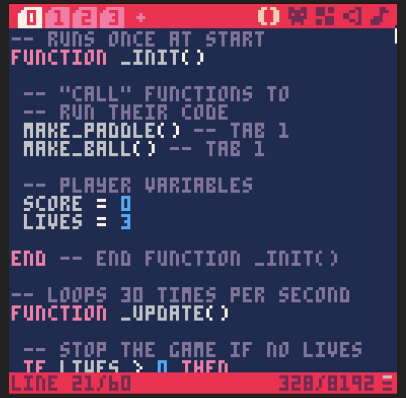
Making a Paddleball Game in PICO-8

Complete Walkthrough



by Matthew DiMatteo

Assistant Professor, Game & Interactive Media Design at Rider University

mdimatteo@rider.edu

as seen in GAM-120: Intro to Game Logic

Contents

- 6) About PICO-8
- 13) Getting Started in the PICO-8 Editor
  - 19) Drawing Sprites
  - 31) Saving Your Work 35) Loading a Saved File
  - 39) Microsoft VS Code as an External Code Editor
- 55) Coding in PICO-8
- 73) The PICO-8 Coordinate Plane

Contents

- 81) The PICO-8 “Game Loop” (Program Structure)
- 86) Using Variables
- 90) Understanding the Game Loop
- 102) Variables, Cont.’d
- 107) Variables and Input
- 110) Rules for Naming Variables
- 113) Detecting Input
- 119) Closing Functions and If Statements

Contents

- 134) Keeping the Paddle on Screen
- 148) Writing and “Calling” Custom Functions
- 161) Collision Detection
- 193) Making the Ball Bounce
- 199) Making the Ball Bounce Sideways
- 212) Bouncing off the Walls & Ceiling
- 217) Resetting the Ball After a Miss

Contents

- 219) Adding Sounds
- 224) Adding Score & Lives
- 232) Implementing a Game Over Screen
- 237) Restarting the Game
- 243) Paddleball with Physics
- 244) Code Examples & PICO-8 Learning Resources

About PICO-8

About PICO-8

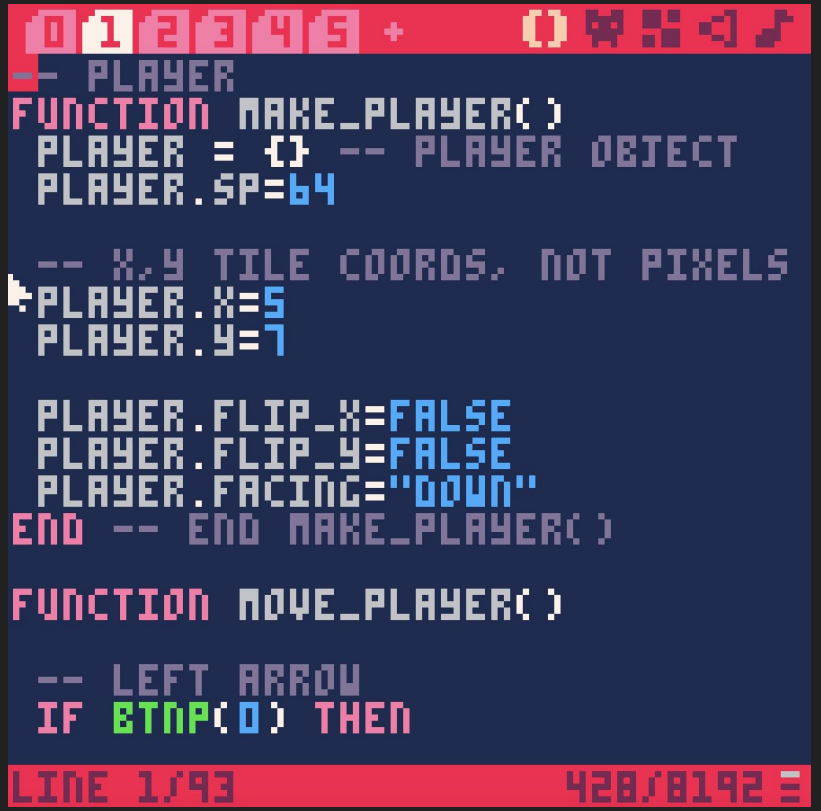
- We'll be using the free-education version, which runs in a browser

Go to pico-8-edu.com

- It has a code editor, sprite editor, map editor, sound editor, and music editor

About PICO-8

PICO-8 has a code editor for writing your code



About PICO-8

PICO-8 has a sprite editor for drawing graphics

You do have to draw all graphics through the editor, no uploading image files

About PICO-8

PICO-8 has a **map editor** for creating your game world

The size is **1024** x **512** px

(8 screens x 4 screens)

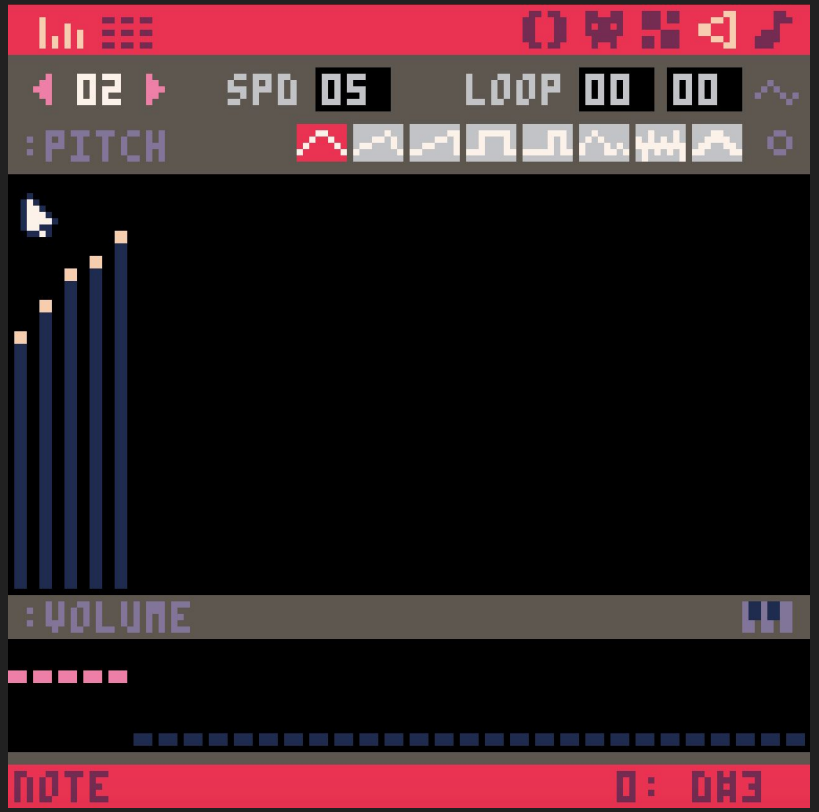
Each screen is 16 x 16 tiles or 128 x 128 px

(each tile is 8 x 8 px)

About PICO-8

PICO-8 has an audio editor for creating sound effects

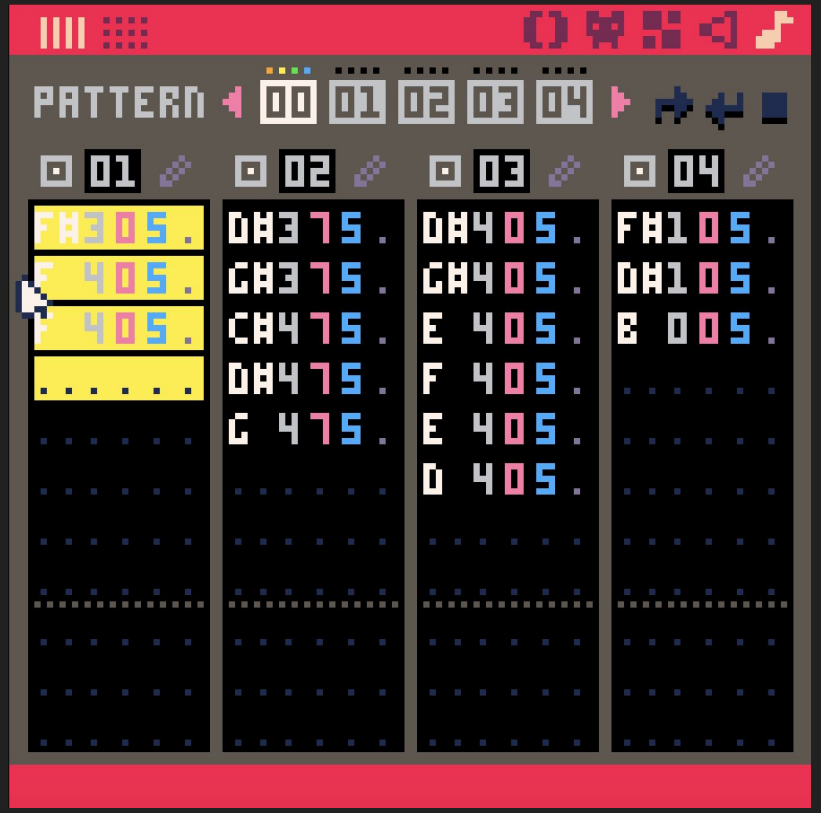
Same deal as with graphics, all must be created in the program, no file uploads



About PICO-8

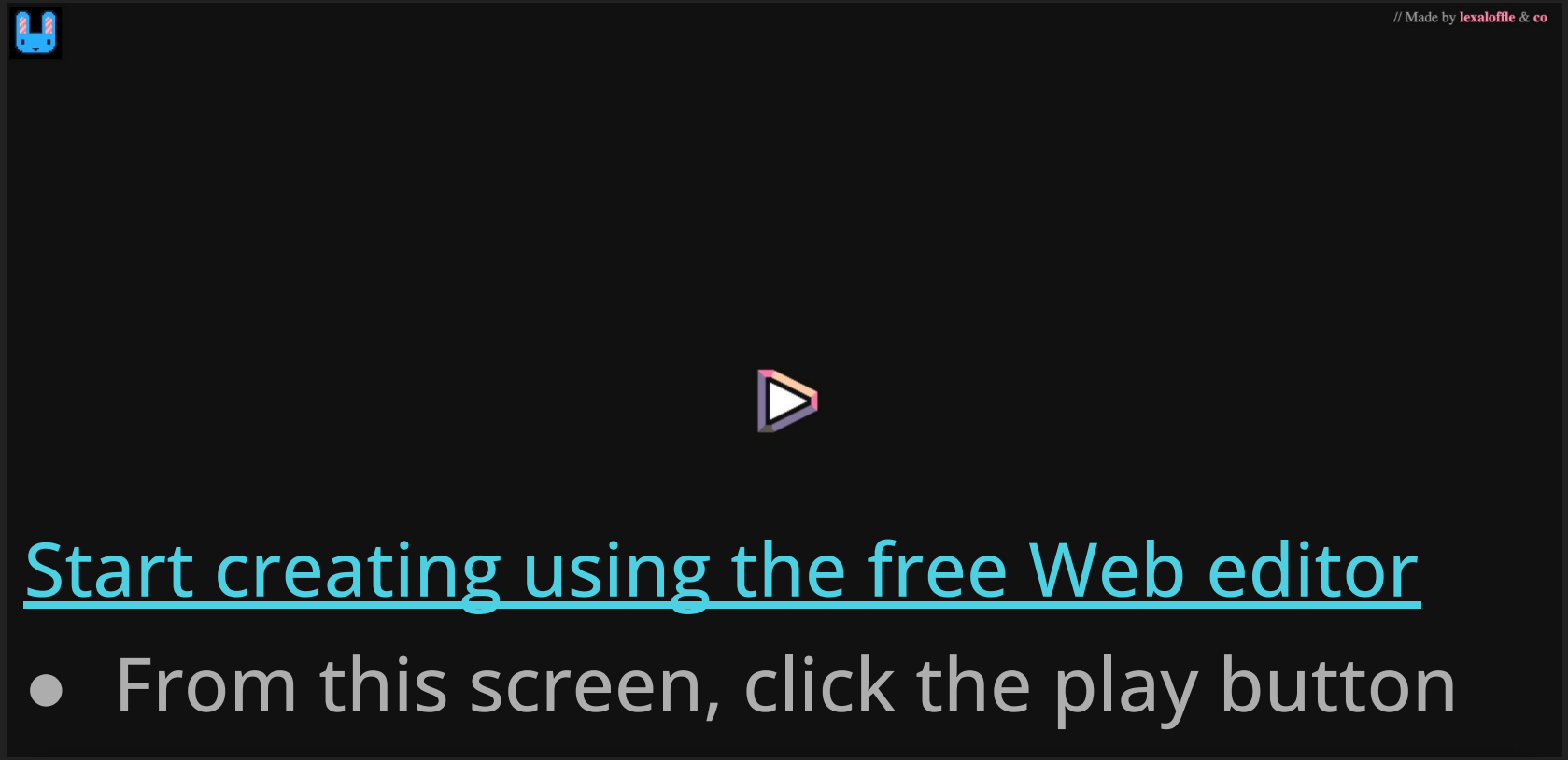
PICO-8 has a music editor for creating chiptunes

Same deal as with graphics, all must be created in the program, no file uploads



Getting Started with PICO-8

Using the PICO-8 Editor



[Start creating using the free Web editor]

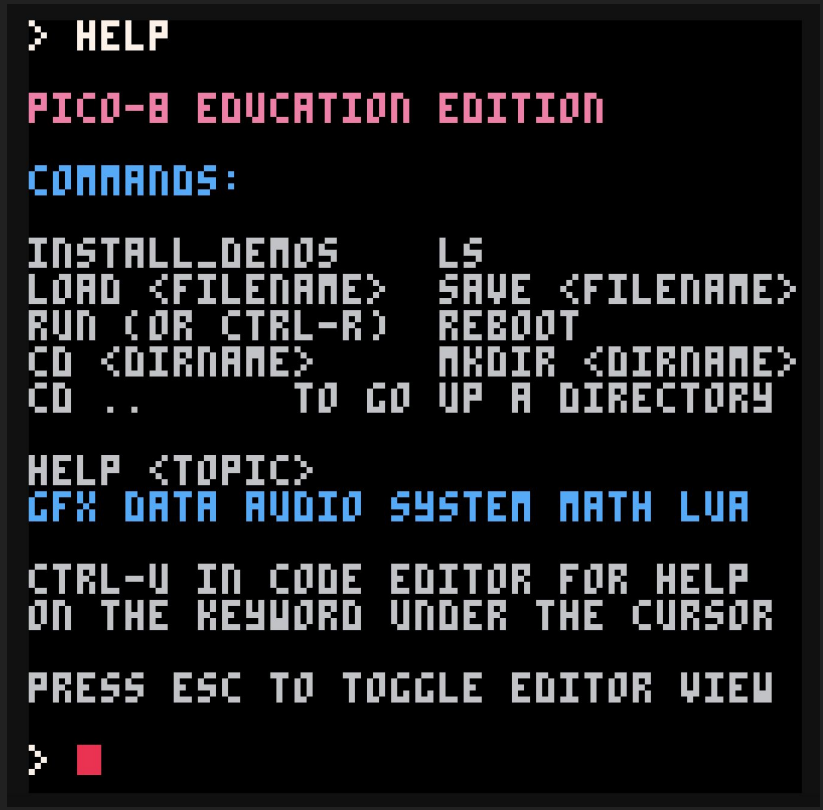
- From this screen, click the play button

Using the PICO-8 Editor

- You'll see a command line interface for entering commands
- Hit Esc to swap between this screen and the editor

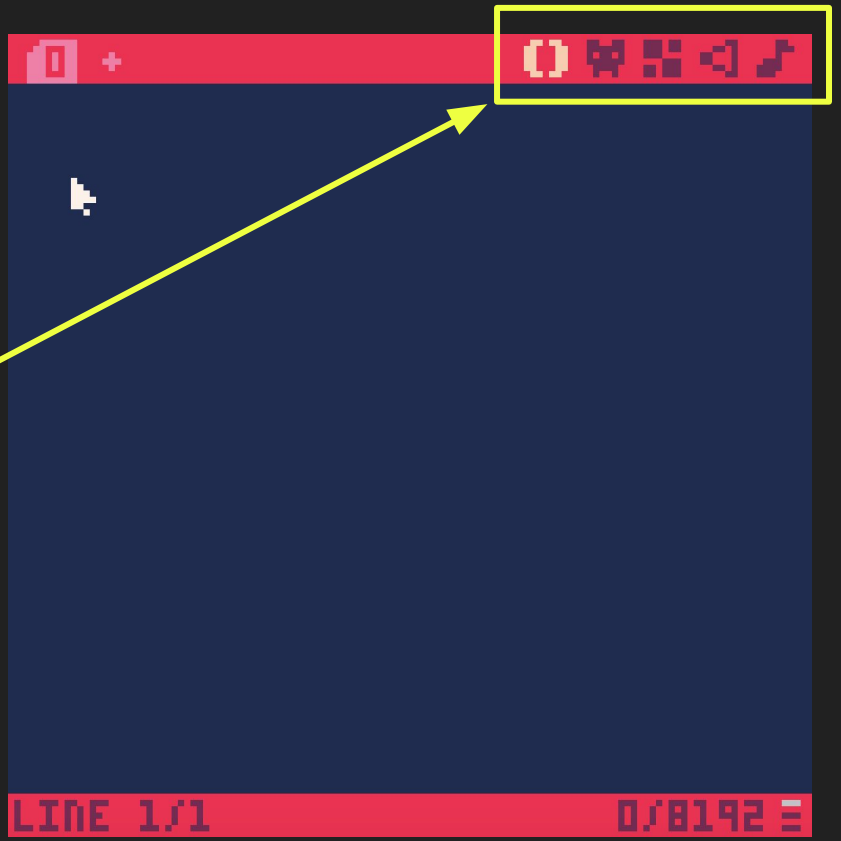
Using the PICO-8 Editor

- If you type HELP and press Enter/Return, you'll see a list of commands
- Hit Esc to swap between this screen and the editor

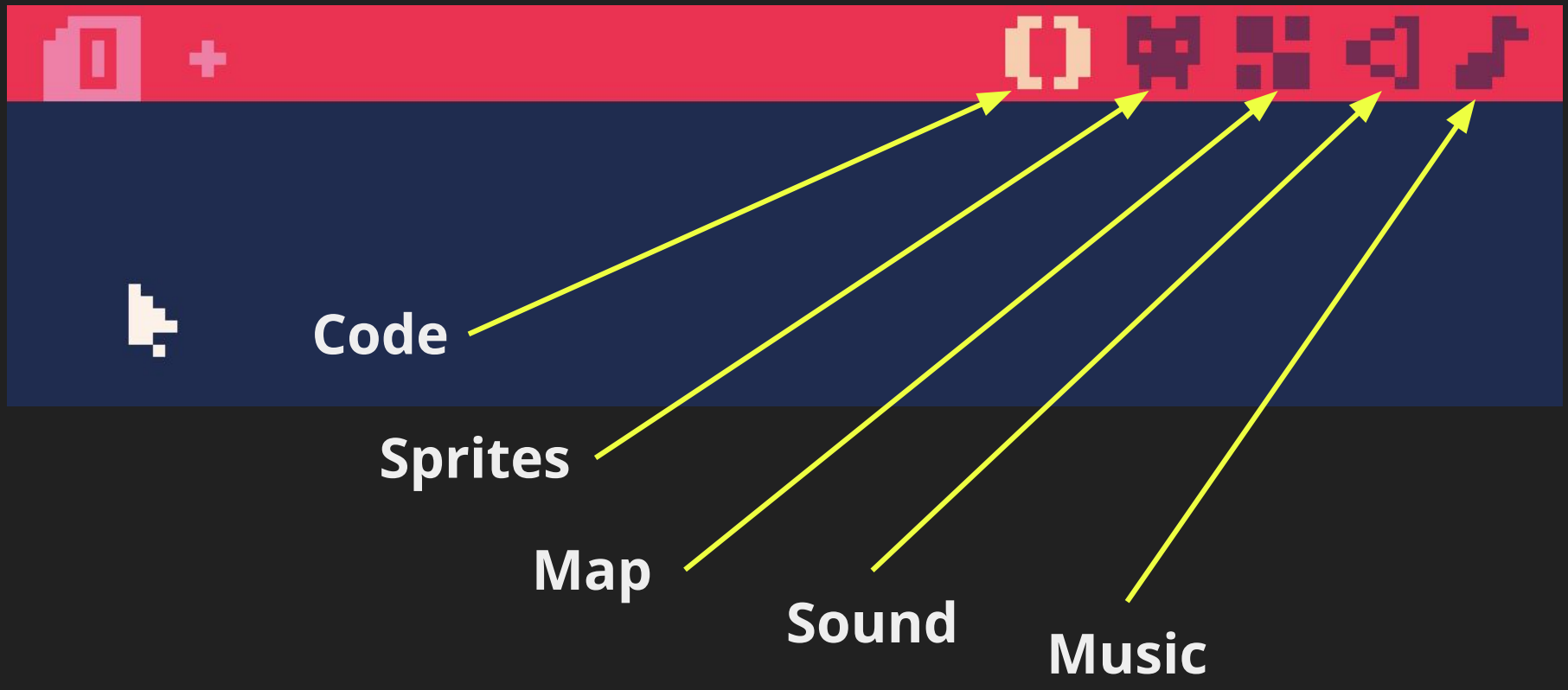


Using the PICO-8 Editor

- Inside the editor, you'll begin on the code editing screen
- Use the top-right navigation to switch to the sprite, map, and sound editors



