLIP_Y]

draw sprite N (0..255) at position X,Y
width and height are 1,1 by default and specify how many sprites wide to blit.
Colour 0 drawn as transparent by default (see PALT[])
FLIP_X=TRUE to flip horizontally
FLIP_Y=TRUE to flip vertically

SSPR SX SY SW SH DX DY [DW DH] [FLIP_X] [FLIP_Y]

Stretch rectangle from sprite sheet (SX, SY, SW, SH) [given in pixels] and draw in rectangle (DX, DY, DW, DH). Colour 0 drawn as transparent by default (see PALT()). DW, DH defaults to SW, SH.
FLIP_X=TRUE to flip horizontally
FLIP_Y=TRUE to flip vertically

FILLP P

The PICO-8 fill pattern is a 4x4 2-colour tiled pattern observed by:

```
CIRC()
CIRCFILL()
RECT()
RECTFILL()
PSET()
LINE()
```

P is a bitfield in reading order starting from the highest bit. To calculate the value of p for a desired pattern, add the bit values together:

32768	16384	8192	4096
2048	1024	512	256
128	64	32	16
8	4	2	1

For example, FILLP(4+8+64+128+256+512+4096+8192) would create a checkerboard pattern.

This can be more neatly expressed in binary:

```
FILLP(0b0011001111001100)
```

The default fill pattern is 0, which means a single solid colour is drawn.

To specify a second colour for the pattern, use the high bits of any colour parameter:

```
FILLP(0B0011010101101000)
CIRCFILL(64,64,20, 0X4E)
-- BROWN AND PINK
```

An additional bit 0B0.1 can be set to indicate that the second colour is not drawn.

```
FILLP(0B0011001111001100.1)
-- CHECKERBOARD WITH TRANSPARENT SQUARES
```

The fill pattern can also be set by setting bits in any colour parameter:

```
POKE(0X5F34, 1)
-- SETS INTEGRATED FILLPATTERN + COLOUR MODE
   CIRCFILL(64,64,20, 0X114E.ABCD)
-- SETS FILL PATTERN TO ABCD

-- BIT  0X1000.0000 MEANS THE NON-COLOUR BITS
   SHOULD BE OBSERVED
-- BIT  0X0100.0000 TRANSPARENCY BIT
-- BITS 0X00FF.0000 ARE THE USUAL COLOUR BITS
-- BITS 0X0000.FFFF ARE INTERPRETED AS THE FILL
   PATTERN
```

TABLES

ADD T V

Add value V to the end of table T.
Equivalent to T[#T+1] = V

```
FOO={}              Create empty table
```

```
ADD(FOO, 11)
ADD(FOO, 22)
PRINT(FOO[2])     22
```

DEL T V

Delete the first instance of value V in table T
The remaining entries are shifted left one index to avoid holes.
Note that V is the value of the item to be deleted, not the index into the table!
DEL() can be called safely on a table's item while iterating over that table.

```
A={1,10,2,11,3,12}
FOR ITEM IN ALL(A) DO
    IF (ITEM < 10) THEN DEL(A, ITEM) END
END
FOREACH(A, PRINT) -- 10,11,12
PRINT(A[3])       -- 12
```

ALL T

Used in FOR loops to iterate over all items in a table (that have a 1-based integer index), in the order they were added.

```
T = {11,12,13};
ADD(T,14)
ADD(T,"HI")
FOR V IN ALL(T) DO PRINT(V) END -- 11 12 13 14 HI
PRINT(#T) -- 5
```

FOREACH T F

For each item in table T, call function F with the item as a single parameter.

```
FOREACH (T, PRINT)
```

PAIRS T

Used in FOR loops to iterate over table T, providing both the key and value for each item.
Unlike ALL(), PAIRS() iterates over every item regardless of indexing scheme.
Order is not guaranteed.

```
T = {["HELLO"]=3, [10]="BLAH"}
T.BLUE = 5;
FOR K,V IN PAIRS(T) DO
    PRINT("K: "..K.."  V:"..V)
END
```

Output:

```
K: 10  V:BLAH
K: HELLO  V:3
K: BLUE  V:5
```

Input

BTN [I] [P]

get button I state for player P (default 0)
I: 0..5: left right up down button_o button_x
P: player index 0..7

Instead of using a number for I, it is also possible to use a button glyph.
(In the coded editor, use SHIFT - L R U D O X)

