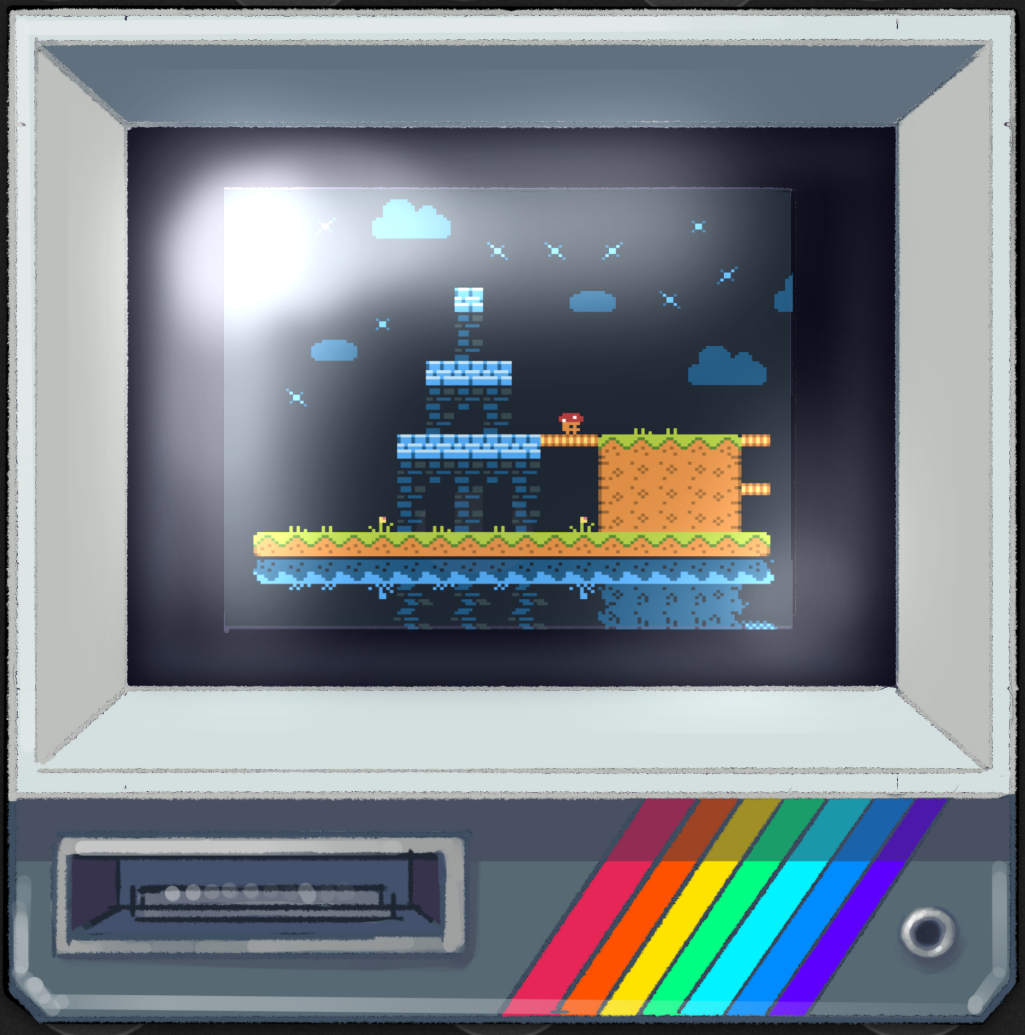


POQ3

Ver. 1.0 User Manual

by Pennie Quinn

PQ93

Thank you for checking out PQ93! I hope you enjoy it. If you need any assistance, please start a community topic on pennie.itch.io/pq93 or send me a message at pennie.ada.quinn@gmail.com

Platforms	macOS, Windows, Ubuntu, web (export)
Resolution	160x144 (Gameboy Color)
Colors	16 (arne16 palette), customizable
Audio	4 voices, 8 instruments
Cart Language	MoonScript
Cart Size	Unrestricted

Other lovely fantasy consoles inspired me to write my own, and this is it.

PQ93 is a fantasy console written in modern C++ (previously in C). it uses Gameboy resolution (160x144), the arne16 color palette, and MoonScript. it runs on Windows, macOS, Ubuntu, and (soon) Raspbian.