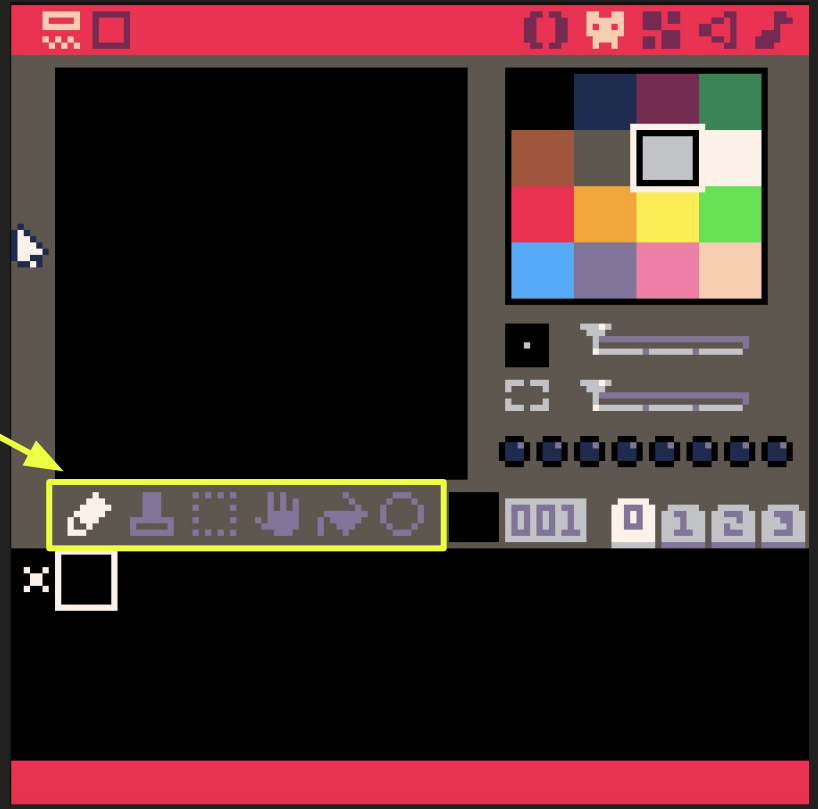
Using the PICO-8 Editor



Drawing Sprites

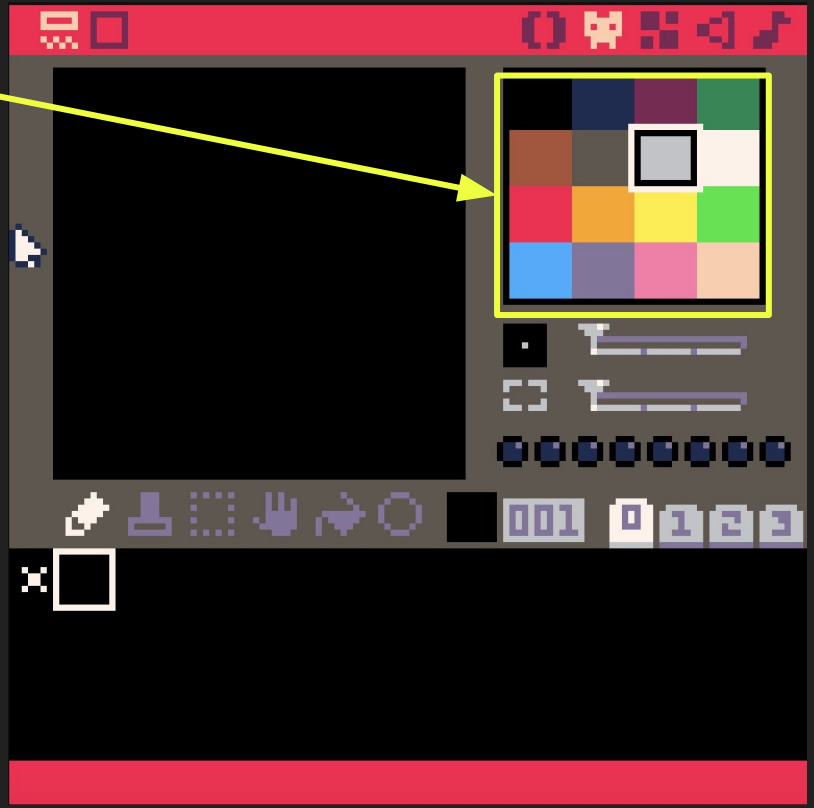
Drawing Sprites

- In the sprite editor, you have a few simple tools for drawing pixel art
- By default, each sprite is 8 x 8 pixels



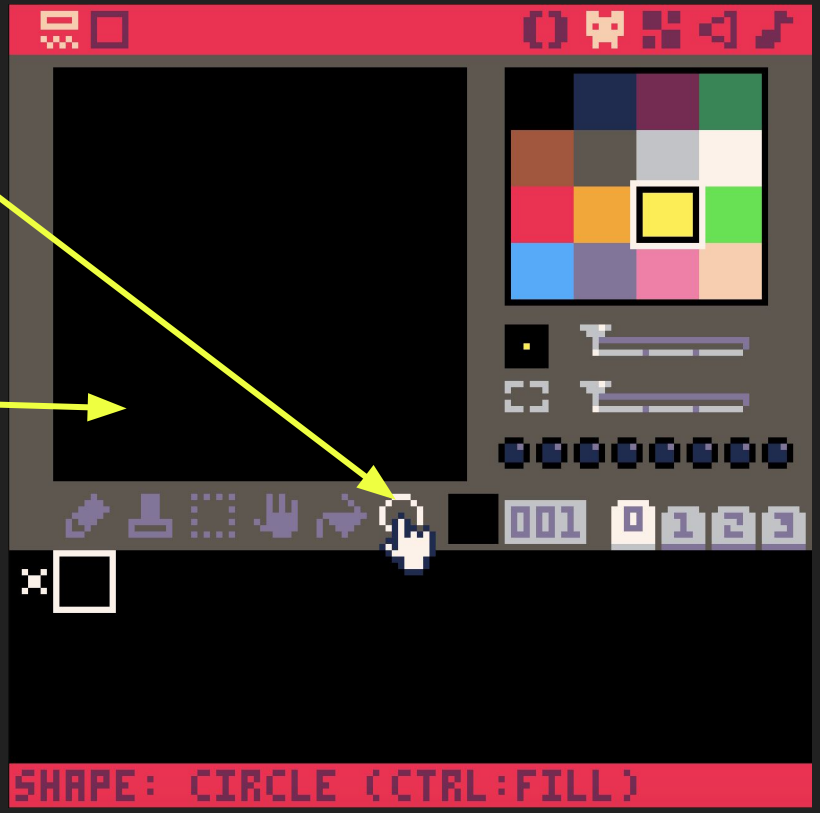
Drawing Sprites

- Select a color here



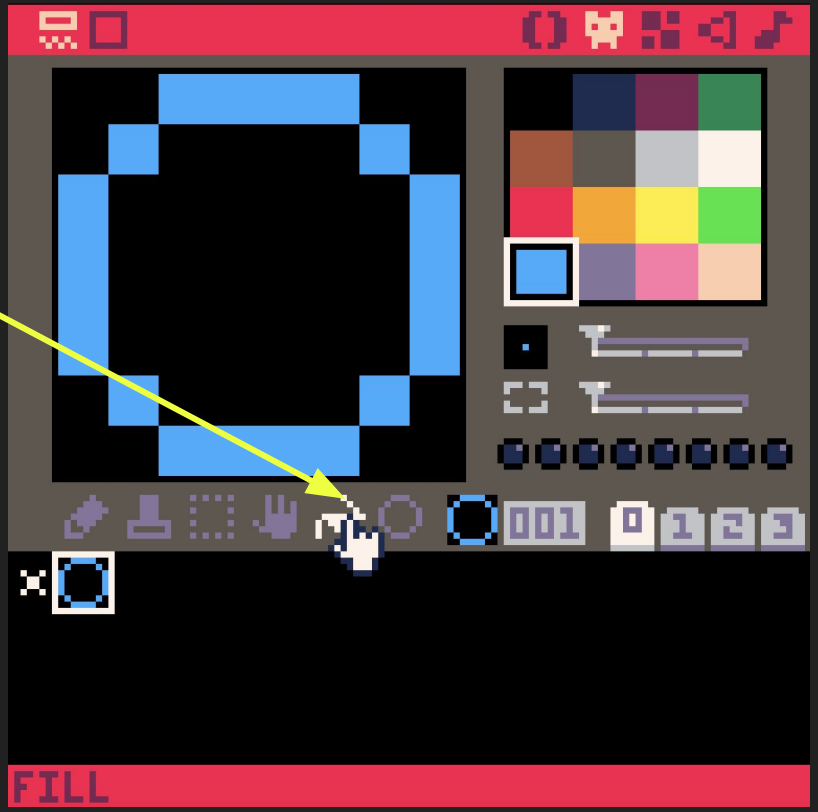
Drawing Sprites

- Click the Circle Tool to select it
- Then click and drag inside the canvas to draw a circle
- Hold CTRL to fill with the selected color



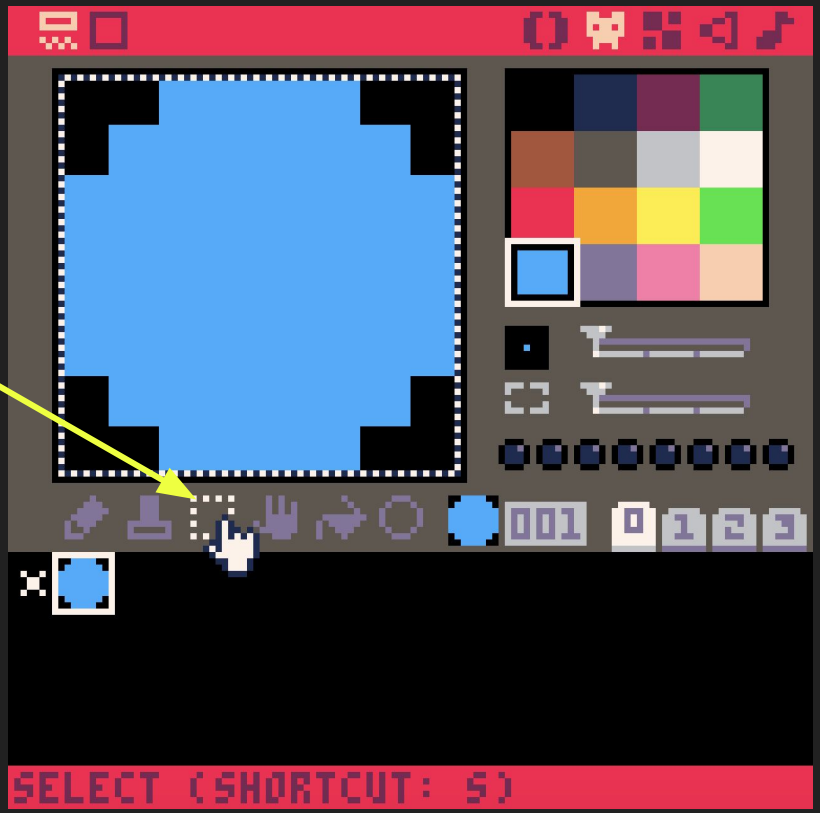
Drawing Sprites

- You can also fill an area using the Fill Tool



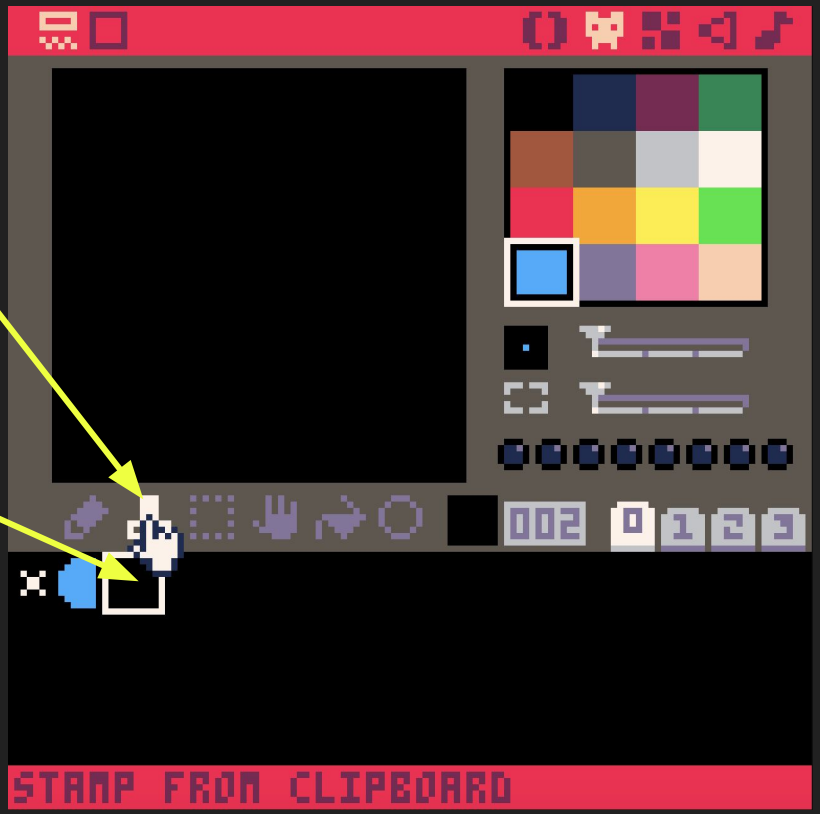
Drawing Sprites

- You can copy a sprite (or a portion of one) to your clipboard using the Select Tool and pressing CTRL C
- *Even on a Mac, the shortcut is CTRL, not CMD*



Drawing Sprites

- Use the Stamp Tool to paste what's on your clipboard (or use CTRL-V)
- Switch to a different sprite slot below the canvas to paste into a new sprite

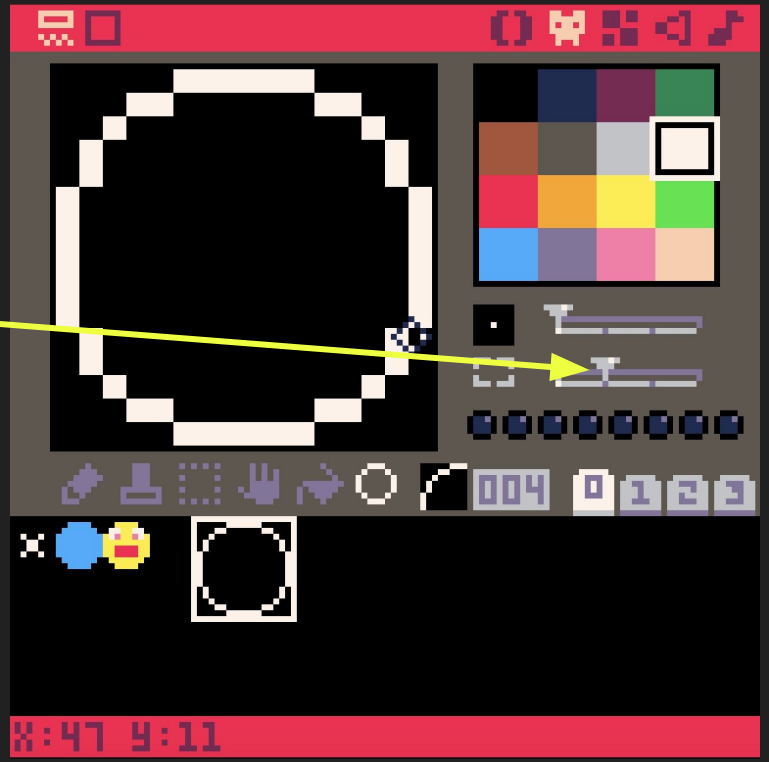


Drawing Sprites

- Use the Draw Tool to draw individual pixels with the selected color

Drawing Sprites

- Use this slider to change the size of your canvas
- This is a 16 x 16px circle
- You can also make 32 or 64 pixel sprites

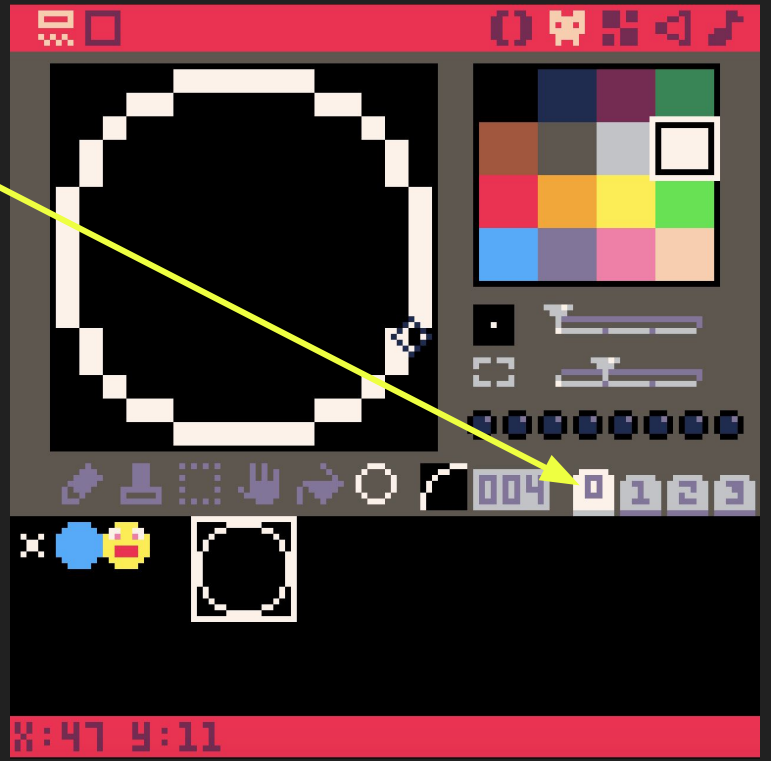


Drawing Sprites

- You have a limited amount of space in your sprite sheet, so larger sprites means fewer sprites

Drawing Sprites

- You have 4 tabs (labeled 0, 1, 2, 3) for drawing sprites, but it's *strongly recommended to only use Tabs 0 and 1*, because the sprite and map editors share some of the same memory



Drawing Sprites

- Not all games require a map, so in that case you could use all 4 tabs to draw sprites
- Just be aware that the map and sprite editors share some memory

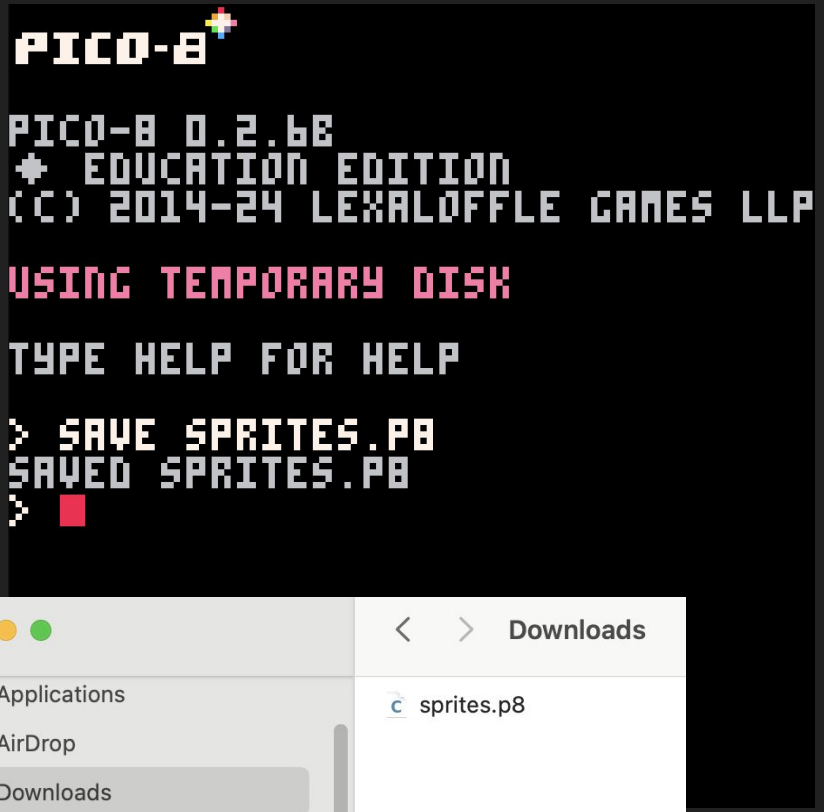
Saving Your Work

Saving Your Work

- Press CTRL S to save your work to the browser window
- But be aware that refreshing your window will erase the data saved in the browser

Saving Your Work

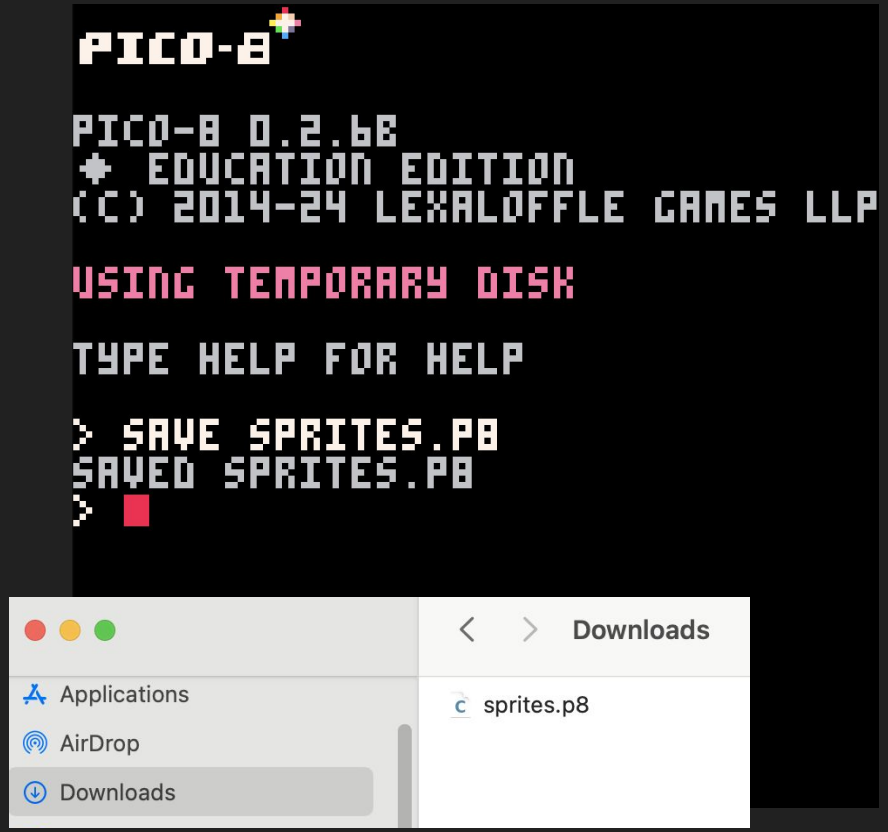
- Hit Esc and type the SAVE command to download your file
- You can load a file from your computer



I recommend saving or backing up your work for this course to Google Drive, so you can easily download the files anywhere

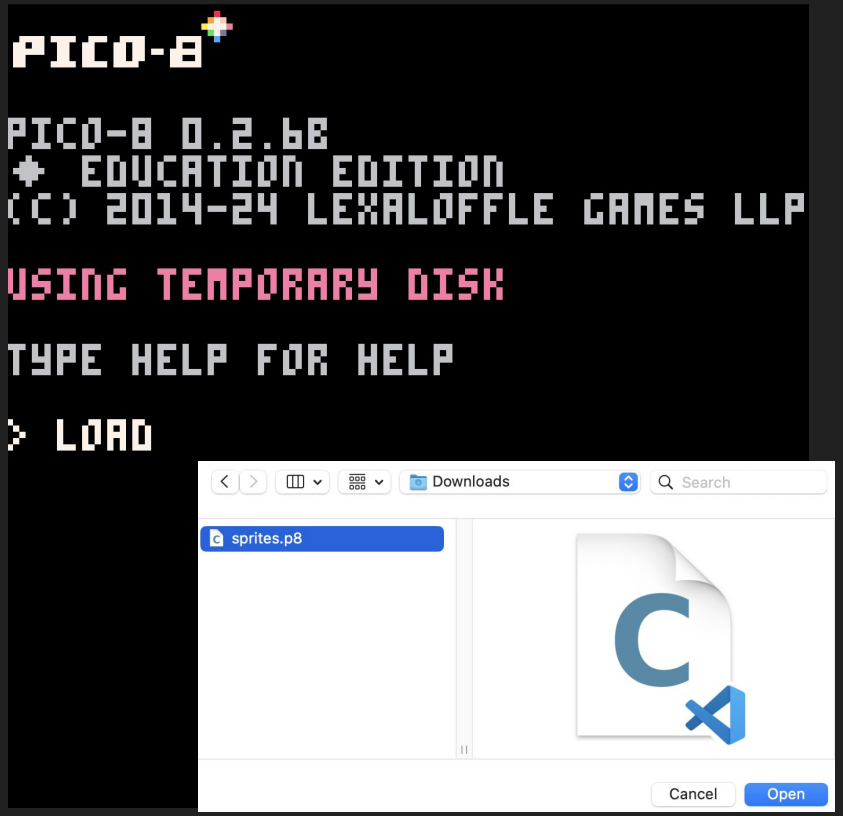
Saving Your Work

- You can provide a filename and extension after the command SAVE
- For example: SAVE MYGAME.p8
- *PICO-8 files use the .P8 extension*
- *You'll type in all caps by default*
- Press Enter/Return to perform the command