If no parameters supplied, returns a bitfield of all 12 button states for player 0 & 1 (P0: bits 0..5 P1: bits 8..13)

Default keyboard mappings to player buttons:

Player 0: [DPAD]: Cursor Keys, [O]: Z C N [X]: X V M
Player 1: [DPAD]: SFED, [O]: LSHIFT [X]: TAB W Q A

** Note for cart authors: when using a physical gamepad, certain combinations of buttons can be awkward (UP to jump/accelerate instead of [X] or [O]) or even impossible (LEFT + RIGHT)

BTNP [I [P]]

BTNP is short for "Button Pressed"; Instead of being true when a button is held down, BTNP returns true when a button is down AND it was not down the last frame. It also repeats after 15 frames, returning true every 4 frames after that (at 30fps -- double that at 60fps). This can be used for things like menu navigation or grid-wise player movement.

The state of BTNP is reset at the start of each call to _UPDATE or _UPDATE60, so it is preferable to use BTNP from inside one of those functions.

Custom delays (in frames @ 30fps) can be set by poking the following memory addresses:

POKE(0X5F5C, DELAY)	set the initial delay before repeating. 255 means never repeat.
POKE(0X5F5D, DELAY)	set the repeating delay.

In both cases, 0 can be used for the default behaviour (delays 15 and 4)

Audio

SFX N [CHANNEL [OFFSET [LENGTH]]]

play SFX N on channel (0..3) from note offset (0..31) for length notes
N -1 to stop sound on that channel
N -2 to release sound on that channel from looping
Any music playing on the channel will be halted
offset in number of notes (0..31)

channel -1 (default) to automatically choose a channel that is not being used
channel -2 to stop the sound from playing on any channel

MUSIC [N [FADE_LEN [CHANNEL_MASK]]]

Play music starting from pattern N (0..63).
N -1 to stop music.
FADE_LEN in ms (default: 0).
CHANNEL_MASK specifies which channels to reserve for music only (e.g. to play on channels 0..2: 1+2+4 = 7).

Reserved channels can still be used to play sound effects on, but only when that channel index is explicitly requested by SFX().

Map

The PICO-8 map is a 128x32 grid of 8-bit cells, or 128x64 when using the shared memory. When using the map editor, the meaning of each cell is taken to be an index into the spritesheet (0..255). However, it can instead be used as a general block of data.

MGET X Y
MSET X Y V

get or set map value (V) at X,Y

MAP CELL_X CELL_Y SX SY CELL_W CELL_H [LAYERS]

Draw section of map (starting from CELL_X, CELL_Y) at screen position SX, SY (pixels)

MAP(0, 0, 20, 20, 4, 2) -- DRAWS A 4X2 BLOCKS OF CELLS STARTING FROM 0,0 IN THE MAP, TO THE SCREEN AT 20,20

If CELL_W and CELL_H are not specified, defaults to 128, 32 (the top half of the map). To draw the whole map (including the bottom half shared with the spritesheet), use:

```
MAP(0, 0, 0, 0, 128, 32)
```

Layers is an 8-bit bitfield. When it is specified, only sprites with matching flags are drawn.
For example, when layers is 0x5, only sprites with flag 0 and 2 are drawn.

Sprite 0 is always taken to mean empty, and is never drawn.

TLINE X0 Y0 X1 Y1 MX MY [MDX MDY] [LAYERS]

Draw a textured line from (X0,Y0) to (X1,Y1), sampling colour values from the map.
When layers is specified, only sprites with matching flags are drawn (similar to MAP())

MX, MY are map coordinates to sample from, given in tiles.
Colour values are sampled from the 8x8 sprite present at each map tile. For example:

> 2.0, 1.0 means the top left corner of the sprite at position 2,1 on the map. 2.5, 1.5 means pixel (4,4) of the same sprite.

MDX, MDY are deltas added to MX, MY after each pixel is drawn. (Defaults to 0.125, 0).

The map coordinates (MX, MY) are masked by values calculated by subtracting 0x0.0001 from the values at address 0x5f38 and 0x5f39. In simpler terms, this means you can loop a section of the map by poking the width and height you want to loop within, as long as they are powers of 2 (2,4,8,16..).