Included in PQ93 is a command shell and editors for code, sprites, sound effects, music patterns, and maps. you can edit your game cartridges in any external text editor you like, and you can import and export sprite sheets via command.

Your games can be published natively on The Big 3 platforms using the prebuilt "projectors" (see: Sharing Your Games).

You can also build your game for the web using the export command.

TABLE OF CONTENTS

Manual

Config

You can configure PQ93 by placing a config.txt file in your save directory, which you can open in your file browser using the folder shell command. You can find your save directory using the ‘folder’ command.

I recommend setting your virtual drive root to a folder in DropBox, iCloud, et cetera, to keep your games and screenshots synced across devices:

```
#config.txt
drive = "/some/absolute/path/on/your/machine"
```

Game Inputs

MOONSCRIPT PRIMER

MoonScript, as creator leafo says, is "CoffeeScript for Lua". If hipster programming languages aren't your cup of JavaScript, the tl;dr is that MoonScript is a concise, *delightfully expressive* and thus *powerful* language built in Lua.

Here i will try to give a 'quick rundown' of the core features and differences from regular Lua:

- ⋆ everything is local by default, unless exported
- ⋆ you can forward-declare local variables like this: local x,y,z
- ⋆ scope is indentation defined, unless otherwise delimited (like table definitions)
- ⋆ OOP is a built-in feature
- ⋆ list and table comprehensions, oh my!
- ⋆ a gazillion other niceties you should try out

For further information and many great examples, I highly recommending checking out moonscript.org.

Hello World Example

Below is the source code for the "wheel.p93" cartridge:

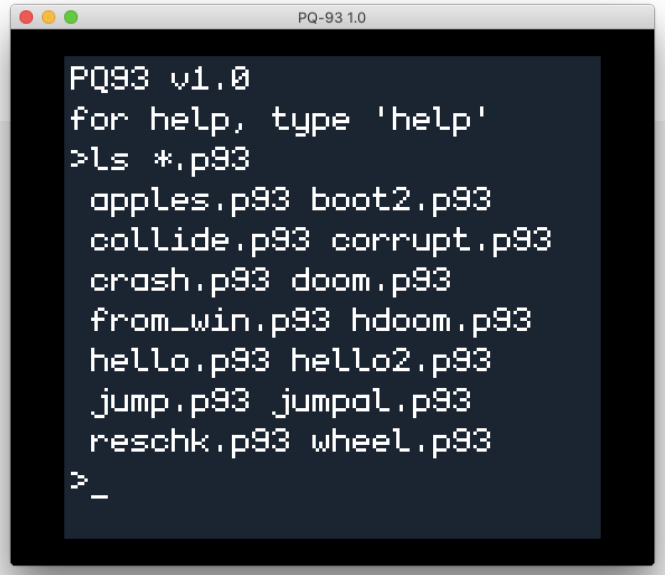
```
cx,cy = WIDTH/2, HEIGHT/2
ticks=0

export _init=->
  cls!

export _update=->
  ticks += 1

export _draw=->
  cls!
  f = ticks/120.0
  for i=1,32 do circ cx+sin(i/32)*64,cy+cos(i/32)*64,6,flr(f*i)%15+1
  for i=1,24 do circf cx+sin(i/24)*48,cy+cos(i/24)*48,6,flr(f*i)%15+1
  for i=1,16 do circ cx+sin(i/16)*32,cy+cos(i/16)*32,6,flr(f*i)%15+1
  s = 'PQ93 ;)'
  w = txtw s
  txt cx-w/2,cy-4, s, 8
```

COMMAND SHELL

After booting PQ93, you will be dropped into the command shell.

Here you can explore the file system, load and run your games, etc.

You can also export your games to run in a browser!



Esc	toggle between the Shell and Editor
Up, Down	scroll through the command history (current session)
Return	submit command

Commands

Note: if you haven't seen a command shell before, square brackets around an argument or list of arguments means they are optional! :)

- help [cmd]
  - list commands, or show help for a specific one

- ls [filter]
  - list entries in the working directory. you can filter with wildcards, as one would expect. ls *.p93 will list all local cartridges, for example

- cd <dir>
  - move to a different directory. you are restricted to PQ93's virtual drive

- pwd
  - show the working directory

- mkdir <dir>
  - create a directory

- load <file>

- load a cartridge. **NOTE: loading a new cart will lose any unsaved changes to the current one!**

- save [file]
  - save the loaded cartridge. for new cartridges, you must supply file. otherwise, it is optional unless you want to "Save As..."