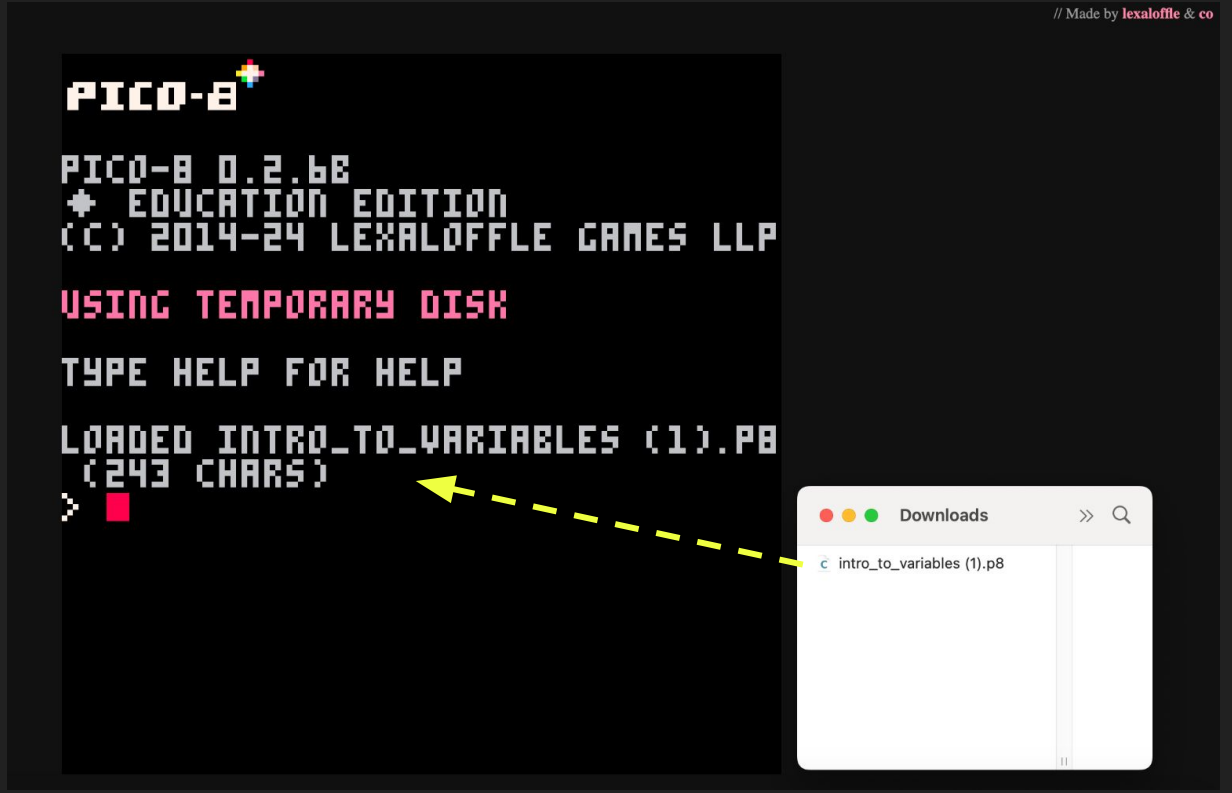
Loading a Saved P8 File

- Once you've saved, you can load a .P8 file using the LOAD command
- You'll be asked to browse for the file



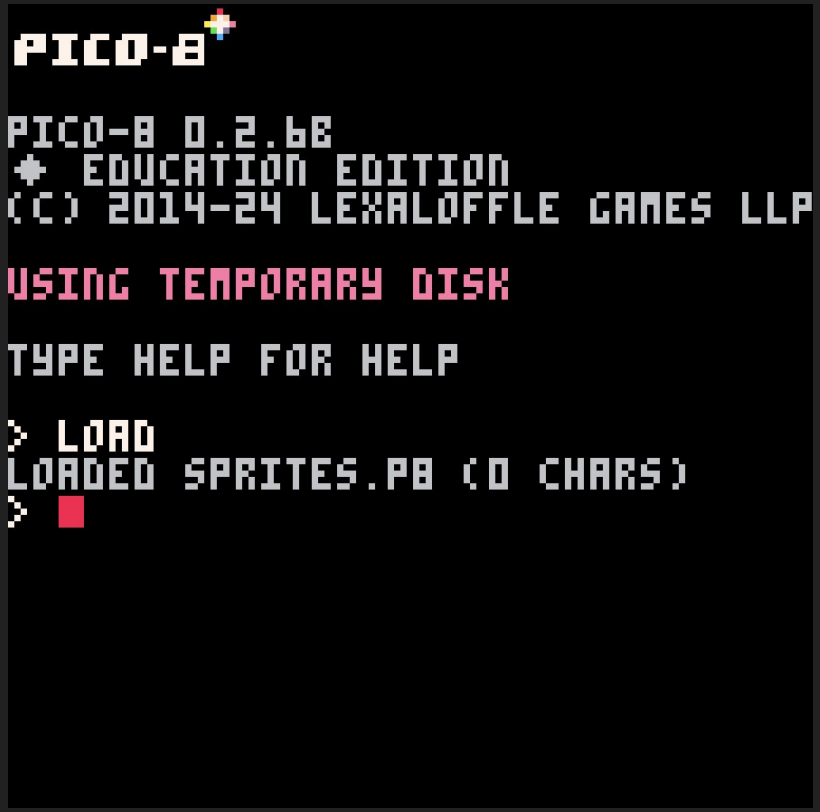
Loading a Saved P8 File

You can also drag the file into the PICO-8 window from your desktop



Loading a Saved P8 File

- When you see a message that the file was loaded, you can press Esc to return to the editor and resume your work



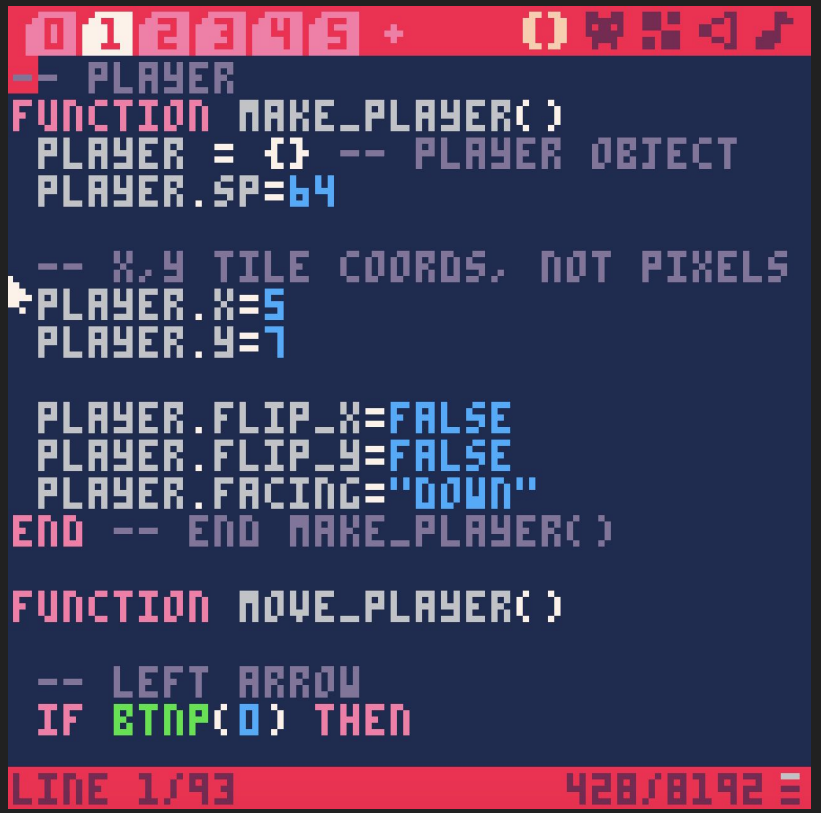
Loading a Saved P8 File

- Please note that *the desktop version of the software is a little different when it comes to loading saved files*
  - *https://www.reddit.com/r/pico8/comments/s0vy39/where_can_i_find_my_save_files/*

Setting up an External Code Editor

Coding in PICO-8

The font in the web editor is very stylized, so if you prefer an alternative code editor, I recommend Microsoft Visual Studio Code with this PICO-8 extension

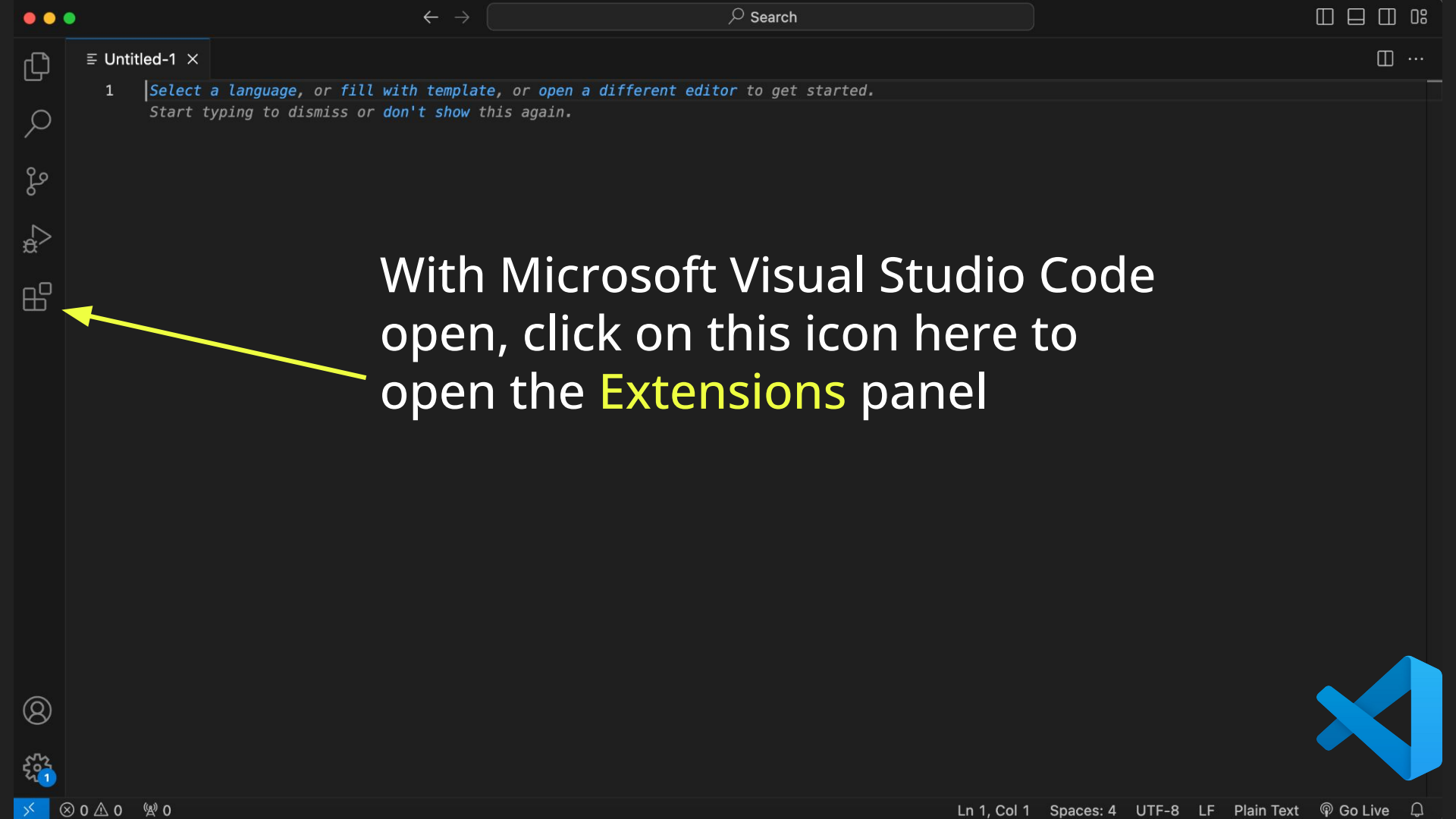


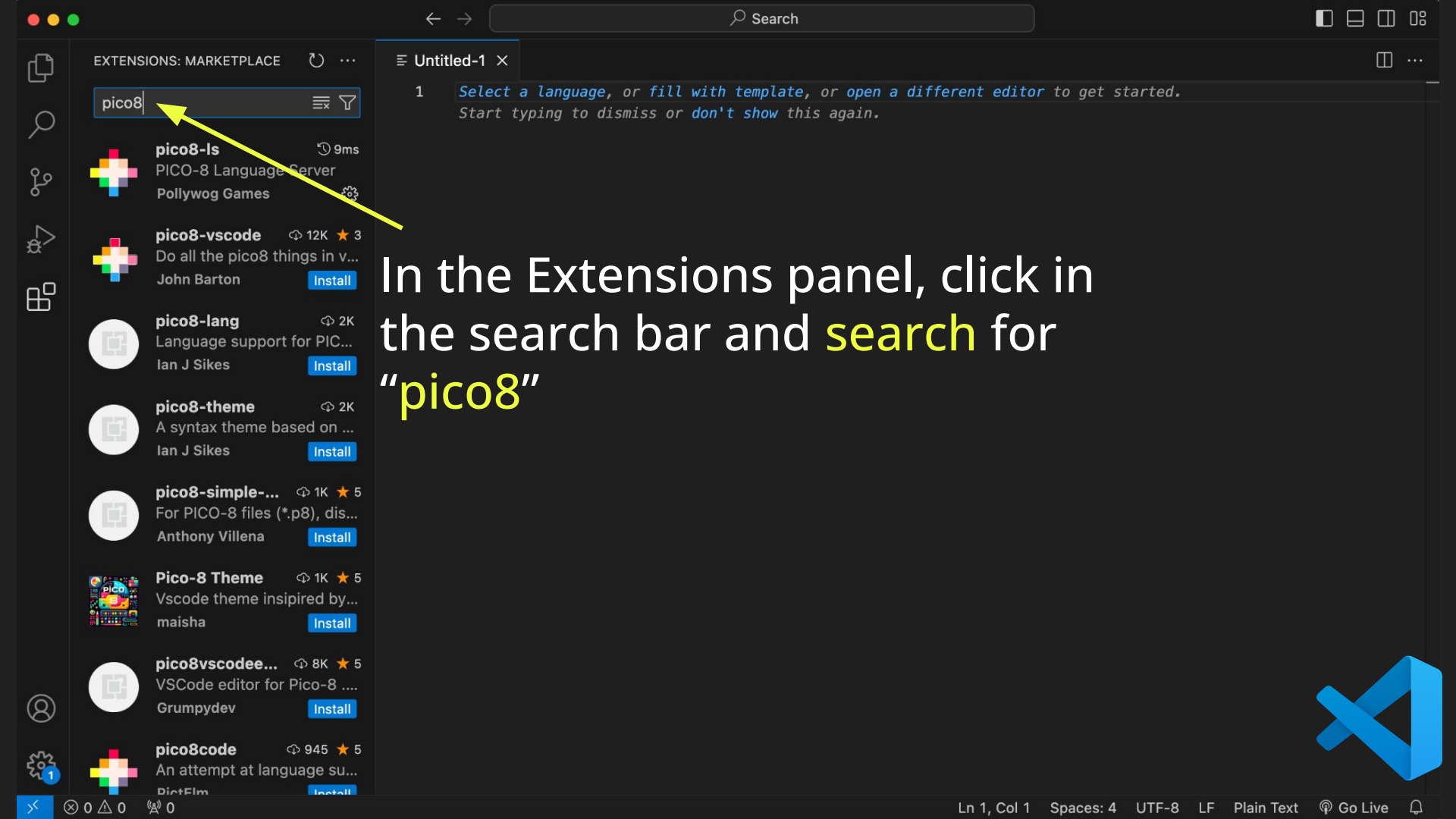
Using an External Code Editor

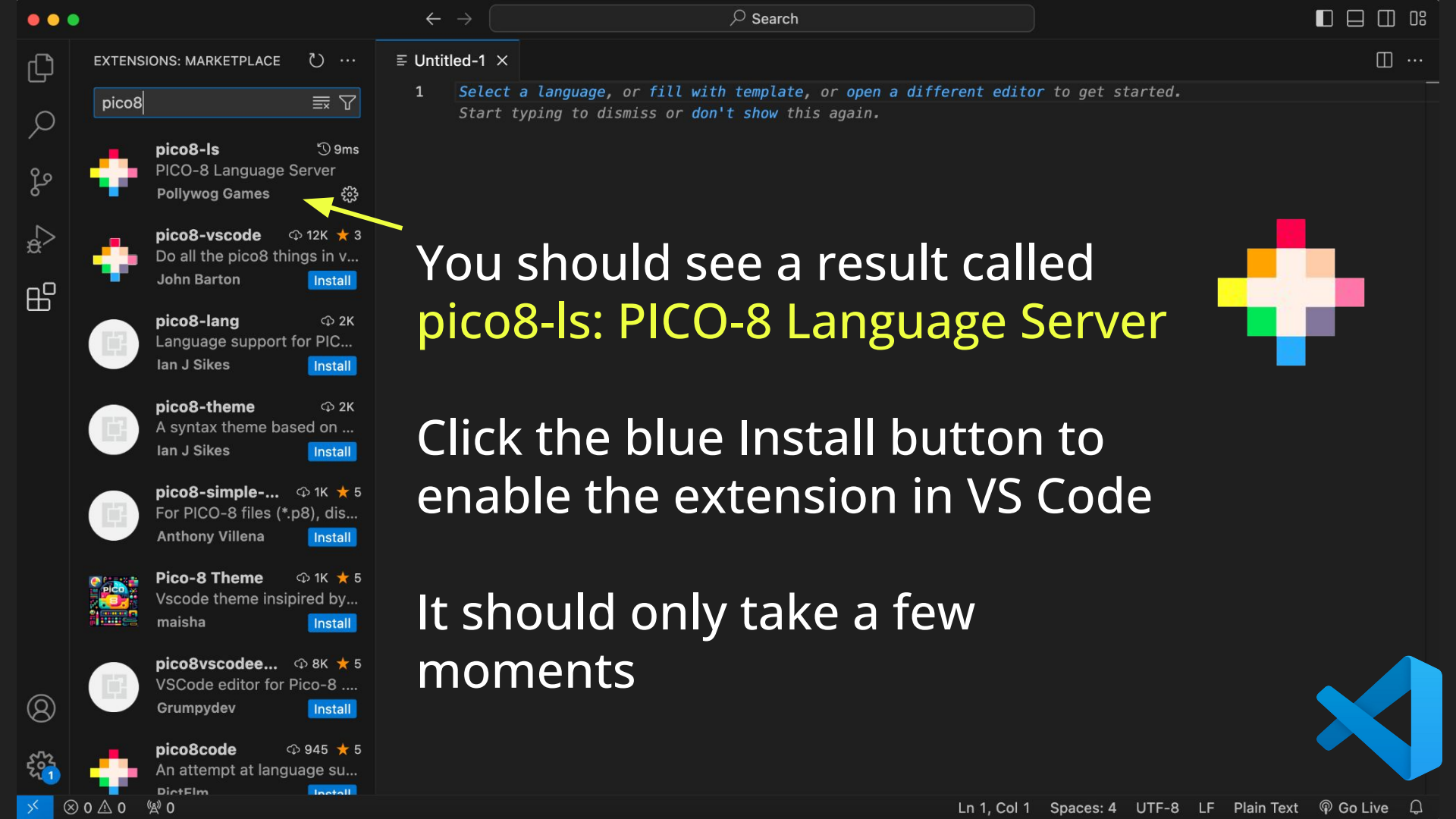
- Microsoft Visual Studio Code 
  - A professional-grade programming environment/code editor
  - Runs on Windows, MacOSX, and Linux
- **Is installed on the lab computers**
- Download on a personal computer

Using an External Code Editor

- Launch Microsoft Visual Studio Code
- Install the pico8-ls Extension in Visual Studio Code
  - Provides syntax highlighting and other Lua/PICO-8 support

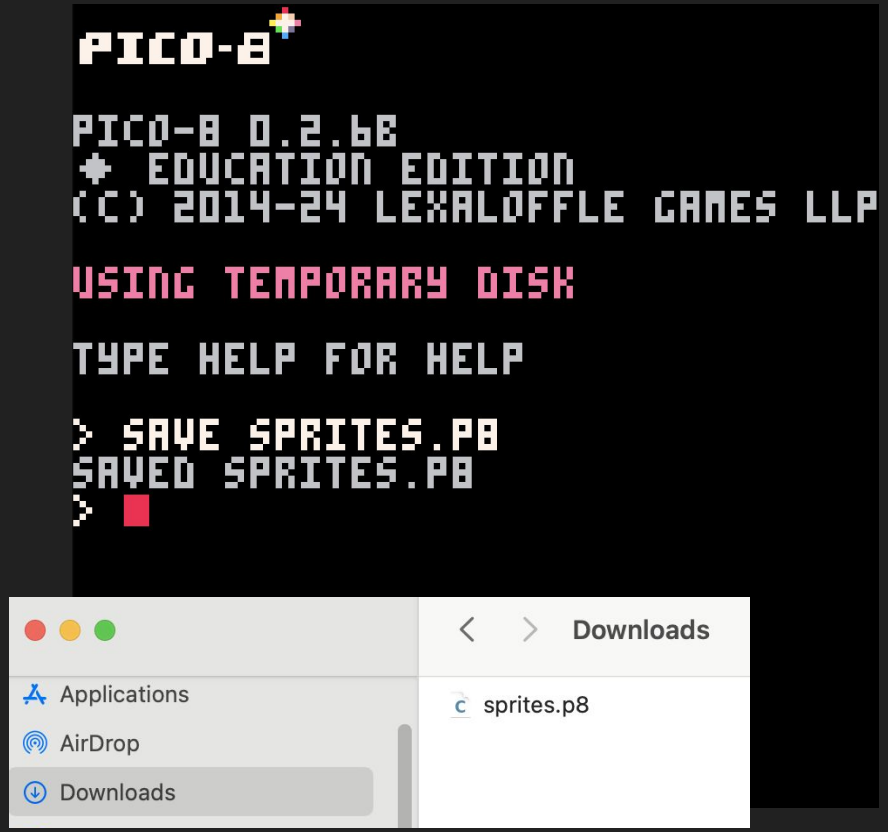

Search
Untitled-1
1
Select a language, or fill with template, or open a different editor to get started.
Start typing to dismiss or don't show this again.
With Microsoft Visual Studio Code open, click on this icon here to open the Extensions panel
Ln 1, Col 1
Spaces: 4
UTF-8
LF
Plain Text
Go Live