For example, to loop every 8 tiles horizontally, and every 4 tiles vertically:

```
POKE(0x5f38, 8)
POKE(0x5f39, 4)
tline(...)
```

The default values (0,0) gives a masks of 0xff.ffff, which means that the samples will loop every 256 tiles.

An offset to sample from (also in tiles) can also be specified at addresses 0x5f3a, 0x5f3b

```
POKE(0x5f3a, offset_x)
POKE(0x5f3b, offset_y)
```

Memory

PICO-8 has 3 types of memory:

1. Base RAM (32k): see layout below. Access with peek() poke() memcpy() memset()
2. Cart ROM (32k): same layout as base RAM until 0x4300
3. Lua RAM (2MB): compiled program + variables

(Technical note: You probably don't need to know ^this.)

While using the editor, the data being modified is in cart ROM, but API functions such as spr() and sfx() only operate on base RAM. PICO-8 automatically copies cart ROM to base RAM (i.e. calls reload()) in 3 cases:

1. When a cartridge is loaded
2. When a cartridge is run
3. When exiting any of the editor modes (can turn off with: poke(0x5f37,1))

Base RAM memory layout

```
0X0       GFX
0X1000    GFX2/MAP2 (SHARED)
0X2000    MAP
0X3000    GFX FLAGS
0X3100    SONG
0X3200    SFX
0X4300    USER DATA
0X5E00    PERSISTENT CART DATA (256 BYTES)
0X5F00    DRAW STATE
0X5F40    HARDWARE STATE
0X5F80    GPIO PINS (128 BYTES)
0X6000    SCREEN (8K)
```

User data has no particular meaning and can be used for anything via `MEMCPY()`, `PEEK()` & `POKE()`. Persistent cart data is mapped to `0X5E00..0X5EFF` but only stored if `CARTDATA()` has been called. Colour format (`GFX`/`SCREEN`) is 2 pixels per byte: low bits encode the left pixel of each pair. Map format is one byte per cell, where each byte normally encodes a sprite index.

```
PEEK ADDR
POKE ADDR VAL
```

Read and write one byte to an address in base RAM.
Legal addresses are `0X0..0X7FFF`
Reading out of range returns 0
Writing out of range causes a runtime error

```
PEEK2 ADDR
POKE2 ADDR VAL
PEEK4 ADDR
POKE4 ADDR VAL
```

16-bit and 32-bit versions. Read and write one number (val) in little-endian format:

- 16 bit: `0XFFFF.0000`
- 32 bit: `0XFFFF.FFFF`

`ADDR` does not need to be aligned to 2 or 4-byte boundaries.

Alternatively, the following operators can be used to PEEK (but not POKE), and are slightly faster:

```
@ADDR       PEEK(ADDR)
%ADDR       PEEK2(ADDR)
$ADDR       PEEK4(ADDR)
```

MEMCPY DEST_ADDR SOURCE_ADDR LEN

Copy LEN bytes of base RAM from source to dest. Sections can be overlapping.

RELOAD DEST_ADDR SOURCE_ADDR LEN [FILENAME]

Same as MEMCPY, but copies from cart ROM.
The code section (>= 0x4300) is protected and can not be read.
If filename specified, load data from a different cartridge (must be local - BBS cartridges can not be read in this way).

CSTORE DEST_ADDR SOURCE_ADDR LEN [FILENAME]

Same as MEMCPY, but copies from base RAM to cart ROM
CSTORE() is equivalent to CSTORE(0, 0, 0X4300)
Can use for writing tools to construct carts or to visualise the state of the map / spritesheet using the map editor / GFX editor.
The code section (>= 0X4300) is protected and can not be written to. If a filename is specified, the data is written directly to that cartridge on disk. Up to 64 cartridges can be written in one session. See the 'Cartridge Data' section for additional notes on using CSTORE.

```
MEMSET DEST_ADDR VAL LEN
```

Set LEN bytes to VAL (quite fast; can use to draw unclipped horizontal scan-lines ASOASF)

Math

```
MAX X Y
MIN X Y
MID X Y Z
```

Returns the maximum, minimum, or middle value of parameters. For example, MID(7,5,10) returns 7

```
FLR  X
CEIL X
```

Returns the closest integer that is equal to or below / above x.

```
?FLR ( 4.1) -->  4
?CEIL( 4.1) -->  5
?FLR (-2.3) --> -3
?CEIL(-2.3) --> -2
```

```
COS X
SIN X
```

Returns the cosine of X, where 1.0 indicates a full turn. Sin is inverted to suit screen space. e.g. SIN(0.25) returns -1.