- run [file]
  - will run the loaded cartridge or the one specified by file. **NOTE: running a different cart will lose any unsaved changes to the current one!**

- folder
  - attempts to open the working directory in your file explorer

- exit
  - close PQ93 **(discards unsaved changes!)**

- egfx [-s i] <file>
  - export a cart's graphics as a PNG
  - you can export all 4 sheets in one image (256x256 px), or supply a specific sheet (128x128 px) to export with -s i, range [0, 3]

- igfx [-s i] <file>
  - import a PNG into the cart's graphics
  - if importing all sheets with igfx foo.png, you must supply a 256x256 PNG file
  - if importing a single sheet, you must supply a 128x128 PNG file.
  - **PQ93 will colorfit the image as best it can, but it is recommended to use the arne16 palette for drawing in external editors.**

- reboot
  - restart PQ93. This clears any loaded cartridge and resets the system palette.

- reload
  - reload a loaded cart. This is useful if you edit your cartridges in an external text editor.

- pal [file]
  - **experimental**! load a custom color palette from a text file can be useful if your game uses a custom palette, and you need to import sprites for it.
  - call without file to reset to the default palette

- epal <file>
  - export the system palette to a text file

- export "My Game" [main] [data1] [data2] ...
  - exports a cart as an html5 game. Will generate a folder in the working directory:

    ```
    My Game/
       index.html
       projector.js
    ```
  - if a cart is loaded, it will be used by default in the export.

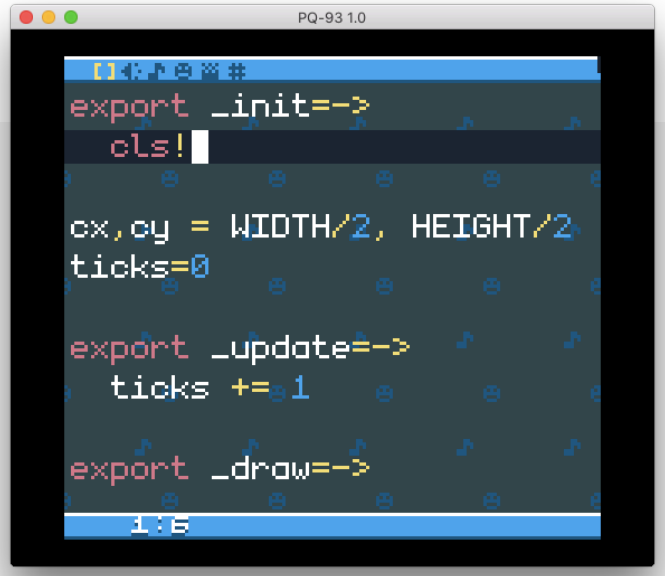
- if you are writing a multicart, you can list all the carts needed. Just make sure the 'main' cart is first in the list!

CODE EDITOR

The first editor tab is the code editor.

You may also use an external text editor and 'reload' your cartridge.

Your game can have as much or as little code as you like.



Esc	toggle between the Shell and Editor
Ctrl+X	cut the current line
Ctrl+C	copy the current line
Ctrl+V	paste a previously cut line *or* text from the OS clipboard

System Callbacks

PQ93 looks for 3 entry points to your game cartridge: exported functions called _init, _update, and _draw. Note that each of these is technically optional, but you will at least want to implement _draw!

Here is the outline to a typical cartridge:

```
-- My Cool Game, by Me
local player, enemies

export _init=->
  -- do setup here

export _update=->
  -- perform your game logic

export _draw=->
  -- draw your scene
```

The Scripting API

PQ93 provides various functions you can call to interact with the system. You can display sprites, draw shapes, play sound and music, draw maps, (temporarily) modify your game's data on the fly, etc.

For a complete list of available system calls, see the API Reference.

SFX EDITOR

Here, you can edit sounds effects and music tracks.

The buttons in the top right corner toggle between the piano roll-style view and the tracker view (pictured).

Esc	toggle between the Shell and Editor
Ctrl+X	cut the selected note(s)
Ctrl+C	copy the selected note(s)
Ctrl+V	paste copied or cut note(s)
Space	play the current sound

Each cartridge may define up to 64 SFX with 32 beats.

There are 8 instruments and 5 octaves of pitches, 5 note effects, and 8 possible amplitudes. SFX may contain loop points, so that when they finish playing, they return to their loop's start and continue.