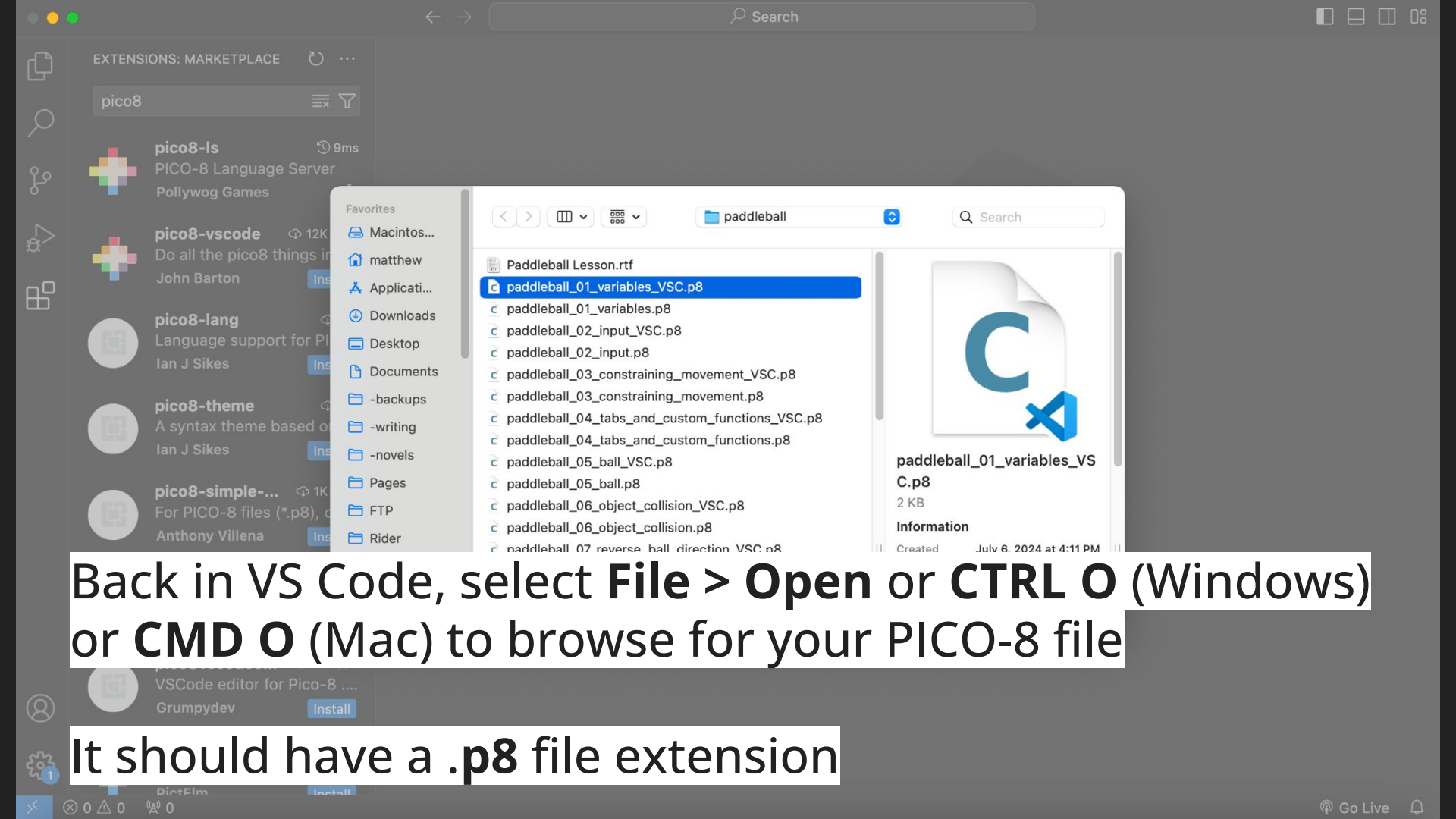

Search
EXTENSIONS: MARKETPLACE
pico8
pico8-ls 9ms
PICO-8 Language Server
Pollywog Games
pico8-vscode 12K ★ 3
Do all the pico8 things in v...
John Barton Install
pico8-lang 2K
Language support for PIC...
Ian J Sikes Install
pico8-theme 2K
A syntax theme based on ...
Ian J Sikes Install
pico8-simple-... 1K ★ 5
For PICO-8 files (*.p8), dis...
Anthony Villena Install
Pico-8 Theme 1K ★ 5
Vscode theme insipired by...
maisha Install
pico8vscodee... 8K ★ 5
VSCode editor for Pico-8 ....
Grumpydev Install
pico8code 945 ★ 5
An attempt at language su...
Untitled-1
1 Select a language, or fill with template, or open a different editor to get started.
Start typing to dismiss or don't show this again.
In the Extensions panel, click in the search bar and search for “pico8”
Ln 1, Col 1 Spaces: 4 UTF-8 LF Plain Text Go Live


Search
EXTENSIONS: MARKETPLACE
pico8
pico8-ls 9ms
PICO-8 Language Server
Pollywog Games
pico8-vscode 12K ★ 3
Do all the pico8 things in v...
John Barton
Install
pico8-lang 2K
Language support for PIC...
Ian J Sikes
Install
pico8-theme 2K
A syntax theme based on ...
Ian J Sikes
Install
pico8-simple-... 1K ★ 5
For PICO-8 files (*.p8), dis...
Anthony Villena
Install
Pico-8 Theme 1K ★ 5
Vscode theme insipired by...
maisha
Install
pico8vscodee... 8K ★ 5
VSCode editor for Pico-8 ....
Grumpydev
Install
pico8code 945 ★ 5
An attempt at language su...
Untitled-1
1 Select a language, or fill with template, or open a different editor to get started.
Start typing to dismiss or don't show this again.
You should see a result called
pico8-ls: PICO-8 Language Server
Click the blue Install button to
enable the extension in VS Code
It should only take a few
moments
Ln 1, Col 1 Spaces: 4 UTF-8 LF Plain Text Go Live

Save Your Work

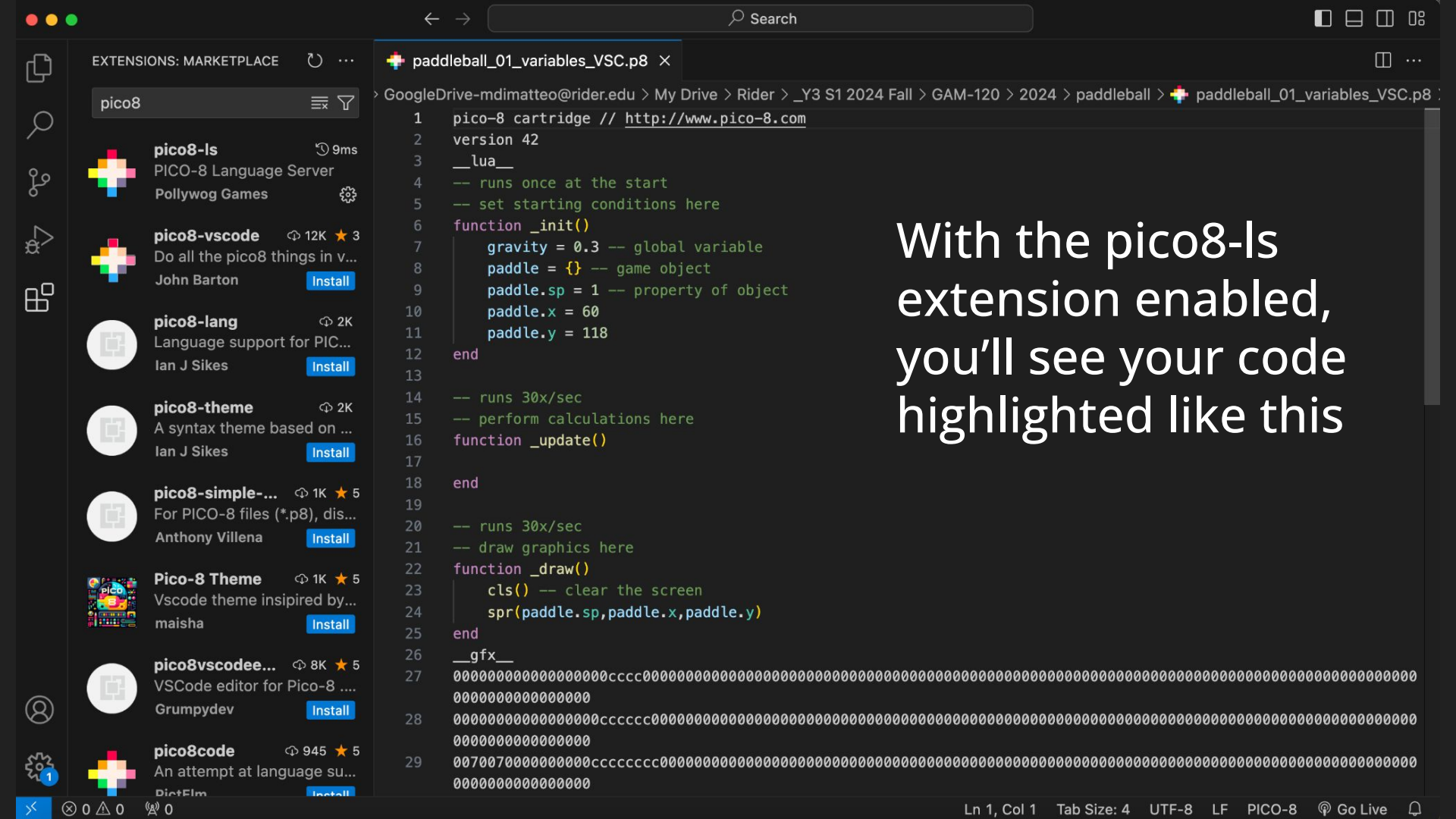
- In the PICO-8 editor, Hit Esc and type the SAVE command with a filename to download your file



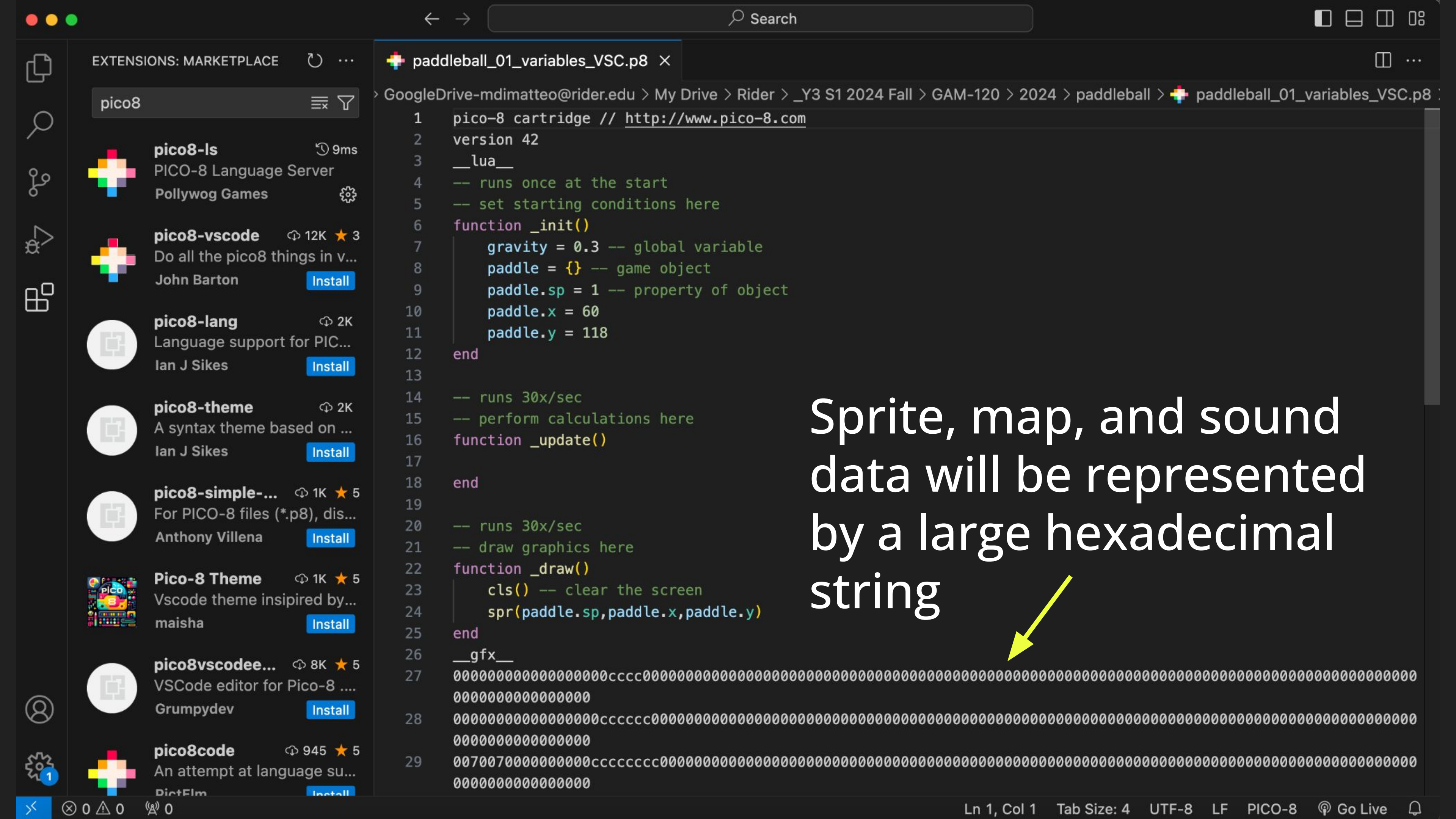


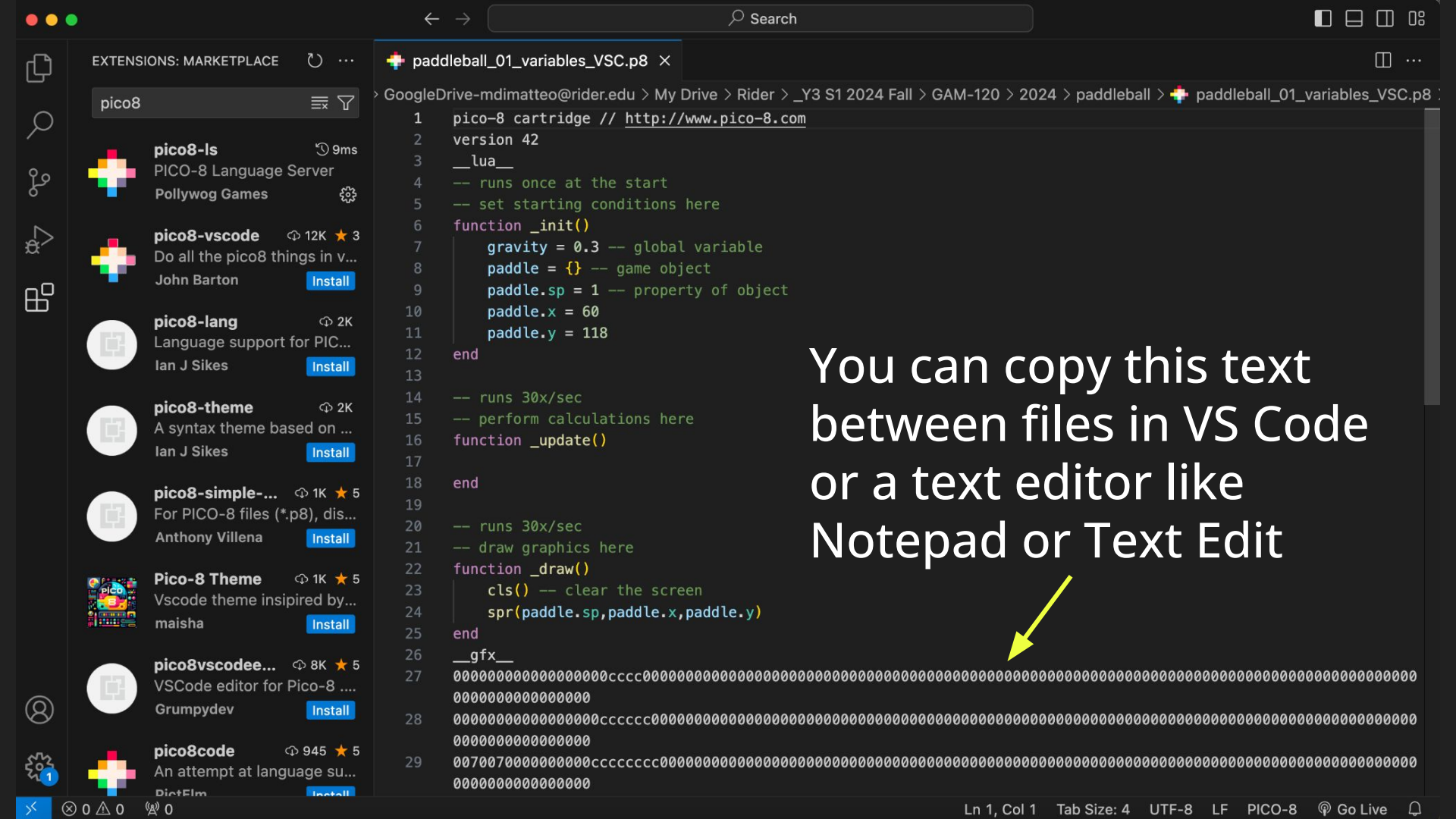
Back in VS Code, select **File > Open** or **CTRL O** (Windows) or **CMD O** (Mac) to browse for your PICO-8 file

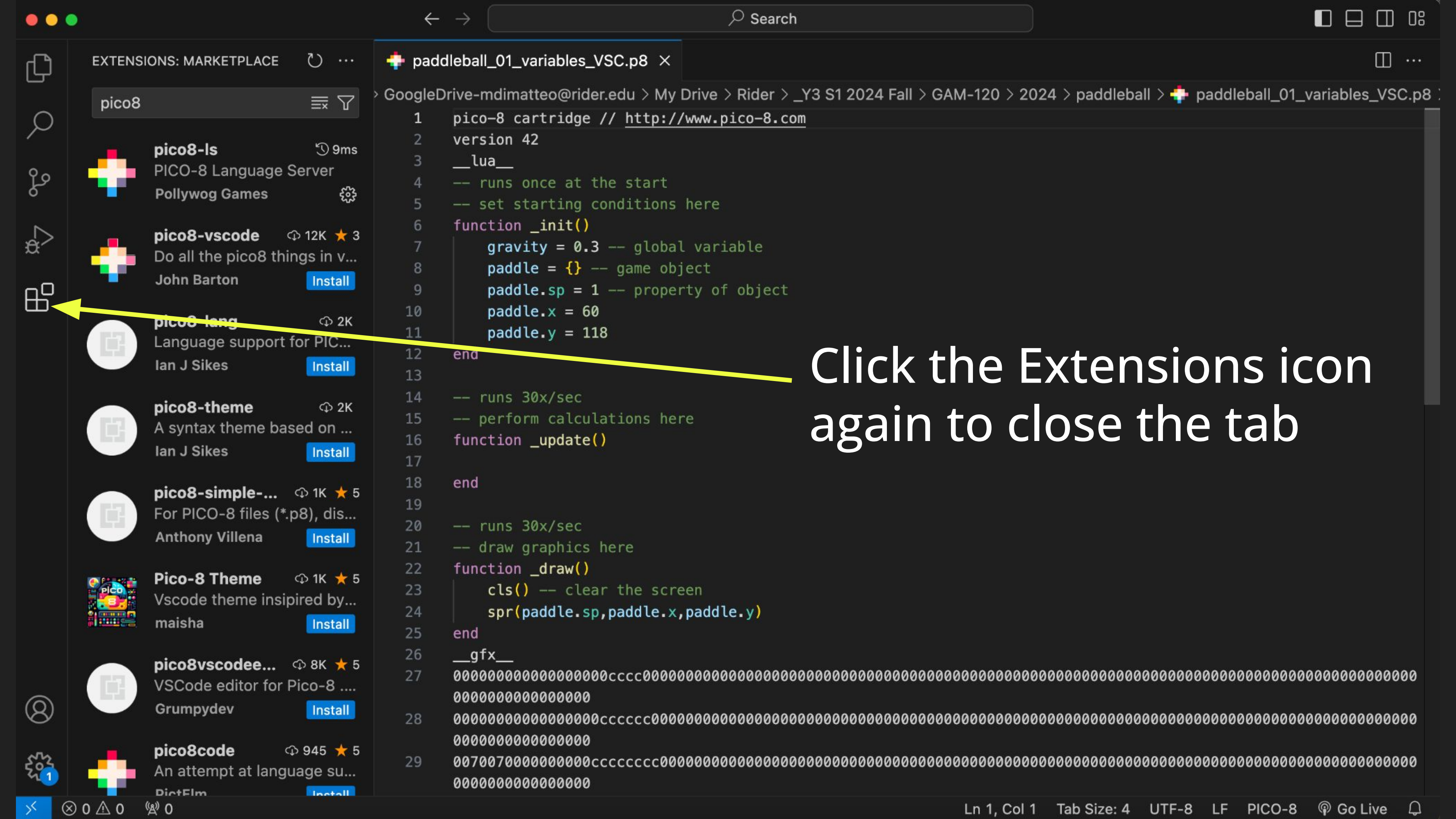
It should have a **.p8** file extension



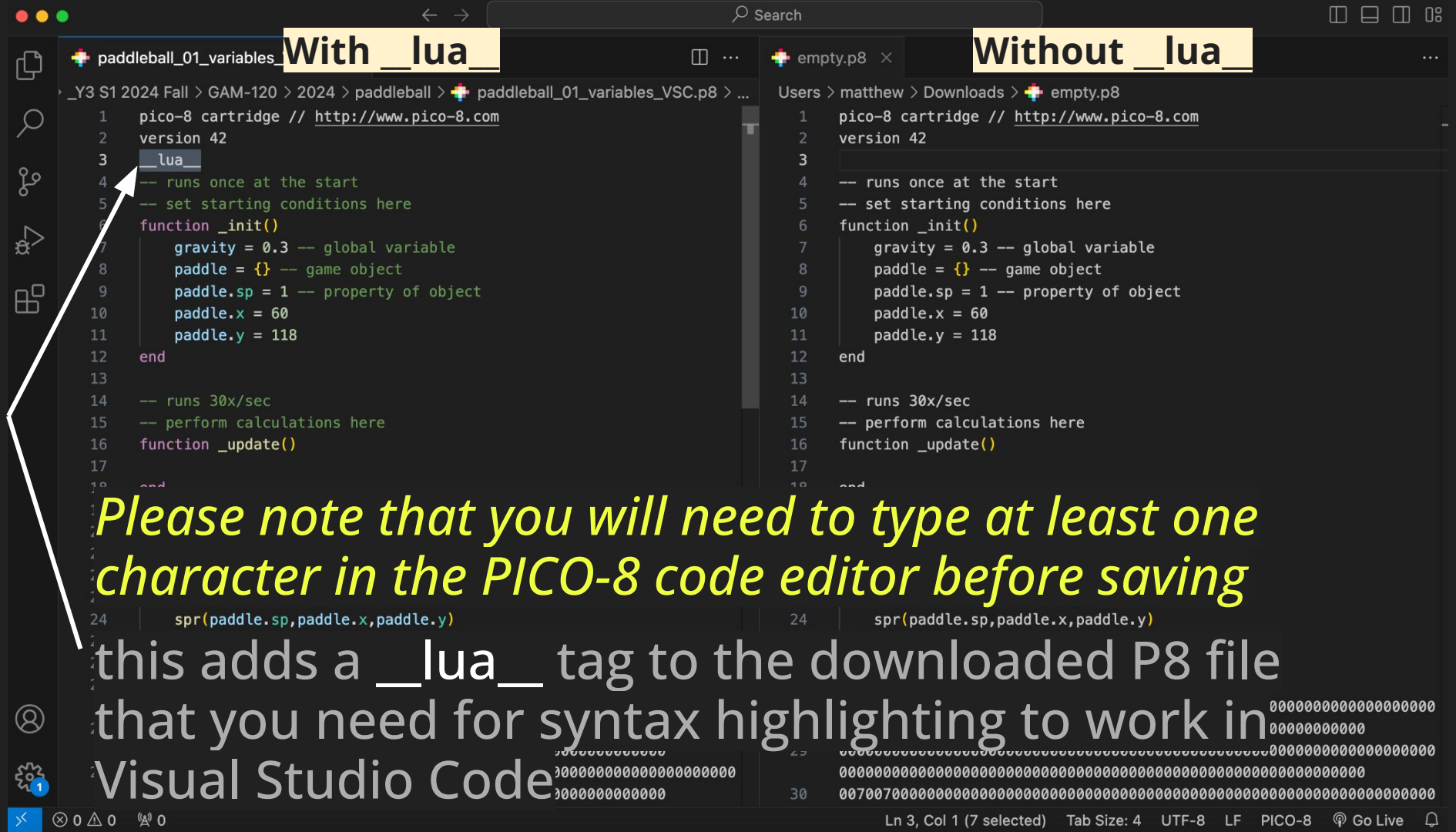
With the pico8-ls extension enabled, you'll see your code highlighted like this