If you'd prefer radian-based trig functions without the Y inversion, paste the following snippet near the start of your program:

```
COS1 = COS FUNCTION COS(ANGLE) RETURN COS1(ANGLE/
(3.1415*2)) END
SIN1 = SIN FUNCTION SIN(ANGLE) RETURN -SIN1(ANGLE/
(3.1415*2)) END
```

ATAN2 DX DY

Converts DX, DY into an angle from 0..1
As with COS/SIN, angle is taken to run anticlockwise in screenspace
e.g. ATAN2(1, -1) returns 0.125

SQRT X

Return the square root of X

ABS X

Returns the absolute (positive) value of X

RND X

Returns a random number n, where 0 <= n < x
If you want an integer, use FLR(RND(X))
If X is an array-style table, return a random element between TABLE[1] and TABLE[#TABLE].

SRAND X

Sets the random number seed.
The seed is automatically randomised on cart startup.

Bitwise Operations

Bitwise operations are similar to logical expressions, except that they work at the bit level.

Say you have two numbers (written here in binary using the "0b" prefix):

```
x = 0b1010
y = 0b0110
```

A bitwise AND will give you bits set when the corresponding bits in x *and* y are both set:

```
print(band(x,y)) -- result:0b0010 (2 in decimal)
```

There are 9 bitwise functions available in PICO-8:

```
band x y -- both bits are set
bor  x y -- either bit is set
bxor x y -- either bit is set, but not both of them
bnot x   -- each bit is not set
shl  x n -- shift left n bits (zeros come in from the
            right)
shr  x n -- arithmetic right shift (the left-most bit
            state is duplicated)
lshr x n -- logical right shift (zeros comes in from
            the left)
rotl x n -- rotate all bits in x left by n places
rotr x n -- rotate all bits in x right by n places
```

Operator versions are also available: &, |, ^^, ~, <<, >>, >>>, <<>, >><< (respectively)

For example:

```
print(67 & 63) -- result:3  equivalent to band(67,63)
```

Operators are slightly faster than their corresponding functions. They behave exactly the same, except that if any operands are not numbers the result is a runtime error (the function versions instead default to a value of 0).

Integer Division

Integer division can be performed with a \

```
PRINT(9\2) -- RESULT:4  EQUIVALENT TO FLR(9/2)
```

Custom Menu Items

MENUITEM INDEX [LABEL CALLBACK]

Add an extra item to the pause menu

Index should be 1..5 and determines the order each menu item is displayed.
LABEL should be a string up to 16 characters long
CALLBACK is a function called when the item is selected by the users

When no label or function is supplied, the menu item is removed:

e.g.:

```
MENUITEM(1, "RESTART PUZZLE", FUNCTION()
RESET_PUZZLE() SFX(10) END)
```

Strings

```
s = "the quick brown fox"

-- length

    print(#s)          -- 19

-- joining strings

    print("three "..4) -- "three 4"

-- sub() to grab substrings

    print(sub(s,5,9))  -- "quick"
    print(sub(s,5))    -- "quick brown fox"

-- conversion

tostr(17)      -- returns "17"
tostr(17,true) -- returns "0x0011.0000"
tonum("17")    -- returns 17
chr(64)        -- returns "@"
ord("@")       -- returns 64
ord("123",2)   -- returns 50 (the second character: "2")
```

Types

type val

Returns the name of the type of value *x* as a string.

tostr val [hex]

Returns *val* as a string.

If hex is true and val is a number, an unsigned hexadecimal writing of the number is returned in the format "0x0000.0000". You can use this to inspect the internal representation of PICO-8 numbers.

If VAL is a boolean, it is written as "true" or "false".

All other VAL types are written as "[typename]" (use hex to include a hex identity in the string if available)

TONUM VAL

Converts VAL to a number.
If VAL is a string, the number is taken to be decimal unless prefixed with "0X" if the conversion fails, TONUM returns no value.

Cartridge Data

Using CARTDATA(), DSET(), and DGET(), 64 numbers (256 bytes) of persistent data can be stored on the user's PICO-8 that persists after the cart is unloaded or PICO-8 is shutdown. This can be used as a lightweight way to store things like high scores or to save player progress. It can also be used to share data across cartridges / cartridge versions.