The speed of an SFX determines how many *frames* each beat lasts for, and PQ93 runs at 60Hz. So, for example, an SFX with speed 15 will have 4 beats per second.

The SFX editor has 2 modes: a "painting" mode and a "tracker" mode. In both, you can press **Space** to play or stop the sound.

Painting Mode

In painting mode, pitches of the upper 3 octaves are graphed, and you can click and drag to lay down notes across them. The vertical axis represents pitch (quantized into half steps), and the horizontal axis represents beats.

Beneath the "piano roll", there is a panel for painting the volume of the notes.

On the left side, there are buttons for the 8 instruments. Select one to paint with, or **Shift+Click** to make all *existing* notes change to that instrument.

Tracker Mode

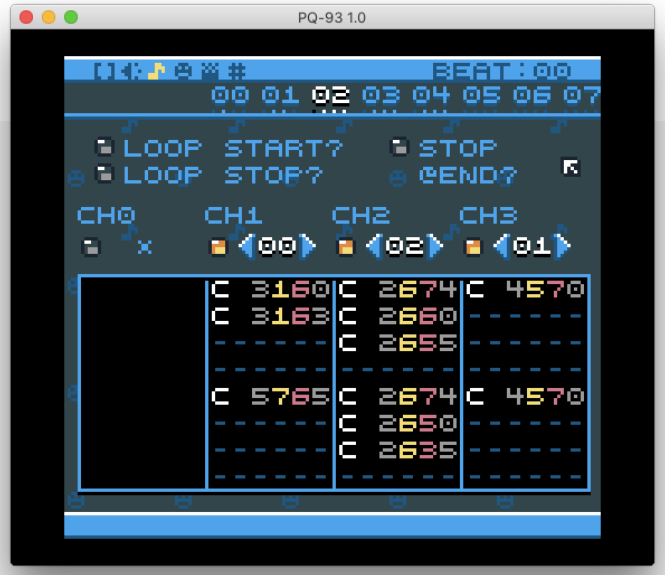
In tracker mode, each beat of the SFX is a line of text: the pitch, followed by the octave, then the instrument, then the amplitude, then the modulator (if any).

To add notes, navigate the track with the arrow keys or your mouse. With a pitch cell highlighted, press a key from **A** to **J** (the white keys of the “piano”) or from **W, E, T, Y,** or **U** (the black keys).

Similar to painting mode, you can **Shift+Click** an instrument or effect to apply it to all existing notes. Also similar, the selected instrument, effect, octave, and amplitude will apply to new notes added.

You can also select, copy, and paste notes between and within SFX using the standard keyboard shortcuts.

MUSIC EDITOR



The music editor allows you to organize sound effects you have made into patterns.

There are only four audio sources, so you may want to leave one open for your game's sound effects!

Esc	toggle between the Shell and Editor

A cartridge can hold up to 64 music patterns. Each pattern specifies which SFX to play with which voices.

There are also 3 Boolean flags: "loop start", "loop stop", and "stop at end".

Loop stop specifies whether or not a pattern is the "end" of a looping song. When a "loop stop" pattern finishes, the nearest "loop start" pattern *before* it will begin to play.

This is useful for background music.

Stop at end specifies whether music playback should simply halt when the pattern finishes playing.

This is better suited for transitions, fanfares, etc.

SPRITE EDITOR

In this tab, you can draw graphics for your game.

You can edit your graphics in 8x8 or 16x16 mode.

You can also set up to 8 Boolean flags for each 8x8 tile.

Esc	toggle between the Shell and Editor
Ctrl+X	cut the selected sprite or region
Ctrl+C	copy the selected sprite or region
Ctrl+V	paste copied or cut sprite
Mouse Wheel	select a sprite quickly