EXTENSIONS: MARKETPLACE
pico8
pico8-ls
PICO-8 Language Server
Pollywog Games
pico8-vscode
Do all the pico8 things in v...
John Barton
Install
pico8-lang
Language support for PIC...
Ian J Sikes
Install
pico8-theme
A syntax theme based on ...
Ian J Sikes
Install
pico8-simple-...
For PICO-8 files (*.p8), dis...
Anthony Villena
Install
Pico-8 Theme
Vscode theme inspired by...
maisha
Install
pico8vscodee...
VSCode editor for Pico-8 ....
Grumpydev
Install
pico8code
An attempt at language su...
paddleball_01_variables_VSC.p8
GoogleDrive-mdimatteo@rider.edu > My Drive > Rider > _Y3 S1 2024 Fall > GAM-120 > 2024 > paddleball > paddleball_01_variables_VSC.p8
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
-- runs once at the start
-- set starting conditions here
function _init()
gravity = 0.3 -- global variable
paddle = {} -- game object
paddle.sp = 1 -- property of object
paddle.x = 60
paddle.y = 118
end
-- runs 30x/sec
-- perform calculations here
function _update()
end
-- runs 30x/sec
-- draw graphics here
function _draw()
cls() -- clear the screen
spr(paddle.sp,paddle.x,paddle.y)
end
__gfx__
Sprite, map, and sound data will be represented by a large hexadecimal string
Ln 1, Col 1
Tab Size: 4
UTF-8
LF
PICO-8
Go Live


Search
EXTENSIONS: MARKETPLACE
pico8
pico8-ls 9ms
PICO-8 Language Server
Pollywog Games
pico8-vscode 12K ★ 3
Do all the pico8 things in v...
John Barton Install
pico8-lang 2K
Language support for PIC...
Ian J Sikes Install
pico8-theme 2K
A syntax theme based on ...
Ian J Sikes Install
pico8-simple-... 1K ★ 5
For PICO-8 files (*.p8), dis...
Anthony Villena Install
Pico-8 Theme 1K ★ 5
Vscode theme insipired by...
maisha Install
pico8vscodee... 8K ★ 5
VSCode editor for Pico-8 ....
Grumpydev Install
pico8code 945 ★ 5
An attempt at language su...
paddleball_01_variables_VSC.p8
GoogleDrive-mdimatteo@rider.edu > My Drive > Rider > _Y3 S1 2024 Fall > GAM-120 > 2024 > paddleball > paddleball_01_variables_VSC.p8
1 pico-8 cartridge // http://www.pico-8.com
2 version 42
3 __lua__
4 -- runs once at the start
5 -- set starting conditions here
6 function _init()
7 gravity = 0.3 -- global variable
8 paddle = {} -- game object
9 paddle.sp = 1 -- property of object
10 paddle.x = 60
11 paddle.y = 118
12 end
13
14 -- runs 30x/sec
15 -- perform calculations here
16 function _update()
17
18 end
19
20 -- runs 30x/sec
21 -- draw graphics here
22 function _draw()
23 cls() -- clear the screen
24 spr(paddle.sp,paddle.x,paddle.y)
25 end
26 __gfx__
27 0000000000000000cccc0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
28 0000000000000000cccccc0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
29 0070070000000000cccccccc0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
You can copy this text between files in VS Code or a text editor like Notepad or Text Edit
Ln 1, Col 1 Tab Size: 4 UTF-8 LF PICO-8 Go Live



Click the Extensions icon again to close the tab

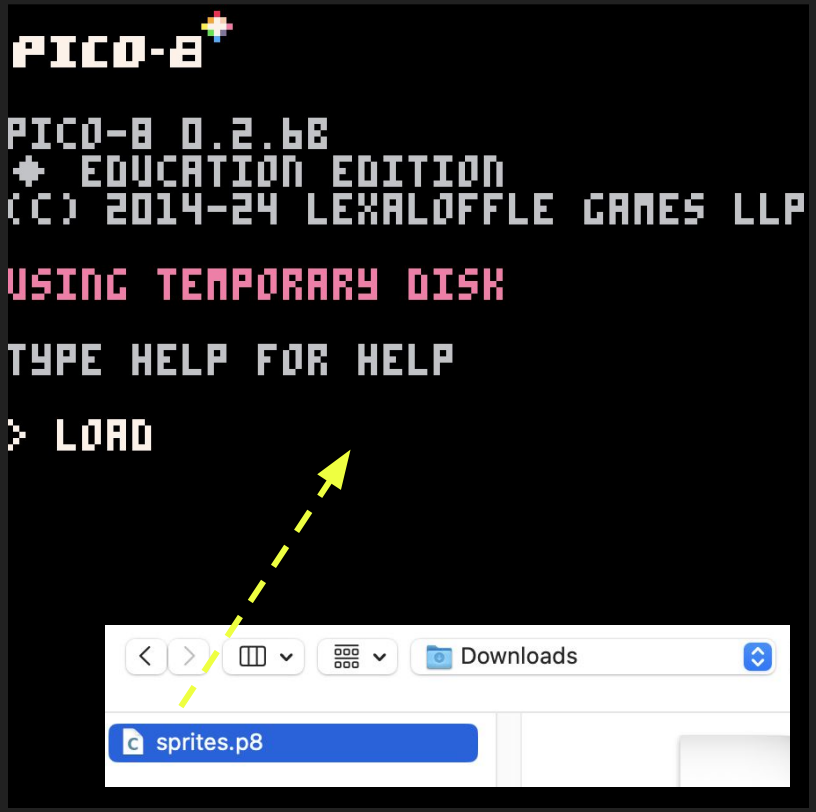

With __lua__
Without __lua__
paddleball_01_variables_
empty.p8
_Y3 S1 2024 Fall > GAM-120 > 2024 > paddleball > paddleball_01_variables_VSC.p8 > ...
Users > matthew > Downloads > empty.p8
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
-- runs once at the start
-- set starting conditions here
function _init()
gravity = 0.3 -- global variable
paddle = {} -- game object
paddle.sp = 1 -- property of object
paddle.x = 60
paddle.y = 118
end
-- runs 30x/sec
-- perform calculations here
function _update()
spr(paddle.sp,paddle.x,paddle.y)
Please note that you will need to type at least one character in the PICO-8 code editor before saving
this adds a __lua__ tag to the downloaded P8 file that you need for syntax highlighting to work in Visual Studio Code
Ln 3, Col 1 (7 selected)
Tab Size: 4
UTF-8
LF
PICO-8
Go Live

You will need to install the pico8-ls extension whenever using a different lab computer or signing in with another account

If you return to the same computer you were using and log in with the same account, the extension should still be enabled

Loading a Saved P8 File

- You can load your game in PICO-8 by refreshing your window and using the LOAD command, or dragging the file in from your desktop



Coding in PICO-8

Coding in PICO-8

- In the code editor, we can type SPR() to display a sprite in our game
- SPR() is a "*function*"
- A *function* is a programming concept for a set of instructions for the computer to execute

Coding in PICO-8

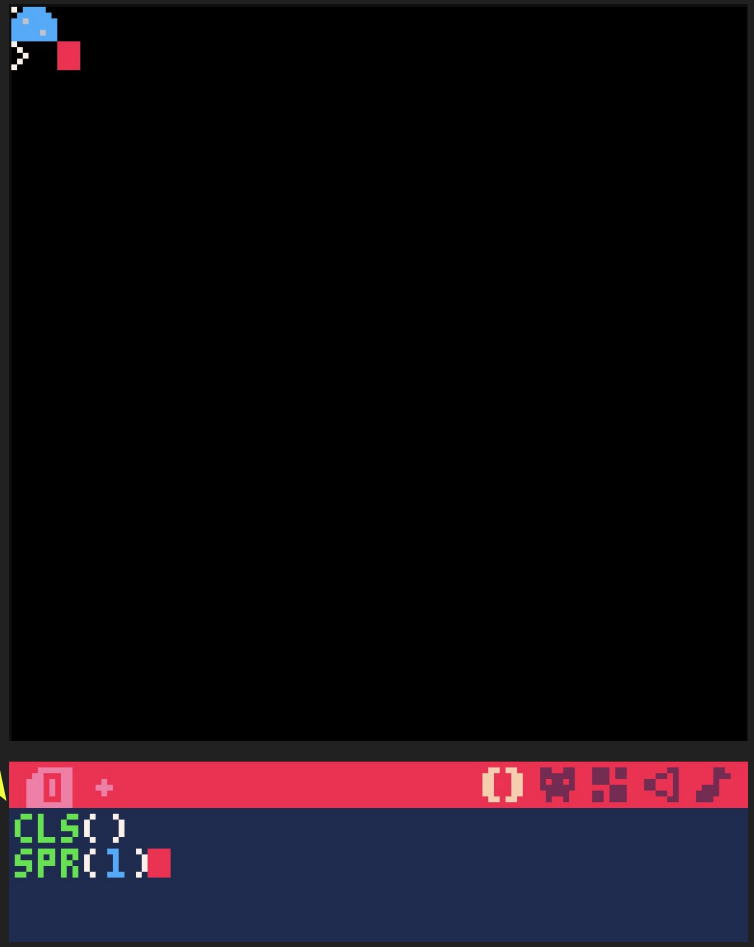
- We need to tell the computer *which* sprite to draw
- Each sprite in our sprite sheet has a number associated with it
- We can type the number 1 inside the parentheses – SPR(1) to tell the program to draw sprite number 1

Coding in PICO-8

- Press CTRL R or hit Esc and use the RUN command to run the game
- Our sprite is drawn at the top left of the screen by default
- Also, it’s drawn on top of our command line

Coding in PICO-8

- We can get rid of the command line by typing CLS() before we type SPR(1)
- CLS() is a function that clears the screen

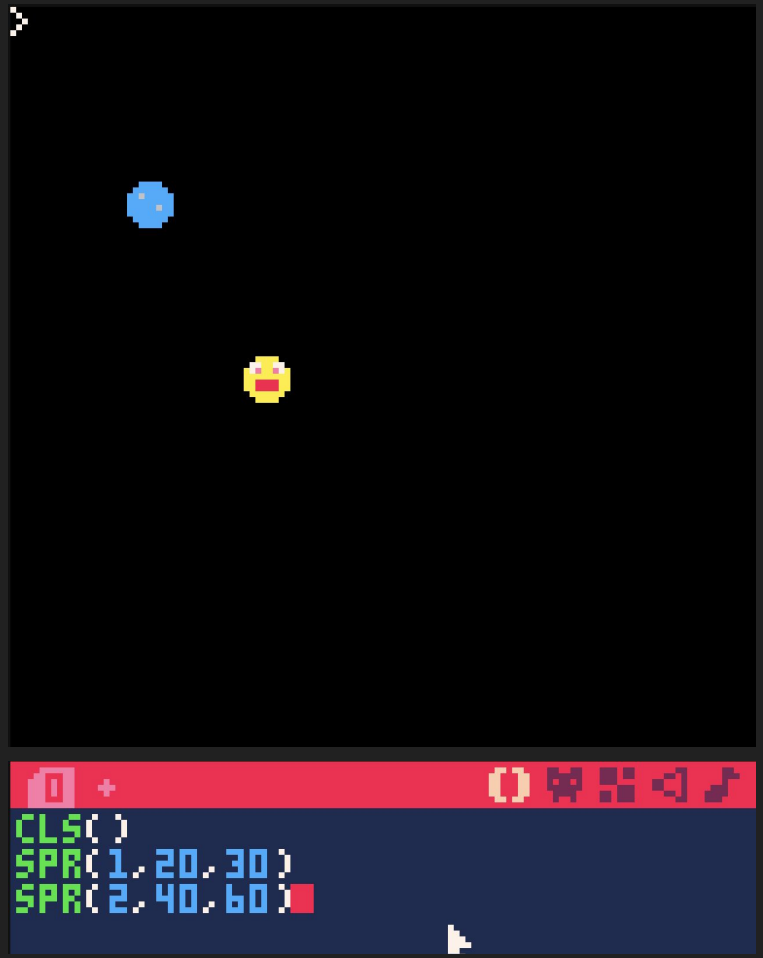


Coding in PICO-8

- Notice what happens if I try to draw sprite 2 after sprite 1
- *Sprite 2 covers up sprite 1 because both are being drawn in the same location*

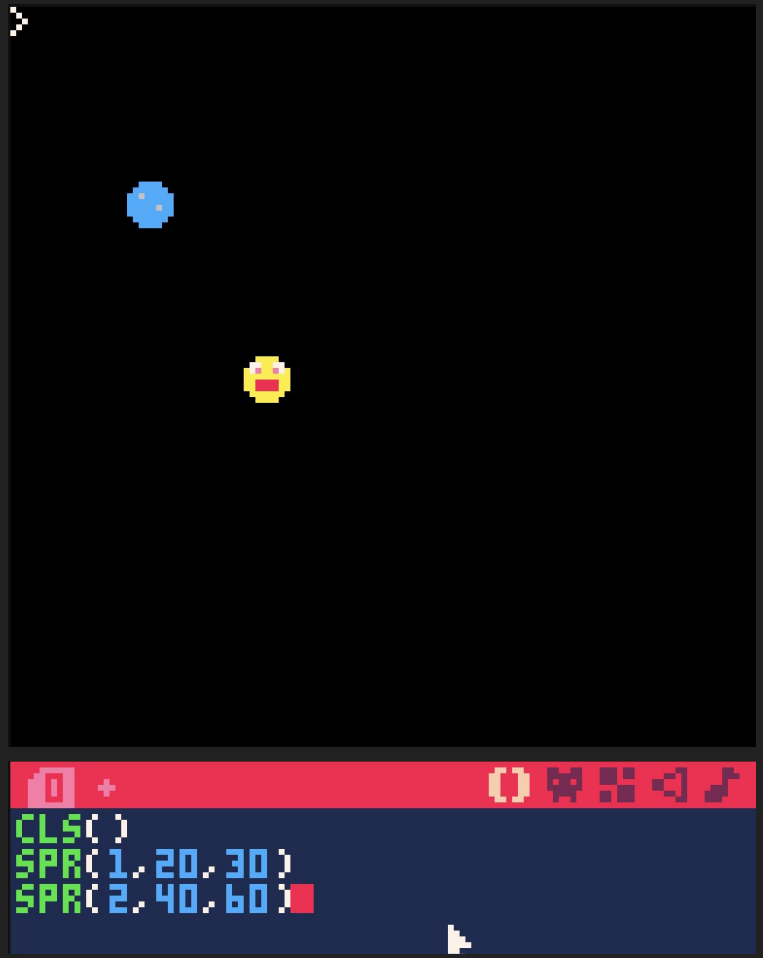
Coding in PICO-8

- We can add other values to the SPR() function to tell the program *where* to draw each sprite on the screen



Coding in PICO-8

- The SPR() function can be given values inside the parentheses for custom behavior
- These values must be given in the proper order



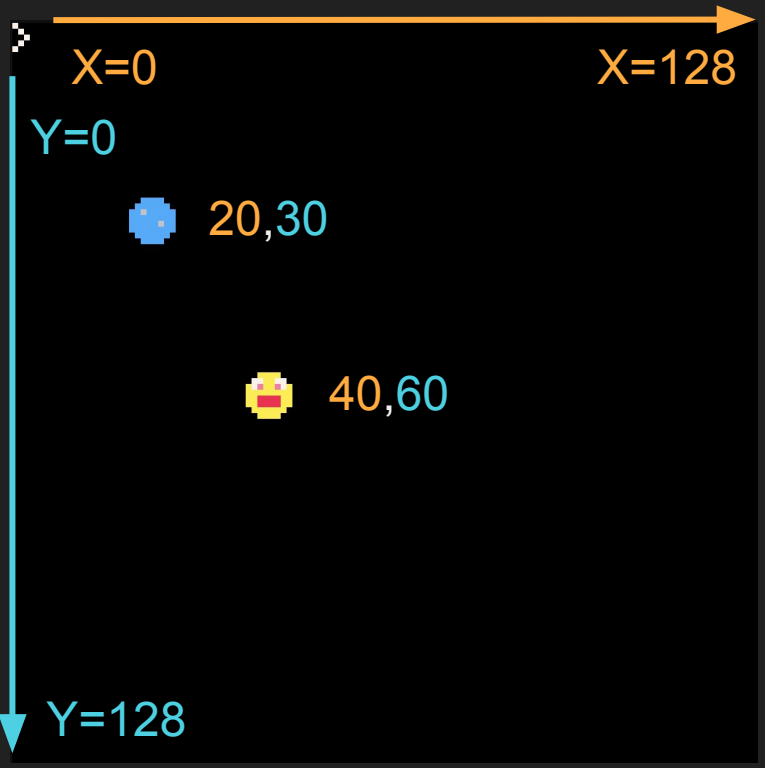
Coding in PICO-8

- 1st value: sprite #
- 2nd value: x coordinate
- 3rd value: y coordinate

There are other values we can provide, but these 3 will suffice for now

Coding in PICO-8

- The X,Y coordinate plane is drawn from left to right and top to bottom




```
SPR(1,20,30)
SPR(2,40,60)
```

PICO-8 Wiki

EXPLORE POPULAR PAGES COMMUNITY

in: Reference, API

Spr

EDIT

```
spr( n, [x,] [y,] [w,] [h,] [flip_x,] [flip_y] )
```

Draws a sprite, or a range of sprites, on the screen.

`n`

The sprite number. When drawing a range of sprites, this is the upper-left corner.

`x`

The x coordinate (pixels). The default is 0.

`y`

The y coordinate (pixels). The default is 0.

You can look up any PICO-8 function on the PICO-8 Wiki, and it will show you the exact order of the values that function can use

PICO-8 Wiki

EXPLORE POPULAR PAGES COMMUNITY

in: Reference, API

Spr

EDIT

```
spr( n, [x,] [y,] [w,] [h,] [flip_x,] [flip_y] )
```

Draws a sprite, or a range of sprites, on the screen.

`n`

The sprite number. When drawing a range of sprites, this is the upper-left corner.

`x`

The x coordinate (pixels). The default is 0.

`y`

The y coordinate (pixels). The default is 0.

Values with [square brackets] around them are optional values

PICO-8 Wiki

EXPLORE POPULAR PAGES COMMUNITY

in: Reference, API

Spr

EDIT

```
spr( n, [x,] [y,] [w,] [h,] [flip_x,] [flip_y] )
```

Draws a sprite, or a range of sprites, on the screen.

`n`

The sprite number. When drawing a range of sprites, this is the upper-left corner.

`x`

The x coordinate (pixels). The default is 0.

`y`

The y coordinate (pixels). The default is 0.