If you need more than 256 bytes, it is also possible to write directly to the cartridge using CSTORE(). The disadvantage is that the data is tied to that particular version of the cartridge. e.g. if a game is updated, players will lose their save games. Also, some space in the data sections of the cartridge need to be left available.

Another alternative is to write directly to a second cartridge by specifying a fourth parameter to CSTORE(). This requires a cart swap (which in reality only means the user needs to watch a spinny cart animation for 1 second).

```
CSTORE(0,0,0X2000, "SPRITESHEET.P8")
    -- LATER:
RELOAD(0,0,0X2000, "SPRITESHEET.P8")
    -- RESTORE THE SAVED DATA
```

CARTDATA ID

CARTDATA() opens a permanent data storage slot indexed by id, that can be used to store and retrieve up to 256 bytes (64 numbers) worth of data using DSET() and DGET().

```
CARTDATA("ZEP_DARK_FOREST")
    -- CAN ONLY BE SET ONCE PER SESSION
    -- LATER IN THE PROGRAM..
DSET(0, SCORE)
```

ID is a string up to 64 characters long, and should be unusual enough that other cartridges do not accidentally use the same ID.

e.g. CARTDATA("ZEP_JELPI")

Legal characters are a..z, 0..9 and underscore (_). Returns true if data was loaded, otherwise false.

CARTDATA can not be called more than once per cartridge execution.

Once a CARTDATA ID has been set, the area of memory 0x5e00..0x5eff is mapped to permanent storage, and can either be accessed directly or via DGET/DSET.

DGET INDEX

Get the number stored at INDEX (0..63)
Use this only after you have called CARTDATA()

DSET INDEX VALUE

Set the number stored at INDEX (0..63)
Use this only after you have called CARTDATA()

There is no need to flush written data; it is automatically saved to permanent storage even if POKE()'ed directly.

GPIO

GPIO stands for "General Purpose Input Output", and allows machines to communicate with each other. PICO-8 maps bytes in the range 0x5f80..0x5fff to GPIO pins that can be POKE()'ed (to output a value; e.g. to make an LED light up) or PEEK()'ed (e.g. to read the state of a switch).

GPIO means different things for different host platforms:

CHIP: 0x5f80..0x5f87 mapped to XIO-P0..XIO-P7
Pocket CHIP: 0x5f82..0x5f87 mapped to GPIO1..GPIO6

(**XIO-P0 & P1 are exposed inside the prototyping area inside the case.)

Raspberry Pi: 0x5f80..0x5f9f mapped to wiringPi pins 0..31
(see http://wiringpi.com/pins/ for mappings on different models.)
(**also: watch out for BCM vs. WiringPi GPIO indexing!)

CHIP and Pi values are all digital: 0 (LOW) and 255 (HIGH)

A simple program to blink any LEDs attached on and off:

```
T = 0
FUNCTION _DRAW()
 CLS(5)
 FOR I=0,7 DO
  VAL = 0
  IF (T % 2 < 1) VAL = 255
  POKE(0X5F80 + I, VAL)
  CIRCFILL(20+I*12,64,4,VAL/11)
 END
 T += 0.1
END
```

SERIAL

For more precise timing, the SERIAL() command can be used. GPIO writes are buffered and dispatched at the end of each frame, allowing clock cycling at higher and/or more regular speeds than is possible by manually bit-banging using POKE() calls.

SERIAL(CHANNEL, ADDRESS, LENGTH)

CHANNEL:

- 0x000..0x0fe
  corresponds to gpio pin numbers; send 0x00 for LOW or 0xFF for HIGH
- 0x0ff
  delay; length is taken to mean "duration" in microseconds (excl. overhead)
- 0x100..0x101
  SPI read/write on channel 0,1 (experimental)
- 0x400..0x401
  ws281x LED string (experimental)

ADDRESS:

The PICO-8 memory location to read from (and subsequently write to in the case of SPI)

LENGTH:

Number of bytes to send. 1/8ths are allowed to send partial bit strings.