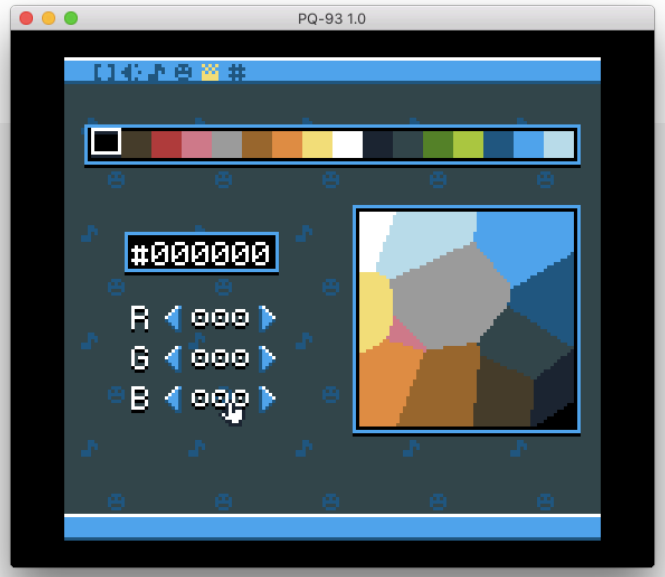
An individual cartridge can hold 4 128x128px sprite sheets. That equates to 256 8x8 tiles per sheet, or 64 16x16 tiles.

You can copy and paste entire tiles around with the standard key shortcuts, but you can use the marquis tool to select a portion of a sprite to copy instead. When you copy, the current tool automatically changes to the stamp, which is used for pasting.

Also, each 8x8 tile has 8 Boolean flags associated with it. This allows you to conveniently associate some basic data with your sprites without writing extra code.

For example, in your game, flag 7 being set might mean that a sprite should be solid, while flag 6 might mean that it gets drawn in front of the player, and flag 5 could mean that it is a collectible item...

PALETTE EDITOR



The palette editor allows you to change the system palette of PQ93.

This is an experimental feature which can get messy, fast!

If you make a palette you like, you can export it with the ‘epal’ command.

Esc	toggle between the Shell and Editor

The palette editor is not quite feature complete yet! However, you can still use it to create a custom palette if you like.

Having a custom palette loaded is useful when importing graphics using an alternate palette from the default. PQ93 tries to color-fit imported graphics, and you don’t want that when you intend to *load a specific palette* in your game’s code.

So you can create or load a custom palette, then import the sprites down with it in your favorite sprite editor, save your cartridge, and reset the system palette with the ‘pal’ command, and your art will remain unmodified.

MAP EDITOR

The map editor provides a convenient way to lay out simple tile maps.

You can use the editor in normal mode (pictured), or you may hide the GUI to see more of the map at once.

Maps can use 8x8 or 16x16 tiles.

Esc	toggle between the Shell and Editor
Ctrl+X	cut the selected region
Ctrl+C	copy the selected region
Space	hold down to scroll by **Click+Dragging**
Mouse Wheel	quickly select a brush tile

Each cartridge can store up to 16x16 screens' worth of 8x8px tiles. That is, 16x16 chunks of 20x18 tiles. However, if you decide to use 16x16px tiles for your map, you can store 32x32 chunks of 10x9 tiles.

Hold **Space**, then **Click+Drag** to scroll around, or use the **arrow keys** to scroll tile-by-tile. The CHUNK roll boxes can be used to scroll chunk-by-chunk.

Select a tile with your mouse, and paint with it by **clicking** or **Click+Dragging** on the map grid. To get a better view, you can hide the GUI by clicking the big rectangle in the top middle of the screen.

Using the marquis tool, you can copy a region of tiles to paint with, using the stamp tool. This is handy when you have repeating, multi-tile structures.

Maps & Sheets

Since a sheet holds 256 tiles, and a map tile entry is a byte, a map can only pull tiles from one sheet at a time. In your game, you can specify which sheet to draw with before drawing your map, using the 'sheet' function.

Tip: you can use this to your advantage with *reusable* maps. If you want a night and day version of a town, for example, you can put the night and day tile sprites at the same places on two different sheets, and draw the same map with both sheets.

API REFERENCE

Note: square brackets around an argument or list of arguments means they are optional! To call a function in MoonScript with *no* arguments, use bang (!), like this: foo!

