Build a Retro-game with the PICO-8: The Basics

thenewstack.io/retro-game-pico-8-basics/

Suchakra Sharma 7/1/2016

The geeks among us have all, at least once, toyed with the idea of creating our own games. However, game development requires perseverance and dedication to sit for long hours polishing the fine details of the code and design.

If you are of an impatient kind, want to learn a little bit of Lua programming language and the basics of pixel-art at the same time, PICO-8 console emulator is the tool of your choice. In this two-part series, we will learn how to get addicted to retro-game development with PICO-8!

The Fantasy Console

Yes, this is how PICO-8 markets itself. Developed by Lexaloffe, PICO-8 is technically a small virtual machine that runs on your actual machine much like a console emulator such as ZSNES. However, instead of emulating a console, it is an actual independent console — a *fantasy console* if you may — one that could be an actual hardware as well. And of course, there is Raspberry Pi support for PICO-8 as well! Apart from that, the Next Thing Co. PocketCHIP computer ships this fantasy console in as part of its Linux-based OS. PICO-8 understands Lua and some built-in shell commands. In fact, when you boot it up, there is a 128×128 pixel display with a tiny shell.

Why PICO-8?

There are multiple reasons to start off with PICO-8. The chief reason is scope. Yes, most of us abandon our pet game dev projects because the scope becomes too big – the story stretches on, the artwork required gets more and more complex or the levels remaining are too many while the spare time we have shrinks eventually.

The PICO-8 console allows us to restrict the scope. It has a limited memory (32K cartridge size) and just 128×128 pixels per frame!

Add to that a fixed 16 color palette and you are pretty restricted in your game complexity and design. This is a good thing, in the sense that you need to be more creative and the probability of your game getting completed gets quite high. Apart from that, PICO-8 supports coding in a restricted subset of Lua. There is no complete standard Lua libraries support but only some specific APIs that work. So this can be a fun way to learn Lua! The fantasy console also allows maps to be developed and supports a maximum of 128 8×8 pixel sprites. There is also an in-built code editor, a sprite and map editor and the best of all — a four channel polyphonic music synth! Yes, this is a complete tiny, cute retro-game development studio!

A Quick Intro

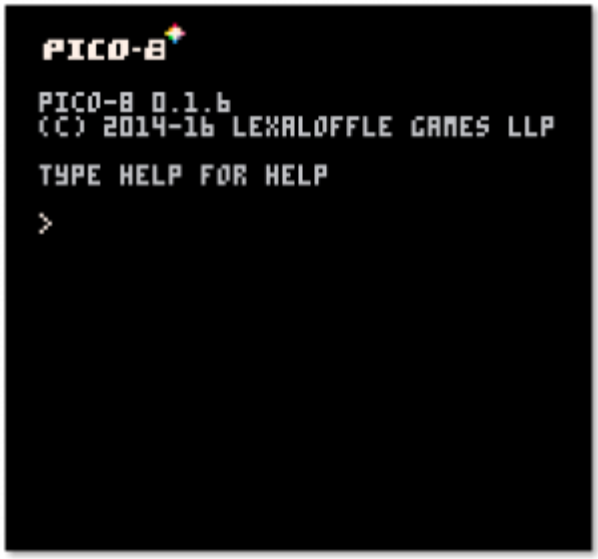
There is a nice collection of resources on how to use PICO-8 console and code in Lua. An official manual is in the works as well. The biggest help, however, are the fanzines that can also be downloaded. Keep them handy for reference.

To get a feel of the console you can directly run any of the community-generated game cartridges exported to HTML. The console loads the cartridge and runs in the browser! For example, Rainmaker is a nice game to get a feel of what all is possible. Now, before getting too bullish lets first start off with a simple goal of creating our 8×8 pixel main character (well call her “Judy”) and moving her around the imaginary 128×128 pixel night sky.

For that, we need to know some basics of the console. Once you obtain the console binary and execute it, you will be greeted with a tiny boot screen and a basic shell where you can give Lua commands. Some of them are quite intuitive such as `LS` (directory listing), `MKDIR` (make a directory) and `HELP` (prints some essential commands). Try the `LOAD, RUN` and `SPLORE` commands for a start.

Just for some fun, you can try some demos:

```
> INSTALL_DEMOS
> CD DEMOS
> LOAD JELPI.P8
> RUN
```



This would start a small inbuilt demo game. Press Esc key to get back to the shell. Ctrl + L clears the screen.