The wiki also explains what each value means and what its default value is

Coding in PICO-8

- Another tradition when learning a new programming language is to begin by printing the words HELLO WORLD on the screen

Coding in PICO-8

- We can use PICO-8's PRINT() function to do this
- Inside the parentheses, type "HELLO WORLD" inside quotes

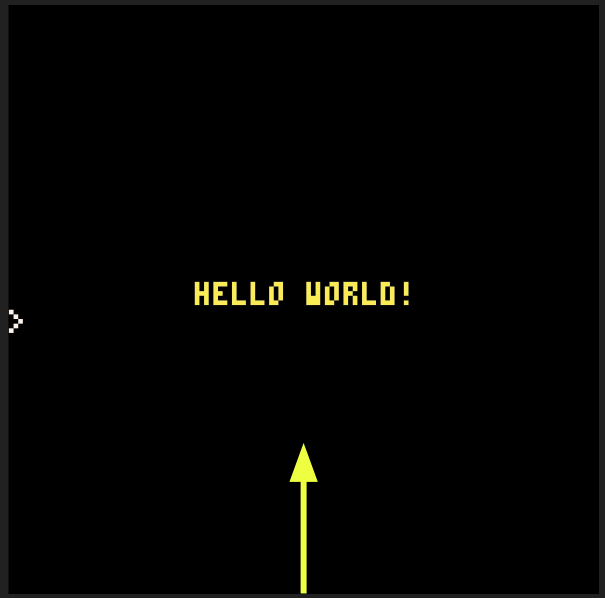
PRINT() is also useful for displaying a game HUD

Coding in PICO-8

- We can add values after the quotes, separated by commas, to set the position of the text
- Here, 40 is the X coordinate and 60 is the Y coordinate

Coding in PICO-8

- We can even add another value to set the color of the text
- Here, 10 corresponds with the color yellow




```
CLS()
PRINT("HELLO WORLD!",40,60,10)
```

Coding in PICO-8

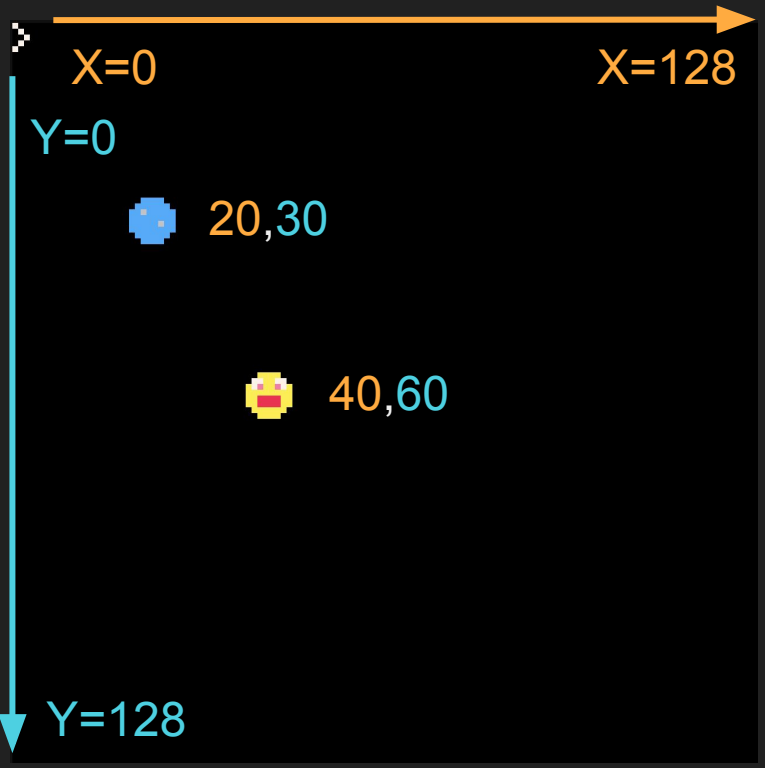
- Each of the 16 available colors in PICO-8 corresponds with a number
- Refer to the PICO-8 Cheat Sheet


```
CLS()
PRINT("HELLO WORLD!",40,60,10)
```

The PICO-8 Coordinate Plane

Coordinate Plane

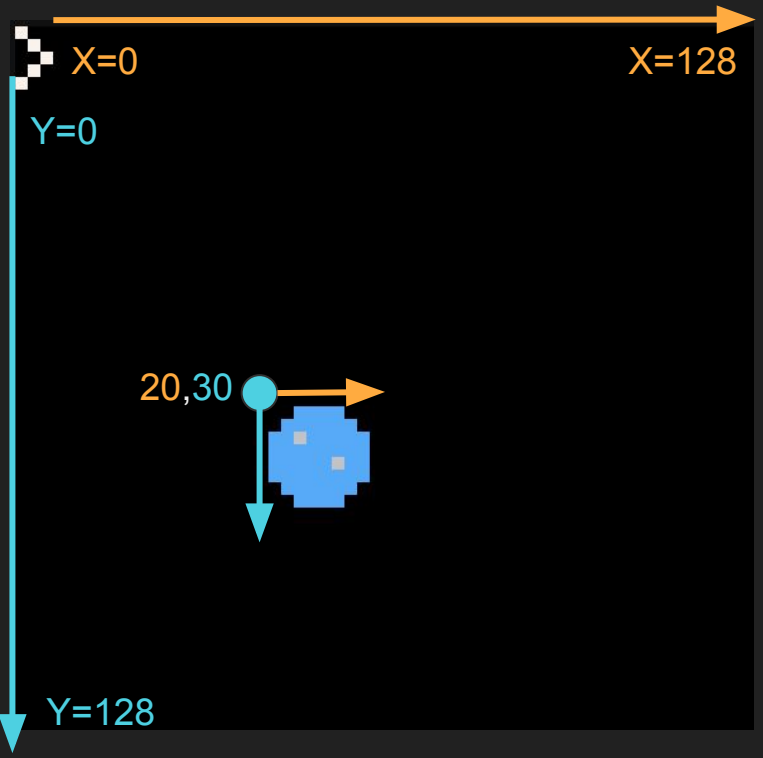
- The X,Y coordinate plane is drawn from left to right and top to bottom
- ***This means higher Y values are lower on the screen***




```
SPR(1,20,30)
SPR(2,40,60)
```

Coordinate Plane

- PICO-8 ***sprites are drawn from the top-left corner***
- Meaning the X,Y coordinates you give a sprite coincide with its top-left corner




```
SPR(1,20,30)
```

Coordinate Plane

- Because sprites are drawn from the top left, *setting the x value to half of the screen width (128/2=64) results in the sprite being slightly off-center* to the right


```
CLS()
SPR(1,64,2)
```

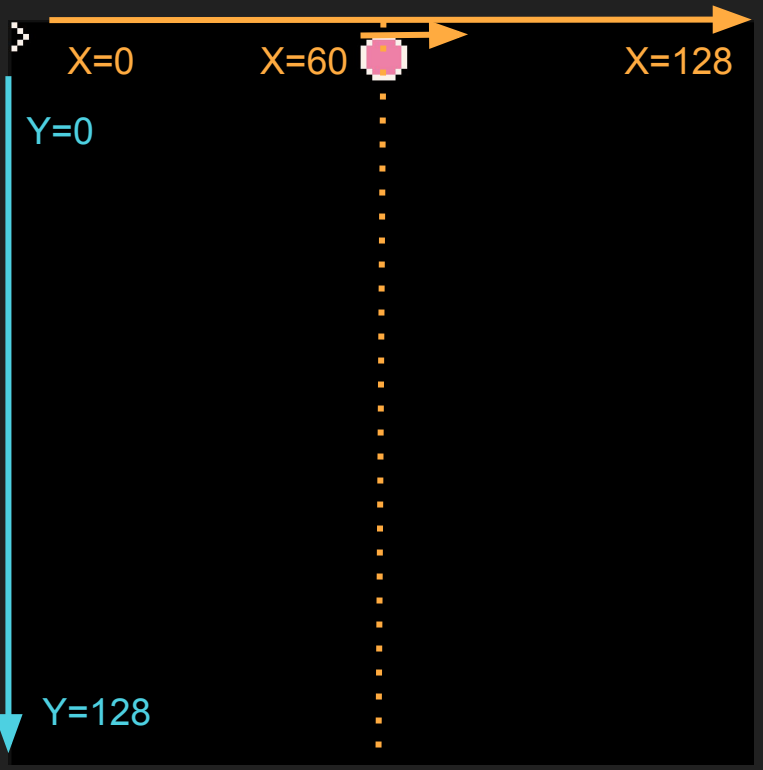
Coordinate Plane

- When trying to center a sprite, you must account for half its width or height because of the top-left origin


```
CLS()
SPR(1,64,2)
```

Coordinate Plane

- When trying to center a sprite, you must account for half its width or height because of the top-left origin
- 64 - (8/2) = 64 - 4 = 60




```
CLS()
SPR(1,60,2)
```

Next, we'll want to get these sprites moving around the screen and responding to player input

Before we can move our sprites in the game, we need to understand the way PICO-8 programs are structured

The PICO-8 Program Structure

Download Example File: basics_02_gameloop.p8

The PICO-8 Program Structure

- These three functions comprise the main structure of your PICO-8 program:
  - _init(), _update(), _draw()
- This is sometimes called the PICO-8 “game loop”

The PICO-8 Program Structure

- _init() — Runs once at the start
  Use for setting initial conditions
  Declaring variables
- _update() — Loops 30x per second
  Use for calculations, movement
- _draw() — Loops 30x per second
  Use to draw sprites, print text

The PICO-8 Program Structure

- We can set our initial variable values in _init()
- Then we can change those values in _update()
- And apply those values in _draw()

```
function _init()
 n=1
 x=60
 y=2
end
```

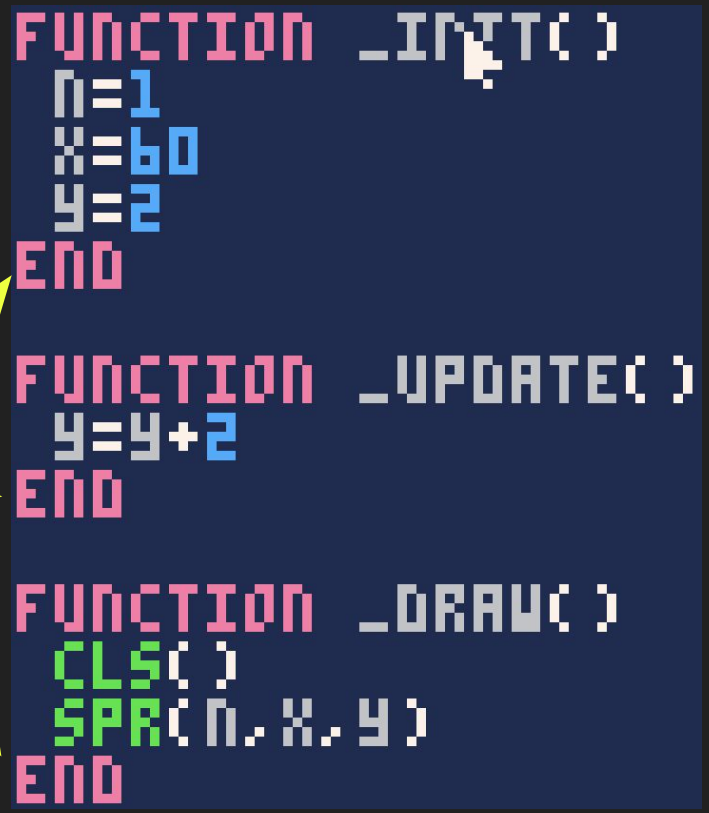
```
function _update()
 y=y+2
end
```

```
function _draw()
 cls()
 spr(n,x,y)
end
```

The PICO-8 Program Structure

- All functions must be “closed”
- You close a function using the keyword **END**

If you don’t close a function, it can result in an error or unexpected behavior



Using Variables

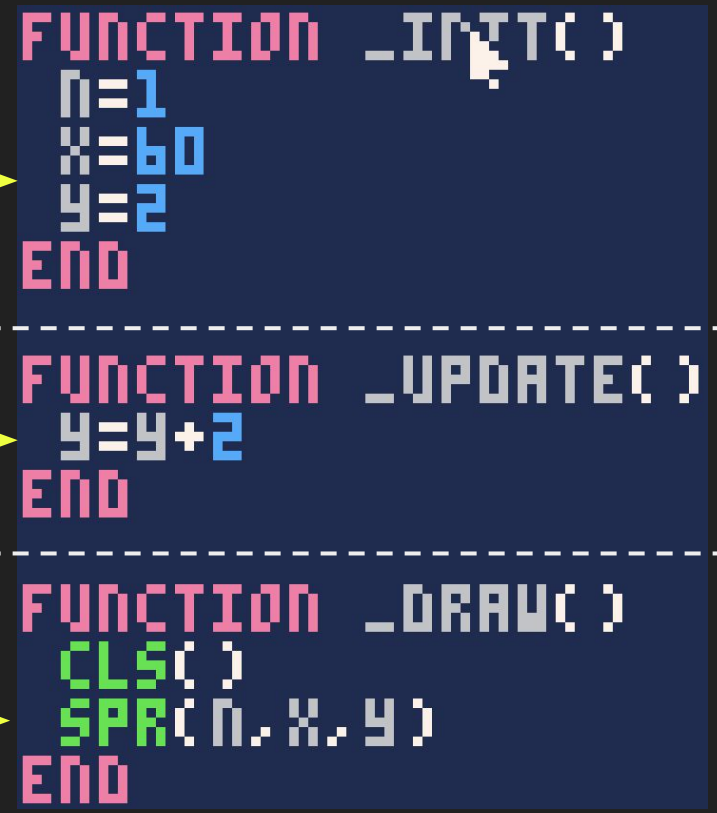
Download Example File: paddleball_basic_02_variables.p8

Using Variables

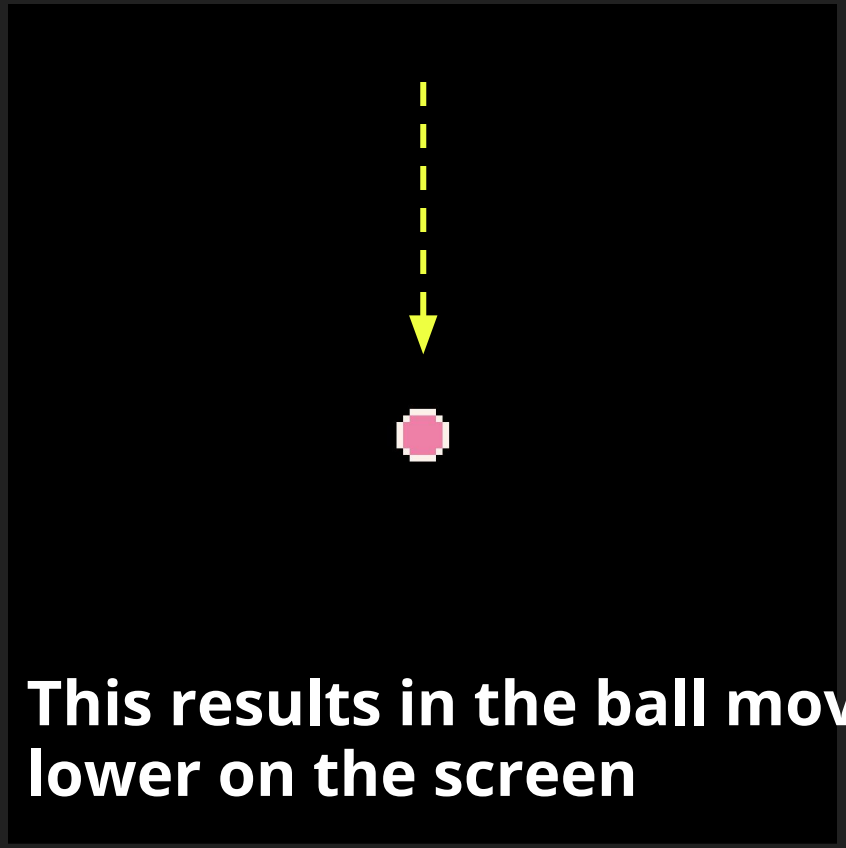
- *If we only use numbers to position our sprites, they won't be able to respond to player input and move*
- Using fixed numbers is called "hard-coding"

Using Variables

- We can set our initial variable values in _init()
- Then we can change those values in _update()
- And apply those values in _draw()



Using Variables



This results in the ball moving lower on the screen

```
function _init()
 n=1
 x=60
 y=2
end

function _update()
 y=y+2
end

function _draw()
 cls()
 spr(n,x,y)
end
```

Understanding the Game Loop

The PICO-8 Program Structure

- _init() only runs *once*, as soon as the game starts

- ***If these values need to change, that will be done in _update()***

The PICO-8 Program Structure

- _update and _draw() both run in a ***loop***, 30 times per second *(or once every 1/30th of a second)*

```
function _init()
 n=1
 x=60
 y=2
end

function _update()
 y=y+2
end

function _draw()
 cls()
 spr(n,x,y)
end
```

The PICO-8 Program Structure

Frame	y
0	2
1	4
2	6
3	8

At frame 0, y is what it's set at in _init(), or 2

The PICO-8 Program Structure

Frame	y
0	2
1	4
2	6
3	8

Because _update() runs continuously, the value of y increases each frame by the amount specified in the code (2)

```
function _init()
 n=1
 x=60
 y=2
end

function _update()
 y=y+2
end

function _draw()
 cls()
 spr(n,x,y)
end
```

The PICO-8 Program Structure

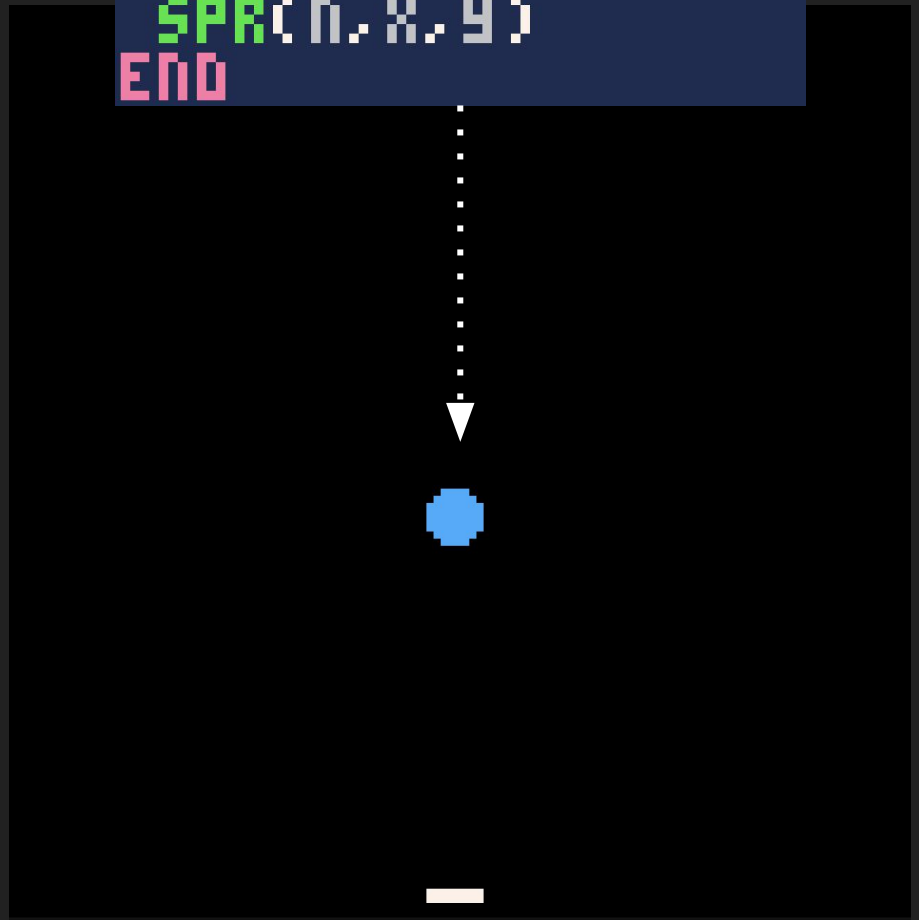
Frame	y
0	2
1	4
2	6
3	8
30 (1 second)	62

After 1 second (30 frames), the ball will have moved 60 pixels beyond its initial position (from 2 to 62)

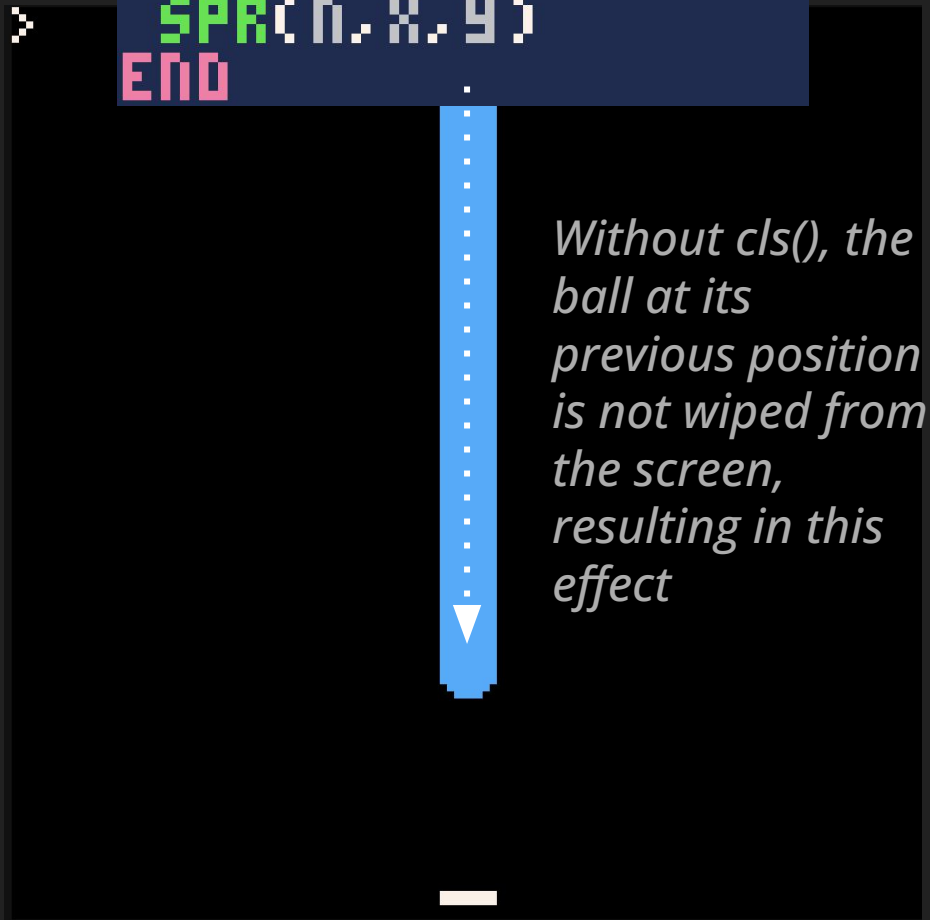
The PICO-8 Program Structure

- It’s important to begin the _draw() block with cls() to clear the screen
- This resets the screen each frame

```
function _draw()
 cls()
 spr(n,x,y)
end
```



```
function _draw()

 spr(n,x,y)
end
```



Without cls(), the ball at its previous position is not wiped from the screen, resulting in this effect

The PICO-8 Program Structure

- It helps to think through what is happening each frame
- *The code executes in the order it is written, and it does this* each *frame*

The PICO-8 Program Structure

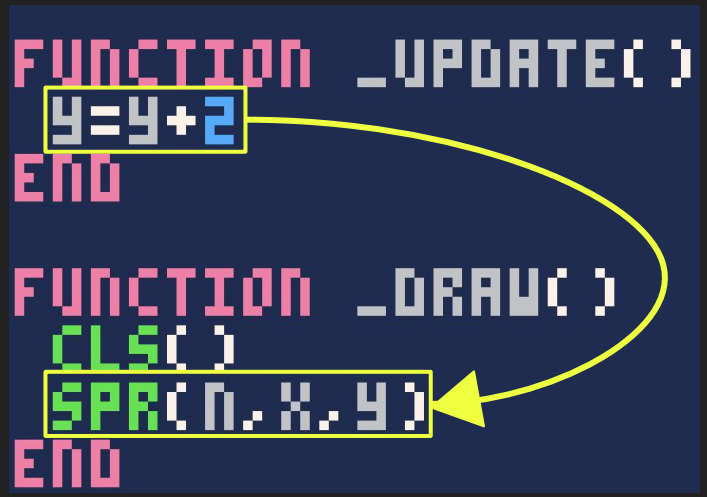
Frame	Effect
1 (Line 1 of _draw)	clear the screen
1 (Line 2 of _draw)	**draw the ball**
2 (Line 1 of _draw)	clear the screen
2 (Line 2 of _draw)	**draw the ball**
3 (Line 1 of _draw)	clear the screen
3 (Line 2 of _draw)	**draw the ball**

And so on . . .

```
function _draw()
  cls()
  spr(n,x,y)
end
```

The PICO-8 Program Structure

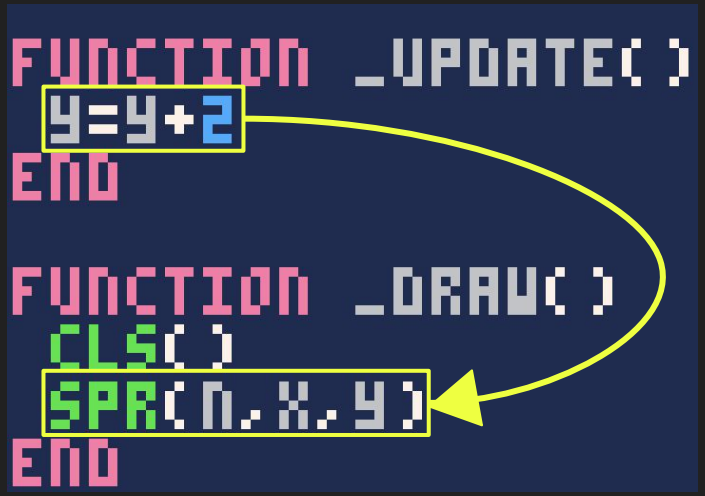
Frame	Effect
1 (Line 1 of _draw)	clear the screen
1 (Line 2 of _draw)	**draw the ball at y=4**
2 (Line 1 of _draw)	clear the screen
2 (Line 2 of _draw)	**draw the ball at y=6**
3 (Line 1 of _draw)	clear the screen
3 (Line 2 of _draw)	**draw the ball at y=8**



Even though the spr() code in _draw() isn't changing, the ball is being drawn in a different position because the value of y is being changed in _update, and the updated value is being used in _draw()

The PICO-8 Program Structure

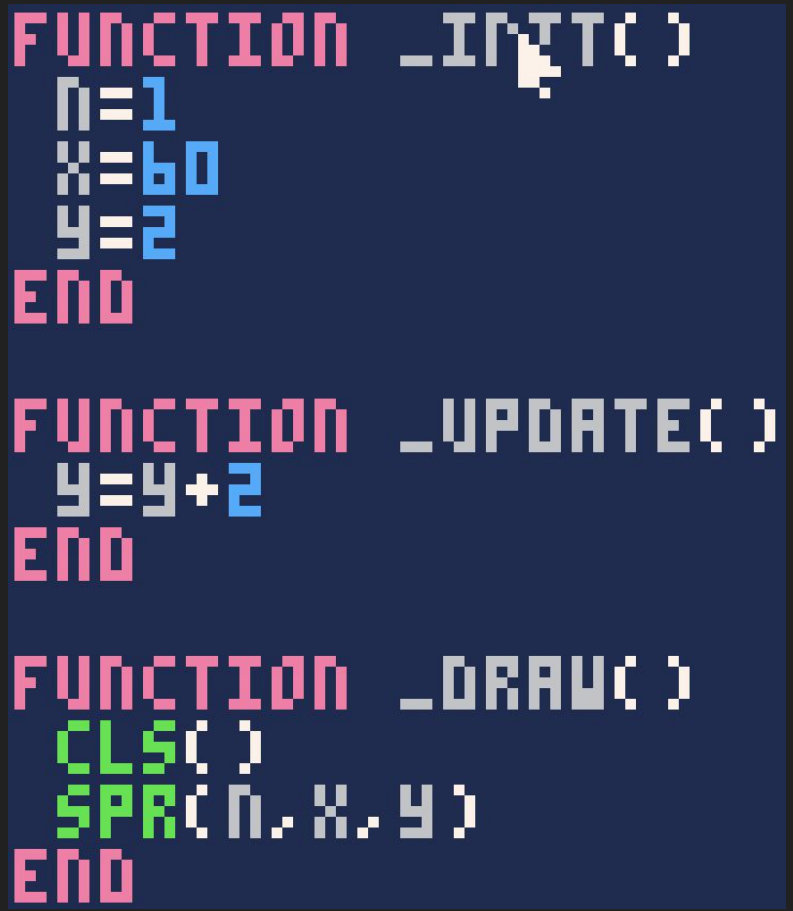
- _update() is where the value of **y** is changed
- Whatever the current value of **y** is, that's the value being fed into the spr() function in _draw() – ***this is what creates motion***



More about variables . . .