Callbacks

Cartridges are able to supply several *callbacks* which PQ93 uses to interact with them. these are _init, _update, and _draw. Create them like this:

```
export _init= ->
export _update= ->
export _draw= ->
```

Graphics

- cls
  - clears the screen to color 1

- col [i]
  - sets the drawing color to i, or 0 by default

- cam [x, y]
  - sets camera position or resets to (0,0)

- setp x, y, [c]
  - sets the pixel at (x, y) on the screen to c or the drawing color

- getp x, y
  - gets the palette index at (x, y) on the screen

Text

- txt x, y, str, [c]
  - draws str at (x,y) with c or the drawing color
  - respects camera coords

- txtw 'foo'
  - gets the drawing width of a string

- mtxt x, y, str, [c]
  - draws str in mini font at (x,y) with c or the drawing color
  - respects camera coords

- mtxtw
  - gets the drawing width of a string in mini font

Shapes

- circ x, y, r, [c]
  - outlines a circle of radius r at (x,y) with c or the drawing color
  - respects camera coords

- circf x, y, r, [c]

- fills a circle of radius r at (x,y) with c or the drawing color
- respects camera coords

- rec x, y, w, h, [c]
  - outlines a rectangle {x, y, w, h} with c or the drawing color
  - respects camera coords

- recf x, y, w, h, [c]
  - fills a rectangle {x, y, w, h} with c or the drawing color
  - respects camera coords

- ln x0, y0, x1, y1, [c]
  - draws a line from (x0,y0) to (x1,y1) with c or the drawing color
  - respects camera coords

- lnh x0, y0, x1,[c]
  - draws a horizontal line from (x0,y0) to (x1,y0) with c or the drawing color
  - respects camera coords

- lnv x0, y0, y1, [c]
  - draws a vertical line from (x0,y0) to (x0,y1) with c or the drawing color
  - respects camera coords

Palette

- pal [c0, c1]
  - sets pal[c0] to pal[c1], or resets the rendering palette
- pcol [i, color]
  - sets pal[i] to color (24bit, like 0xff00ff), or resets the rendering palette

Sheets

- sheet i
  - sets the sprite sheet used for drawing to sheet i
  - sheets range from 0..3

- sget x, y
  - gets the color at (x, y) on the drawing sheet

- sset x, y, [c]
  - sets the color at (x, y) on the drawing sheet to c or the drawing color

Sprites

- spr i, x, y, [c]
  - draws sprite i at (x, y)
  - if c is supplied, the sprite will be stenciled in that color
  - respects camera coords

- spr i, x, y, w, h,[c]
  - draws sprite i with width and height (w,h) at (x,y)
  - if c is supplied, the sprite will be stenciled in that color

    * respects camera coords

* spr i, x, y, w, h, flpx, flpy
    * draws sprite i with (w,h) at (x,y)
    * flpx, flpy indicate whether or not to *flip* the sprite
    * respects camera coords

* sspr i, x, y, xf, yf
    * draws sprite i at (x, y), scaled by (xf, yf)
    * respects camera coords

* sspr i, x, y, w, h, xf, yf
    * draws sprite i with width and height (w, h) at (x, y), scaled by (xf,yf)
    * respects camera coords

* txrec sx,sy,sw,sh, dx,dy,dw,dh, [flipx, flipy]
    * draws a rectangle of pixels from the drawing sheet {sx, sy, sw, sh} onto a rectangle of the framebuffer {dx, dy, dw, dy},
    * optionally flipped
    * respects camera coords

Map

* mget x, y
    * gets the map tile value at (x,y).

* mset x, y, t
    * sets the map tile at (x, y) to t.

* map8 mx,my,mw,mh, sx,sy, [flags]
    * draws a map with 8px tiles from map region {mx, my, mw, mh} at screen position (sx,sy).
    * flags: if given, will filter tiles drawn by their sprite flags.
    * respects camera coords.

* map8 chunk, sx,sy, [flags]
    * draws a map chunk (screen width by screen height) at screen position (sx, sy).
    * flags: same as above.
    * respects camera coords.

* map16
    * behaves exactly like map8, except it uses 16px tiles.

Triangles

* tri x0,y0, x1,y1, x2,y2, [c]
    * outlines a triangle from (x0,y0) to (x1,y1) and (x2,y2) with color c or the drawing color
    * respects camera coords

* trif x0,y0, x1,y1 ,x2,y2, [c]
    * fills a triangle from (x0,y0) to (x1,y1) and (x2,y2) with color c or the drawing color
    * respects camera coords

- txtri x0,y0, x1,y1, x2,y2, s0,t0, s1,t1, s2,t2
  - draws a textured triangle from the drawing sheet; (x*,y*) are the vertices, (s*,t*) are the texture coordinates (in pixels).
  - respects camera coords.