For example, to send a byte one bit at a time to a typical APA102 LED string:

```
VAL = 42
    -- VALUE TO SEND
    DAT = 16 CLK = 15
        -- DATA AND CLOCK PINS DEPEND ON DEVICE
    POKE(0X4300,0)
        -- DATA TO SEND (SINGLE BYTES: 0 OR 0XFF)
    POKE(0X4301,0XFF)
    FOR B=0,7 DO
```

```
            -- SEND THE BIT (HIGH FIRST)
            SERIAL(DAT, BAND(VAL, SHL(1,7-B))>0 AND 0X4301 OR
            0X4300, 1)
        -- CYCLE THE CLOCK
        SERIAL(CLK, 0X4301)
        SERIAL(0XFF, 5) -- DELAY 5
        SERIAL(CLK, 0X4300)
        SERIAL(0XFF, 5) -- DELAY 5
    END
```

HTML

Cartridges exported as HTML / .js use a global array of integers (pico8_gpio) to represent GPIO pins. The shell HTML should define the array:

```
VAR PICO8_GPIO = ARRAY(128);
```

Mouse and Keyboard Input

Mouse and keyboard input can be achieved by enabling devkit input mode (experimental! The HTML implementation is still limited & buggy):

```
POKE(0x5F2D, 1)
```

Note that not every PICO-8 will have a keyboard or mouse attached to it, so when posting carts to the Lexaloffle BBS, it is encouraged to make keyboard and/or mouse control optional and off by default, if possible. When devkit input mode is enabled, a message is displayed to BBS users warning them that the program may be expecting input beyond the standard 6-button controllers. The state of the mouse and keyboard can be found in STAT(X):

STAT(30)	(Boolean) True when a keypress is available.
STAT(31)	(String) character returned by keyboard.
STAT(32)	Mouse X
STAT(33)	Mouse Y
STAT(34)	Mouse buttons (bitfield).
STAT(36)	Mouse wheel event.

Additional Lua Features

PICO-8 also exposes 2 features of Lua for advanced users: Metatables and Coroutines.

For more information, please refer to the Lua 5.2 manual.

Metatables

Metatables can be used to define the behaviour of objects under particular operations.
For example, to use tables to represent 2D vectors that can be added together, the '+' operator is redefined by defining an "__add" function for the metatable:

```
vec2d={
 __add=function(a,b)
    return {x=(a.x+b.x), y=(a.y+b.y)}
 end
}

v1={x=2,y=9} setmetatable(v1, vec2d)
v2={x=1,y=5} setmetatable(v2, vec2d)
v3 = v1+v2
print(v3.x..","..v3.y) -- 3,14
```

setmetatable t, m

Set table t metatable to m.

getmetatable t

Return the current metatable for table t, or nil if none is set.

rawset t key value

```
RAWGET T KEY
RAWEQUAL T1 T2
RAWLEN T
```

Raw access to the table, as if no meta-methods were defined.

Function Arguments

The list of function arguments (or remain arguments) can be specified with ...

```
FUNCTION PREPRINT(PRE, S, ...)
    LOCAL S2 = PRE..TOSTR(S)
    PRINT(S2, ...)
        -- PASS THE REMAINING ARGUMENTS ON TO PRINT()
END
```

To accept a variable number of arguments:

```
FUNCTION FOO(...)
    LOCAL ARGS={...}
        -- BECOMES A TABLE OF ARGUMENTS
    FOREACH(ARGS,PRINT)
END
```

Coroutines

Coroutines offer a way to run different parts of a program in a somewhat concurrent way, similar to threads. A function can be called as a coroutine, suspended with YIELD() any number of times, and then resumed again at the same points.

```
FUNCTION HEY()
    PRINT("DOING SOMETHING")
    YIELD()
    PRINT("DOING THE NEXT THING")
    YIELD()
    PRINT("FINISHED")
END
```

```
C = COCREATE(HEY)
FOR I=1,3 DO CORESUME(C) END
```

COCREATE F

Create a COROUTINE for function F.

CORESUME C [P0 P1 ..]

Run or continue the coroutine c. Parameters p0, p1.. are passed to the coroutine's function. Returns true if the coroutine completes without any errors. Returns false, error_message if there is an error. (**Runtime errors that occur inside coroutines do not cause the program to stop running. It is a good idea to wrap CORESUME() inside an ASSERT(). If the assert fails, it will print the error message generated by CORESUME.)

```
ASSERT(CORESUME(C))
```

COSTATUS C

Return the status of coroutine C as a string:

- "running"
- "suspended"
- "dead"

YIELD

Suspend execution and return to the caller.