Using Variables

- Variables must be “declared” before they can be used
- When we type X=60 in _INIT(), we are declaring a variable named X and assigning it a value of 60



Using Variables

- If I fail to declare the variable Y, and then try to change its value, I will get an error

```
function _init()
 n=1
 x=60

end

function _update()
 y=y+2
end
```

```
runtime error line 7 tab 0
 y=y+2
attempt to perform arithmetic on
 global 'y' (a nil value)
in _update line 7 (tab 0)
at line 0 (tab 0)
```

Using Variables

If you ever see a reference to “A NIL VALUE” in your error message, *it’s likely that you did not define the variable (or function) before trying to do something with it*

```
function _init()
 n=1
 x=60

end

function _update()
 y=y+2
end
```

```
runtime error line 7 tab 0
 y=y+2
attempt to perform arithmetic on
 global 'y' (a nil value)
in _update line 7 (tab 0)
at line 0 (tab 0)
```

This is also a common error if you **change** a variable **name** in one place but not another

Troubleshooting Errors

Most runtime errors are caused by simple typos; if you get an error, check:

- Are there any unclosed parentheses?
- Are there any missing END keywords?
- Are all your variables and function names spelled correctly and consistently?

Variables and Input

Download Example File: paddleball_basic_04_input_and_movement.p8

Variables & Input

- We can use variables to make the paddle move when the player inputs a keypress
- We'll do this inside the _update() function

```
function _init()

 -- ball
 bal_n=1
 bal_x=60
 bal_y=2

 -- paddle
 pad_n=2
 pad_x=60
 pad_y=118

end
```

1. We can set variables for both the ball and paddle in _INIT()

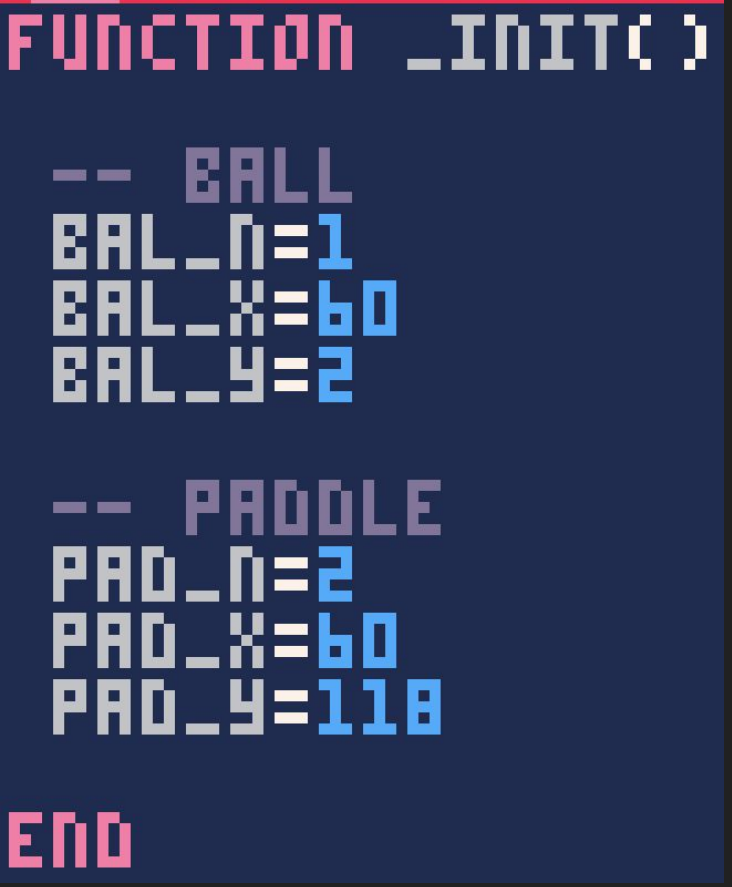
2. We can then use the BTN() function to check for a button press in _UPDATE()

```
function _update()
 if btn(⬅️) then
  pad_x = pad_x - 1
 end
end
```

3. Finally, apply those variables in the SPR() function in _DRAW()

```
function _draw()
 cls()
 spr(bal_n,bal_x,bal_y)
 spr(pad_n,pad_x,pad_y)
end
```

Some rules for naming variables . . .



There are a few rules when naming variables:

- The variable name *cannot be another recognized word like* spr *or* _draw
- The variable name must be one word and cannot have spaces
- Underscores **ARE** acceptable, so use these instead of spaces, or just make the variable name one unbroken word

PICO-8 allows you to use **symbols** as variable names

But many other languages require that the variable name:

- *must **NOT** contain **symbols***
- *must **NOT** begin with a **number***

Detecting Input

Download Example File: paddleball_basic_04_input_and_movement.p8

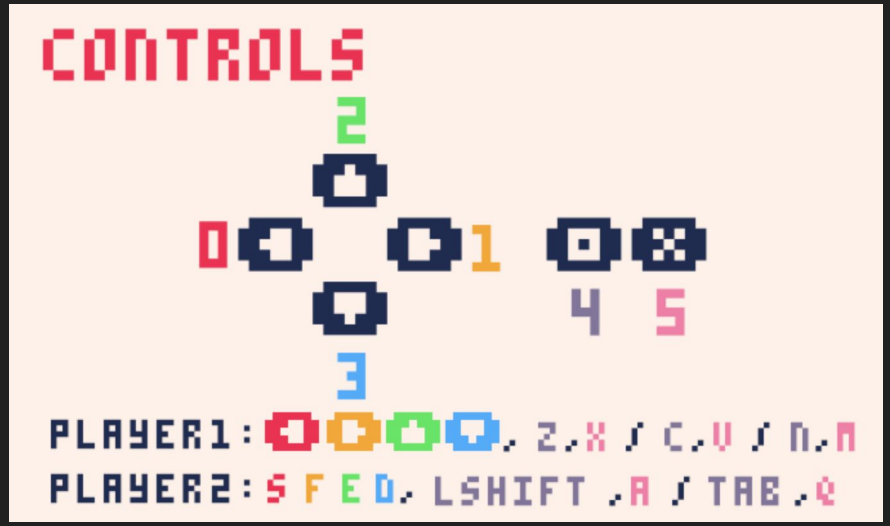
- Because the **_UPDATE()** function is used for **calculations**, this is where we will move our paddle sprite when a key is pressed

- **We can use the keywords IF and THEN to create a "conditional statement"**
- **If the condition following IF is met, the code written after THEN will be executed**

```
FUNCTION _UPDATE()
  IF BTN(⬅️) THEN
    PAD_X = PAD_X - 1
  END
END
```

- **The BTN() function checks whether a particular key was pressed**
  - **You'll specify *which* key you're checking for inside the parentheses**
- *Type **SHIFT L** to create a **left arrow icon** to represent the **left arrow key***

- You can type **SHIFT** and the first letter of a directional key to produce a **symbol** for that arrow
  - SHIFT L = left arrow
  - SHIFT R = right arrow
  - SHIFT U = up arrow
  - SHIFT D = down arrow
  - SHIFT O = O key (the letter O)
  - SHIFT X = X key
- Each of these arrow keys also has a numeric code associated with it
  - 0 = left arrow
  - 1 = right arrow
  - 2 = up arrow
  - 3 = down arrow
  - 4 = Z / C
  - 5 = X

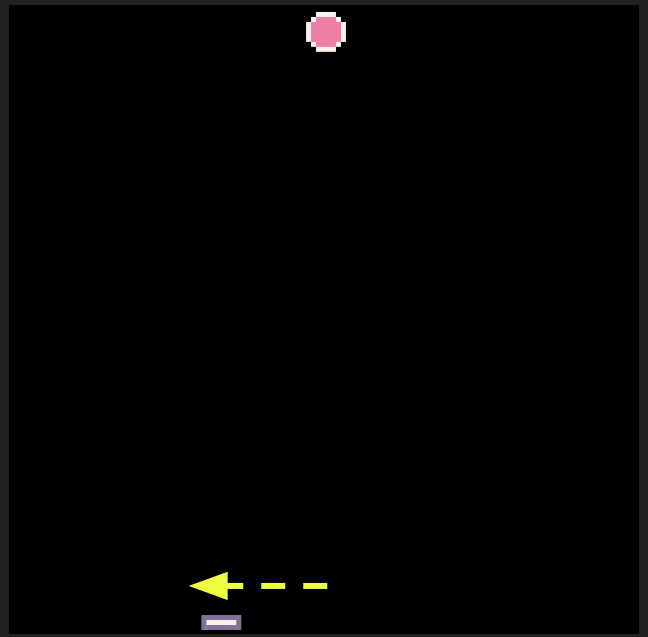


Refer to the [PICO-8 Cheat Sheet] for key codes

```
FUNCTION _UPDATE()
 IF BTN(0) THEN
  PAD_X = PAD_X - 1
 END
END
```

- To **change the paddle's x position**, we can take the variable PAD_X (*or whatever you named your variable*) and add or subtract from its value
- To move left, we'll **subtract 1 from the value of PAD_X**

```
FUNCTION _UPDATE()
 IF BTN(⬅️) THEN
  PAD_X = PAD_X - 1
 END
END
```



- We can shorten this by using a **minus sign followed by an equals sign** (**-=**) to represent the same operation
- And we can use **+=** to do the same thing but add to the value instead of subtract
- This process is called **incrementing** (increasing) or **decrementing** (decreasing)

```
function _update()
 if btn(⬅️) then
  pad_x = pad_x - 1
 end
end
```

```
function _update()
 if btn(⬅️) then
  pad_x -= 1
 end
 if btn(➡️) then
  pad_x += 1
 end
end
```

```
PAD_X = PAD_X - 1
```

Each expression means the same thing
(subtract one from the variable's value)

```
PAD_X -= 1
```

“Closing” functions and if-statements

- Whenever we define a function using the keyword **FUNCTION**, we need to "**close**" the function by typing the keyword **END** at the end of the block of code
- A "**block**" is just several related lines of code

- We also need to “**close**” our **IF/THEN** blocks using the keyword **END**

*Some languages, such as JavaScript, use **curly brackets { }** instead of **END***

If you don’t close a function or if-statement, it can result in an error or unexpected behavior

- Notice how the IF/THEN block fits “**between**” the FUNCTION/END block
- We call this “**nesting**” (like the Russian nesting dolls)
- It’s good practice to **indent** any nested lines of code

This way, we can see which ENDs are attached to which FUNCTIONs, IFs, etc.

```
FUNCTION _UPDATE()
 IF BTN(⬅️) THEN
 |PAD_X = PAD_X - 1
 END
END
```

- Visual Studio Code displays a **vertical line connecting the beginnings and ends of each block**, which can help you visualize the nesting structure

```
function _update()
    if btn(⬅️) then
        pad_x = pad_x - 1
    end
end
```

Make sure you use the tab key to create an indentation (spaces may not produce this effect)

- If you get an **error** when you run your game, often it is due to an **unclosed block**
- I prefer to use VS Code to troubleshoot because it more clearly illustrates the beginnings and ends of each block

```
function _update()
    if btn(⬅️) then
        pad_x = pad_x - 1
    end
end
```

Make sure you use the tab key to create an indentation (spaces may not produce this effect)

- **In VS Code, make sure you use the tab key to create an indentation (spaces may not produce this effect)**
- ***Also, indentation between VS Code and the PICO-8 editor may not be the same – choose one editor and stick with it***

You can also use **CTRL/CMD]** to indent a selected block and **CTRL/CMD [** to outdent (move left)

```
function _update()
    if btn(⬅️) then
        pad_x = pad_x - 1
    end
end
```

- Sometimes, I'll even go so far as to add **comments** after each END statement to remind myself what they are closing
- Write a comment in PICO-8 using **two dashes** (the minus-sign key) **--**

```
function _update()

    -- move left
    if btn(⬅️) then
        pad_x -= pad_speed
    end -- end if btn(⬅️)

    -- move right
    if btn(➡️) then
        pad_x += pad_speed
    end -- end if btn(➡️)

end -- end function _update()
```

VS Code is also helpful in that it highlights the end statements that close each block

- Comments are an underrated aspect of programming – **use them as much as you want to provide reminders for yourself**; you'll forget why you wrote something the way you did after some time has passed!

```
function _update()

    -- move left
    if btn(⬅️) then
        pad_x -= pad_speed
    end -- end if btn(⬅️)

    -- move right
    if btn(➡️) then
        pad_x += pad_speed
    end -- end if btn(➡️)

end -- end function _update()
```

Comments also help other people understand your code

- You can also use comments to “**deactivate**” lines of code without permanently deleting them
- This comment will stop the game from drawing the ball
- You can remove the comment to “reactivate” the code

```
function _draw()
    cls()
    --spr(bal_n,bal_x,bal_y)
    spr(pad_n,pad_x,pad_y)
end
```

Our full program so far . . .

Variables & Input

This code:

- Defines variables for the sprite number and x,y position of both the ball and paddle
- Changes the value of the paddle's x coordinate when the left or right arrow keys are pressed
- Draws the ball and paddle at whatever value their x and y coordinates are

```
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
function _init()
    -- ball
    bal_n=1
    bal_x=60
    bal_y=2

    -- paddle
    pad_n=2
    pad_x=60
    pad_y=118
end

function _update()
    -- move left
    if btn(⬅️) then
        pad_x -= 1
    end

    -- move right
    if btn(➡️) then
        pad_x += 1
    end
end

function _draw()
    cls()
    spr(bal_n,bal_x,bal_y)
    spr(pad_n,pad_x,pad_y)
end
```

Variables & Input

- The paddle is probably moving too slowly to catch up to the ball in all instances
- We could simply change += 1 and -= 1 to += 3 and -= 3
- But what if we need to keep tweaking this value? It would be more convenient to only have to change it in one place instead of two
- In cases like this, it makes sense to create a new variable – we can call it SPEED

```
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
function _init()
    -- ball
    bal_n=1
    bal_x=60
    bal_y=2

    -- paddle
    pad_n=2
    pad_x=60
    pad_y=118
end

function _update()
    -- move left
    if btn(⬅️) then
        pad_x -= 1
    end

    -- move right
    if btn(➡️) then
        pad_x += 1
    end
end

function _draw()
    cls()
    spr(bal_n,bal_x,bal_y)
    spr(pad_n,pad_x,pad_y)
end
```

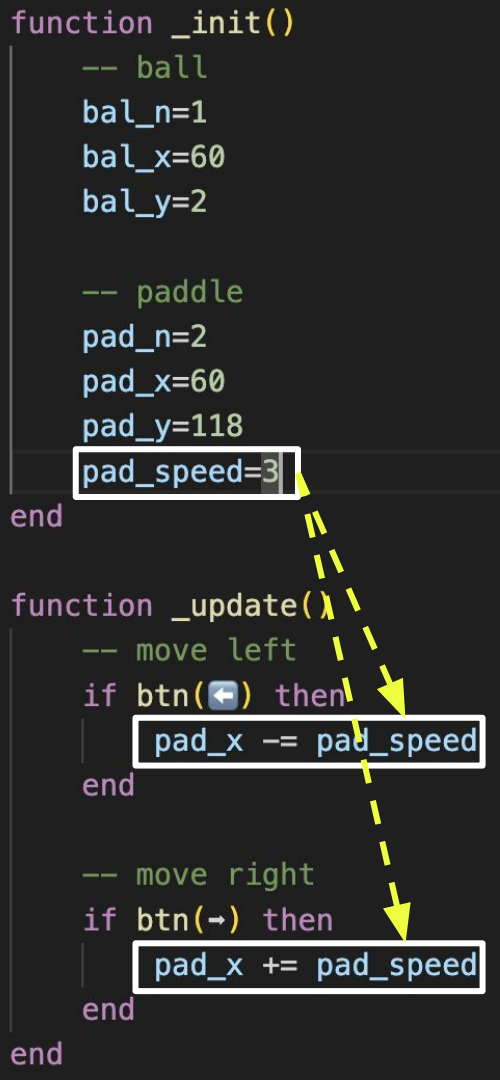
Variables & Input

- Here, I created a variable in _init() called pad_speed (for the paddle's speed) and set it equal to 3
- Then I applied that variable inside _update() in both the left and right movement blocks
- If I decide 2, 4, or some other value will work better, I only need to change the variable value once, in _init()

```
function _init()
    -- ball
    bal_n=1
    bal_x=60
    bal_y=2

    -- paddle
    pad_n=2
    pad_x=60
    pad_y=118
    pad_speed=3
end

function _update()
    -- move left
    if btn(⬅️) then
        pad_x -= pad_speed
    end

    -- move right
    if btn(➡️) then
        pad_x += pad_speed
    end
end
```

Variables & Input

- When you assign a variable a value and then refer to that variable later in your code, you can *think of that variable as standing for the value given to it*
- Here, **pad_speed means the same as 3** because I assigned pad_speed the value of 3



Keeping the Paddle on Screen

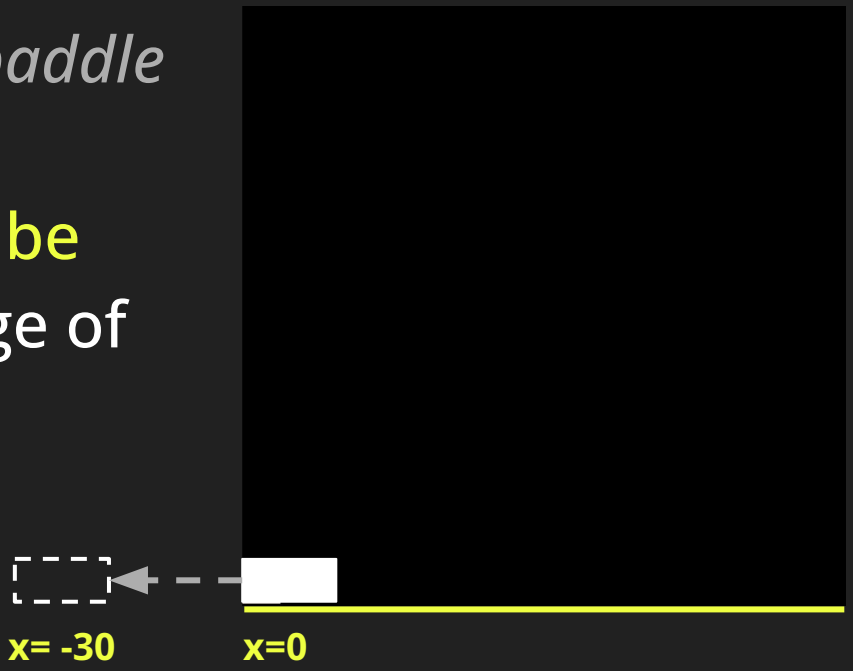
Download Example File: paddleball_basic_05_conditional_logic.p8

Keeping the Paddle on Screen

- You may have noticed that the player can continue moving the paddle with the arrow keys, even after it's gone *off screen* to the left or right
- We can use if/then statements to determine when the paddle has gone off screen and then correct its position