Flags

- fget i, [flag]
  - if no flag index is supplied, returns a byte holding all flags for sprite i.
  - if flag *is* supplied, returns the boolean value of the flag
  - flags range from 0..7

- fset i, [flag], v
  - if flag *is not* supplied, sets the flags byte of sprite i to v (an integer).
  - if flag *is* supplied, sets the specified flag of sprite i to v (a boolean).

Audio

- sfx i, [channel, [pos, [len]]]
  - plays sound effect i.
  - channel dictates which of the 4 audio channels should be used; they range from 0..3. a value of -1 will choose an empty channel if available.
  - pos specifies which beat to start on.
  - len specifies how many beats to play before stopping.
  - returns the audio channel used to play the sfx.

- mus [i]
  - will play pattern i, or stop all music by default.

- reverb [percent]
  - will set """reverb""" to percent, or reset to zero.
  - try it out in a cave!

Input

- btn b, [t]
  - used to check button state, returns a boolean.
  - b: the button you want to query. possible values are:
    - directional buttons: 'u','d','l','r'.
    - action buttons: 'a', 'b', 'x', 'y', 'start', 'select'.
    - these buttons can also be queried by *index.*
  - t: indicates the button *state* you want to check for.
    - default behavior checks if a button is **down**.
    - 'p' check if a button was **pressed** this frame.
    - 'r' check if a button was **released** this frame.

- mbtn b, [t]
  - used to check mouse buttons, returns a boolean.
  - b: 'l', 'r', or 'm'.
  - t: 'p', or 'r', works as with btn.

- mwheel
  - returns number of "wheel ticks" scrolled this frame.

- mouse
  - returns the position (x, y) of the mouse cursor.

Math

- sin x
  - returns sine of x, where 1.0 represents 360 degrees (2pi, or tau).

- cos x
  - returns cosine of x, where 1.0 represents 360 degrees (2pi, or tau).

- clamp x, min, max
  - returns x clamped to the inclusive range [min, max].

- round x
  - returns x rounded to the nearest integer.

- flr x
  - returns x rounded down to the nearest integer.

- ceil x
  - returns x rounded up to the nearest integer.

- sgn x
  - returns the sign of x as 1 or -1.

Carts

- merge 'cart.p93'
  - loads the assets (not the code!) from the given cart.
  - useful for games needing lots of graphics, maps, etc.
  - only overwrites where loaded assets overlap previous ones.

- run ['cart.p93']
  - will reload and restart the cartridge, or jump to a new one.

- stop
  - will stop the running cartridge.

SHARING YOUR GAMES

Cartridge File

The simplest way to share your game is to distribute the .p93 file itself.

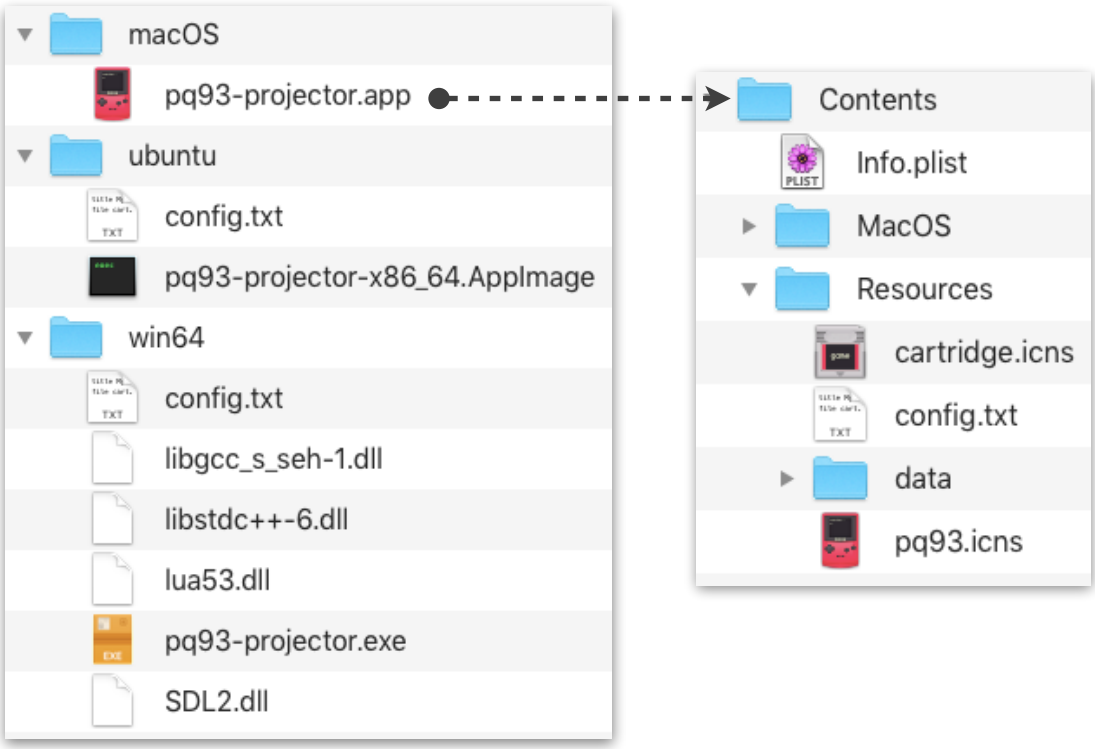
Of course, this will only be convenient if your players *also* have PQ93.