As you have guessed, the `.P8` format is the cartridge format. The cartridge can also be exported as a `.PNG` file (yes an image) which contains the image of the cartridge with the game code and all the sprites as part of the image file itself. Pretty neat! So now if you use the `SPLORE` command and search for Rainmaker, you can directly load the game and run it in your console.

So now that you have played some games to your heart’s content, its time to build one! We will be using some PICO-8 specific APIs along with Lua syntax to familiarize ourselves. Game is a mix of art and code. So, we start with the art aspect first — creating the player sprite. Use `REBOOT` command and reset the console. Now press Esc and you will now see a screen with some tools for game development. There is an editor, a sprite designer, a map designer a sound effects creator and even a track interface for music. For now, we just focus on the **Editor** and the **Sprite Designer**.

On your right, you can see the character sprite and some Lua code for our small game. For creating Judy, we have to use the 16 color palette on the right and fill out the 8×8 pixel box on the left. Let the creative energy flow and draw to your heart's content. As you draw, the sprite (a two-dimensional bitmap integrated into a larger scene) will show up on the tile below. You can Ctrl + C, and Ctrl + V the tiles and make modifications for animating the sprite. You can also spread a single sprite as big as you want – maybe even 2 or 4 8×8 tiles big.

In code, we can define the dimension (such as 8×16, etc. for a single sprite if we wish to). For now, we just stick to a simple 8×8 sprite. As you see in the image, the tile number is 000. In the code window, you can see that the `move()` function is cycling the sprite as the character moves and `player.sprite` is set to 0 (the initial position). This brings us to the coding part. The code window is a bit tiny, however. If it bothers you, you can always edit the `P8` file directly on the disk (`~/.lexaloffle/pico-8/carts/`) in your favorite editor.

There are three main functions that you would be using in your games apart from the global variables (usually): **_init()** which is the initialization routine at program startup, **_update()** which is called once per update at 30 FPS and **_draw()** which is called once per visible frame. Here is the code for tiny goal we have set in which we make *Judy* run across the screen:

```
-- JUDY --
-- Based on demo by Shane Riley's :
https://gist.github.com/shaneriley/cae98eac6136e7293b28 --
player = {}
player.x = 20
player.y = 20
player.sprite = 0
player.speed = 3

function move()
player.moving = true
player.sprite += 1
if player.sprite > 3 then
player.sprite = 0
end
```

```
end

function _update()
player.moving = false
if btn(0) then
player.x -= player.speed
move()
end
if btn(1) then
player.x += player.speed
move()
end
if btn(2) then
player.y -= player.speed
move()
end
if btn(3) then
player.y += player.speed
move()
end
if not player.moving then
player.sprite = 0
end
end

function _draw()
cls()
spr(player.sprite, player.x, player.y)
end
```

view raw JUDY.P8 hosted with ❤ by GitHub

This is not the complete `P8` file but just the Lua code. If you are editing the file on disk, this code goes under the `__lua__` and `__gfx__` section. You can save it and go back to the sprite editor in console. Changes made in the console can be saved by Ctrl + S and the file on disk will be updated with encoded sprites as ASCII in the `P8` file.

In the code above, we first define a Lua table as `player = {}` and then some properties such as `x` and `y` position, sprite, etc. to make the code more readable. The **btn()** API allows the player to move in 4 directions by incrementing or decrementing her `x` and `y` positions. We can call the position change value as `player.speed` as it moves Judy faster or slower. The **spr()** function draws Judy's sprite (0) at the new positions obtained from **_update()**. There is some more tiny logic to make sure the sprite returns to 0 position when there is no more button press event. We are almost done here.

Press Esc and write `SAVE JUDY.PNG` in the console to save the game as a PNG file that you can share with others and play. You can just right-click and download the one you see on your left! You can also give the `EXPORT JUDY.HTML` command to save the game as an HTML and JS file that can be used on your web page. You can try the online JUDY demo here.

So till now, we learned how to write a fun demo in PICO-8 and release it to the public. In the next part, we will build upon our character Judy and put her in the middle of an adventure complete with crazy 8-bit levels, some chip-tune music and a small story of her journey to The New Stack headquarters!

Feature Image: RPG Town Pixel Art Assets by ansimuz licensed under CC 0.