If you want more people to be able to play, keep reading. :)

Desktop (macOS, Ubuntu, Windows)

To publish your game cartridge for desktop, first go to pennie.itch.io/pq93 and scroll down to the Downloads section. Then download the file "pq93-projectors.zip".

Extract the archive, and you will find 3 folders, one for each major platform.



For Ubuntu and Windows, we will be copying files into these directories. For **macOS**, you will need to copy your files into `pq93-projector.app/Contents/Resources/` instead.

If you are using macOS, you will need to **right-click** or **Ctrl+Click** the .app file and select **"Show Package Contents"** to see the inner files.

Now, open the "config.txt" file in *your platform's* folder, and edit the file to match your game:

```
title The Sacred Chalice
file tsc-main.p93
```

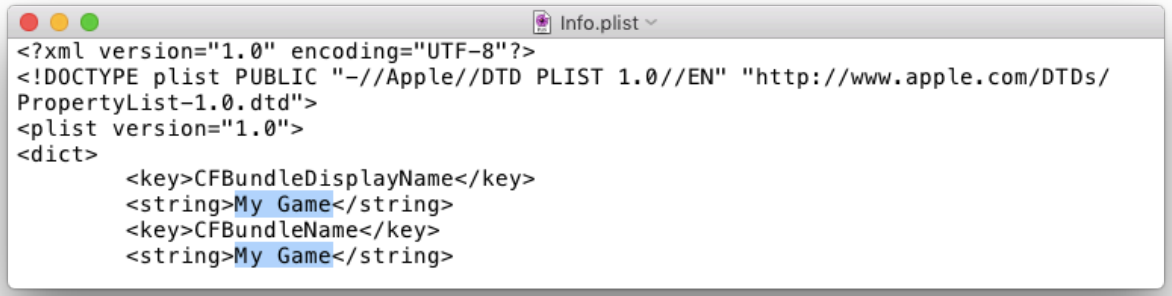
Then, copy your game's cartridge file(s) into the same folder as the config.txt.

Once that's done, run the projector. It should load your game after the boot animation.

If everything works as expected, go ahead and copy the config.txt and your cartridges to the other 2 folders (replace the old files).

You can of course change the name of the projector to match your game, and there are various methods of changing their icons, as well.

Extra Step for macOS: You may want to edit the "Info.plist" file inside the projector, so that macOS users will show the correct title and version information for your game in the About menu or QuickLook:



Once you are happy with your projectors, you can zip up each folder to distribute. :)

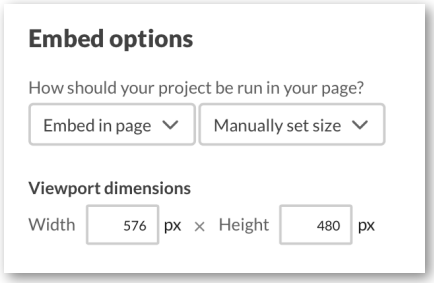
Web (HTML5)

To build your game for the web, first use the 'export' command in the command shell (see: 'export' command).

This should leave you with a folder that looks like this.

If you want to test your game before uploading somewhere, be sure to use a browser which will allow *index.html* to load the local js file! Many browsers disallow this.

If you intend to embed your game in an iframe (on itch.io, for example), remember that the game window expects the iframe to be **576x480px**.



If you *do* decide to publish your game on itch.io, *please* tag the game with 'pq93' so I can find it and play it! :)

thank you.

~Pennie