Keeping the Paddle on Screen

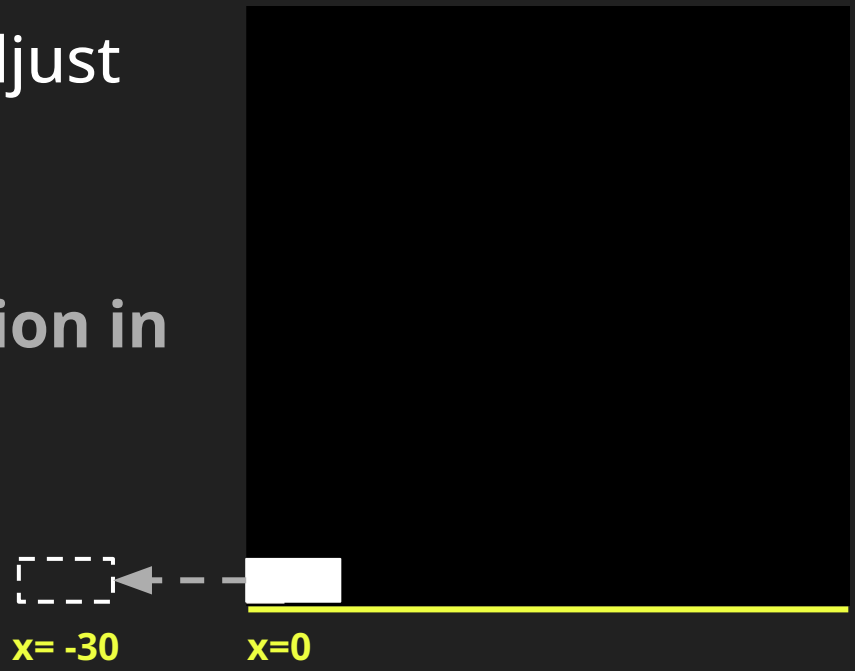
- *What do we **know** about the paddle when it's off screen?*
- We know its x position must be less than 0, since the left edge of the screen is at 0
- So we know we only need to adjust its position if its x is less than 0



Keeping the Paddle on Screen

- We know we only need to adjust the paddle's position if:
  its **x** is less than **0**
- **We can express this condition in the code as:**
  if padx < 0

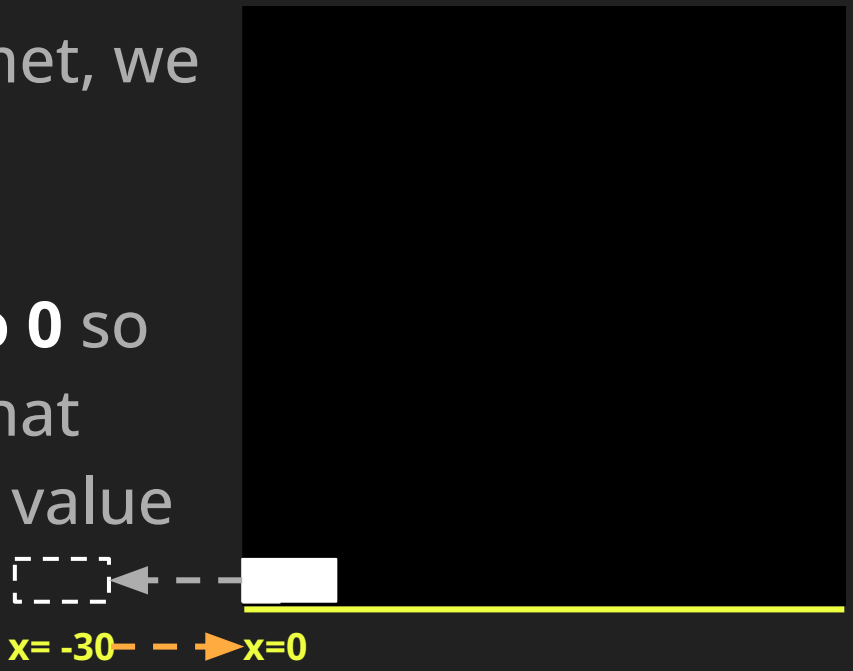
```
IF PADX < 0 THEN

END
```



Keeping the Paddle on Screen

- If the condition **padx < 0** is met, we need to adjust the paddle's position to keep it on screen
- We can simply **reset padx to 0** so that it never gets less than that
- This is called "*constraining*" a value

```
IF PADX < 0 THEN
  PADX = 0
END
```



Keeping the Paddle on Screen

- We'll place this code in our **_update()** function, **AFTER** the code that moves the paddle left when the key is pressed

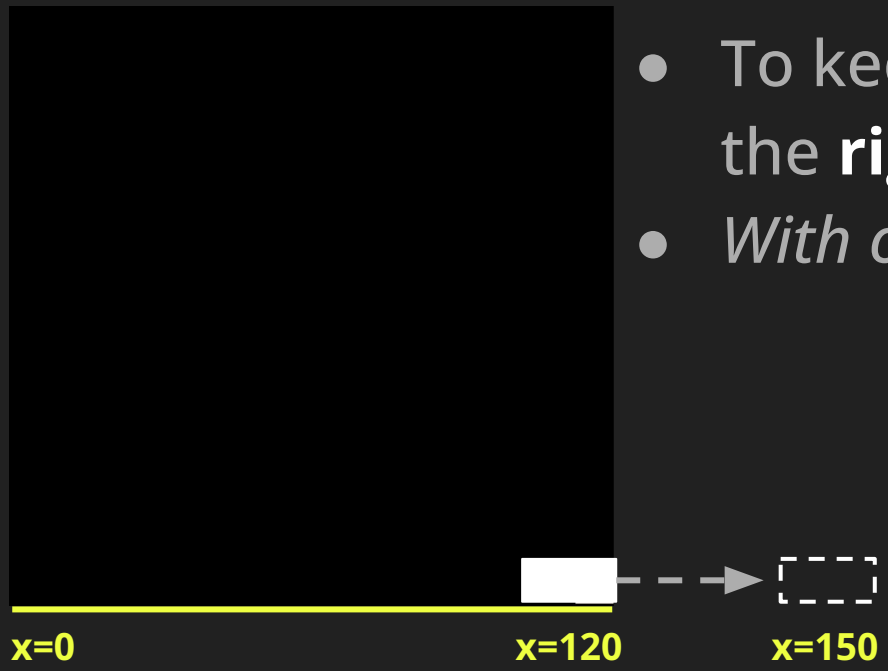
```
-- LEFT ARROW MOVES PAD LEFT
-- TYPE SHIFT L FOR ⬅
IF BTN(⬅) THEN
 PADX = PADX - PADSPD
END

-- RIGHT ARROW MOVES PAD RIGHT
-- TYPE SHIFT R FOR ➡
IF BTN(➡) THEN
 PADX = PADX + PADSPD
END

-- KEEP PAD ON SCREEN LEFT
IF PADX < 0 THEN
 PADX = 0
END
```

Keeping the Paddle on Screen

- If **padx** becomes less than **0** here . . .
- It gets reset back to **0** here

```
-- LEFT ARROW MOVES PAD LEFT
-- TYPE SHIFT L FOR ⬅
IF BTN(⬅) THEN
 PADX = PADX - PADSPD
END

-- RIGHT ARROW MOVES PAD RIGHT
-- TYPE SHIFT R FOR ➡
IF BTN(➡) THEN
 PADX = PADX + PADSPD
END

-- KEEP PAD ON SCREEN LEFT
IF PADX < 0 THEN
 PADX = 0
END
```

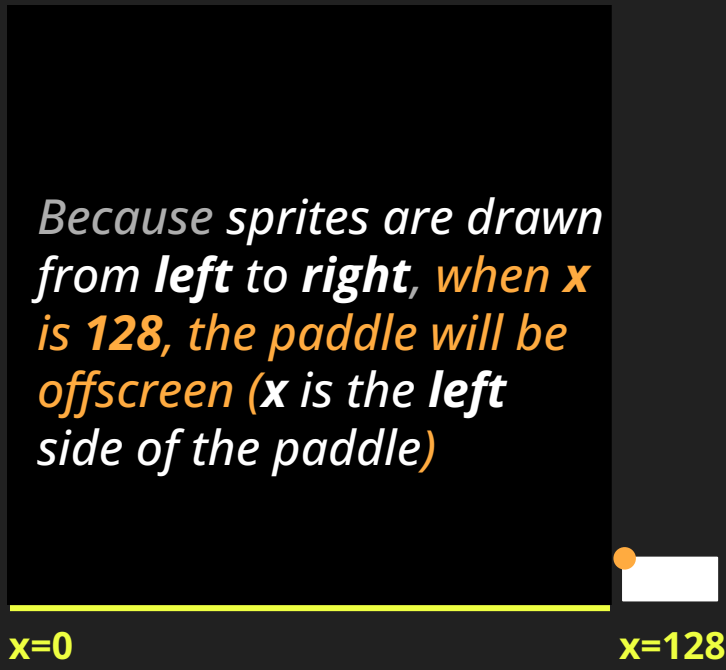
Keeping the Paddle on Screen



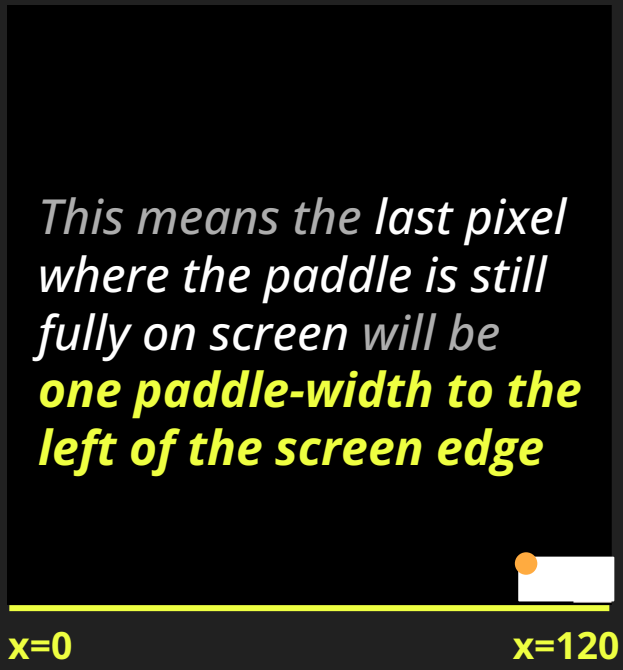
- To keep the paddle on screen to the **right**, it's a similar process
- *With one difference . . .*

Keeping the Paddle on Screen

*Because sprites are drawn from **left** to **right**, when **x** is **128**, the paddle will be offscreen (**x** is the **left** side of the paddle)*

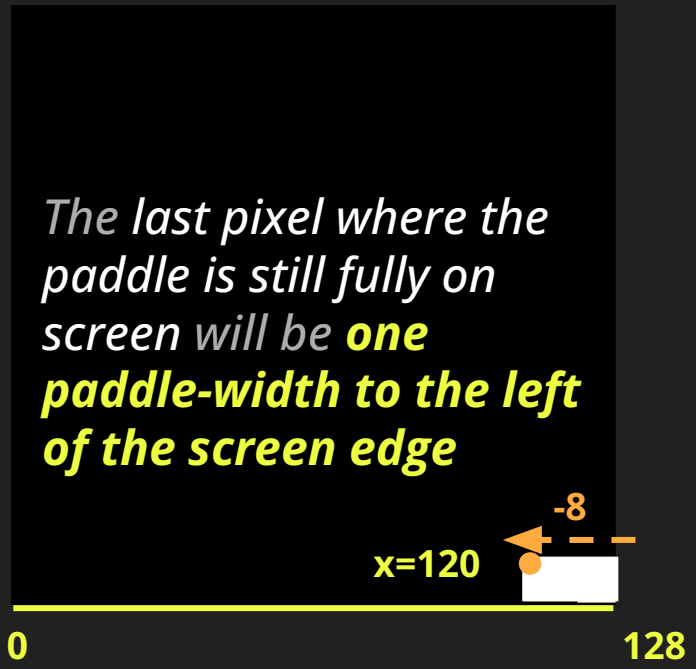


*This means the last pixel where the paddle is still fully on screen will be **one paddle-width to the left of the screen edge***



Keeping the Paddle on Screen

- **128** is the right edge of the screen
- If the paddle is **8px wide**, we subtract 8 from 128 to get the last point where the paddle will be fully on screen
- 128-8=**120**



Keeping the Paddle on Screen

- This is our full code for constraining the paddle to the screen bounds

```
-- KEEP PAD ON SCREEN LEFT
IF PADX < 0 THEN
 PADX = 0
END

-- KEEP PAD ON SCREEN RIGHT
IF PADX > 120 THEN
 PADX = 120
END
```

Keeping the Paddle on Screen

- An even better solution would be to use a **variable** for the paddle's width, so that the calculation still works if the paddle's size changes due to a bonus item or some other factor

```
-- KEEP PAD ON SCREEN LEFT
IF PADX < 0 THEN
 PADX = 0
END

-- KEEP PAD ON SCREEN RIGHT
IF PADX > 128-PADW THEN
 PADX = 128-PADW
END
```

Keeping the Paddle on Screen

- An even better solution would be to use a **variable** for the paddle's width, so that the calculation still works if the paddle's size changes due to a bonus item or some other factor

```
-- RUNS ONCE AT START
FUNCTION _INIT()

 -- DECLARE VARIABLES FOR PADDLE
 PADN = 1 -- SPRITE NUMBER
 PADX = 60 -- X COORDINATE
 PADY = 118 -- Y COORDINATE
 PADW = 8 -- WIDTH
 PADSPD = 3 -- SPEED
```

*We would just need to **declare** this variable first, in _init()*

```
-- KEEP PAD ON SCREEN LEFT
IF PADX < 0 THEN
 PADX = 0
END

-- KEEP PAD ON SCREEN RIGHT
IF PADX > 128-PADW THEN
 PADX = 128-PADW
END
```

Next, we'll work more with if-statements and start creating our own functions

Writing Custom Functions

Download Example File: paddleball_basic_06_functions.p8

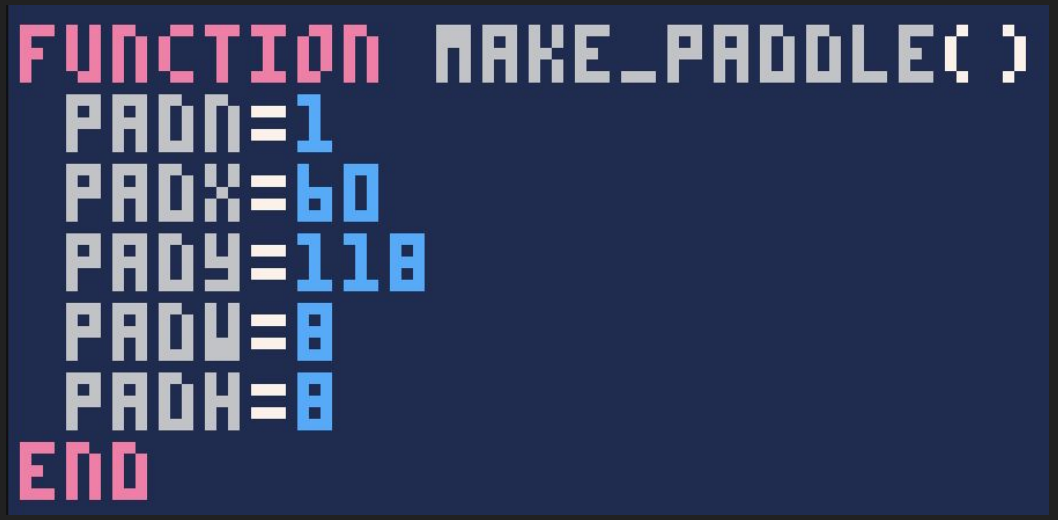
Writing Custom Functions

- We can write our *own* functions to do whatever we want – use the function keyword, followed by a name and parentheses

Functions need to be "*closed*" by including the keyword end at the end

Writing Custom Functions

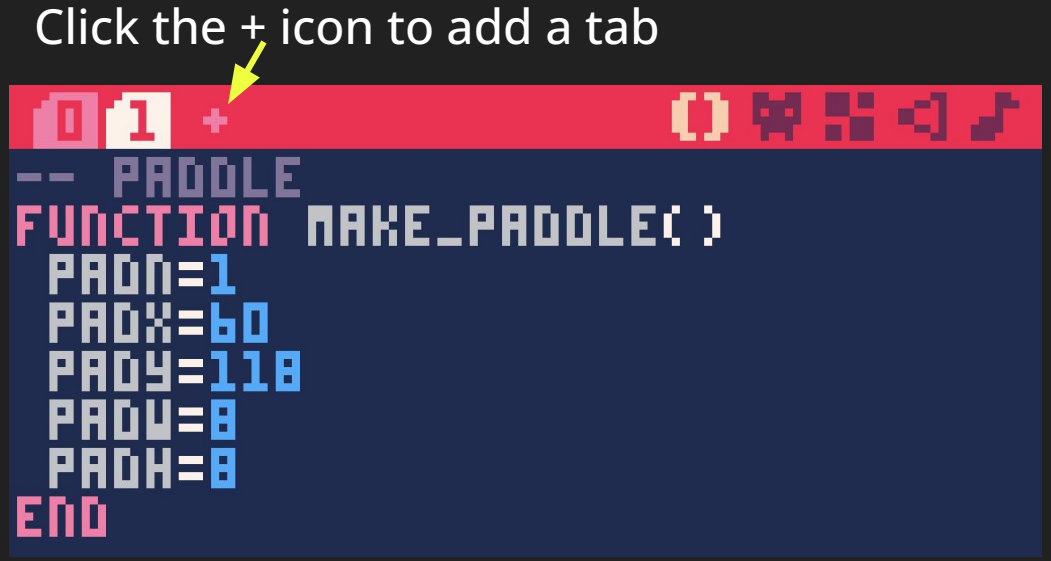
- We can write a custom function to define all our paddle variables



I named mine make_paddle(), but you can name yours whatever you want, as long as there are no spaces in the name (I used an underscore _ to connect compound words) and the function name begins with a letter

Writing Custom Functions

- I can write this custom function on a separate tab in the PICO-8 editor to save space on my main tab (0)



Writing Custom Functions

- *This code will not execute on its own*
- The function needs to be “called” elsewhere in the program

```
FUNCTION MAKE_PADDLE()
  PADN=1
  PADX=60
  PADY=118
  PADV=0
  PADH=0
END
```

You might also say the function must be “run,” “triggered,” or “executed” – calling a function means the same thing

Writing Custom Functions

- Think of a function like an item in your inventory
- It doesn’t do anything until you tell it to

```
FUNCTION MAKE_PADDLE()
  PADN=1
  PADX=60
  PADY=118
  PADV=8
  PADH=8
END
```

Writing Custom Functions

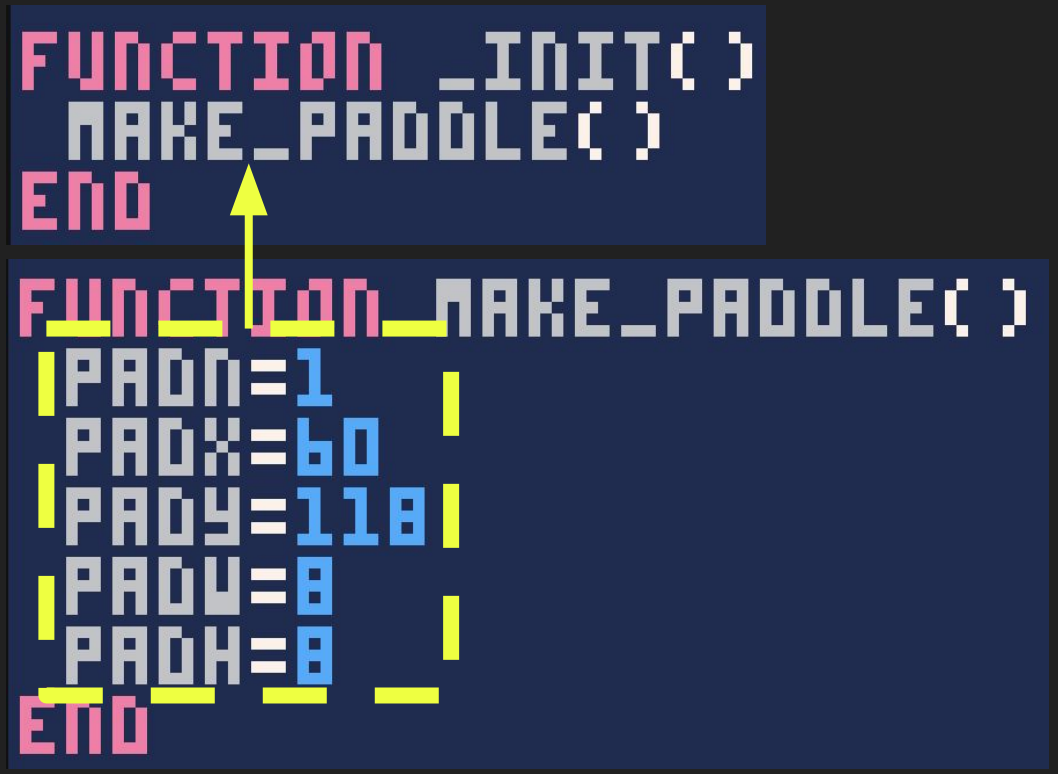
- We need to “call” our function in either _init(), _update(), or _draw()
- Here, I “call” make_paddle() in _init() since that’s where I declare variables

```
function _init()
 make_paddle()
end
```

```
function make_paddle()
 padn=1
 padx=60
 pady=118
 padv=8
 padh=8
end
```

Writing Custom Functions

- The code inside a function runs right when/where you call it
- Think of it as a shortcut or portal



Writing Custom Functions

```
function _init()
 padn=1
 padx=60
 pady=118
 padv=8
 padh=8
end
```

=

```
function _init()
 make_paddle()
end
```

```
function make_paddle()
 padn=1
 padx=60
 pady=118
 padv=8
 padh=8
end
```

Instead of declaring my variables directly in _init(), I can do that in a custom function and then “call” that function in _init()

Writing Custom Functions

```
function _init()
 padn=1
 padx=60
 pady=118
 padv=0
 padh=0
end
```

=

```
function _init()
 make_paddle()
end
```

```
function make_paddle()
 padn=1
 padx=60
 pady=118
 padv=0
 padh=0
end
```

The code on the left will do the same thing as the code on the right; the only difference is how it's organized

This may seem like a lot of work for something that doesn't change the program's behavior … but *the result is more readable code*

With custom functions

```
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
function _init()
    make_paddle()
    make_ball()
end

function _update()
    move_paddle()
    move_ball()
end

function _draw()
    cls()
    draw_paddle()
    draw_ball()
end
-->8
-- functions defined on other tabs below
```

Without custom functions

```
pico-8 cartridge // http://www.pico-8.com
version 42
__lua__
function _init()
    pad_n = 1
    pad_x = 60
    pad_y = 118
    pad_spd = 3

    bal_n = 2
    bal_x = 60
    bal_y = 2
    bal_spd = 3
end

function _update()
    -- left arrow
    if btn(0) then
        pad_x -= pad_spd
    end -- end if btn(0)

    -- right arrow
    if btn(1) then
        pad_x += pad_spd
    end -- end if btn(1)

    -- keep on screen left
    if pad_x < 0 then
        pad_x = 0
    end -- end if paddle.x < 0
```

Next, let's *detect when the ball and paddle collide*, so we can *make the ball change direction* when this happens

Collision Detection

Download Example File: paddleball_basic_07_collision.p8

Collision Detection

- *Collision detection means determining when two objects in your game are touching*

Ball and paddle are colliding

Ball and paddle are NOT colliding

Collision Detection Illustrated.pdf

Collision Detection

- Collision detection is a bit more involved than anything we've done so far
- Many game engines will do collision detection for you
- But it helps to understand how it works

Collision Detection

- Basically, we want to compare the position of one object (the paddle) with the position of another object (the ball)

Ball and paddle are colliding

Ball and paddle are NOT colliding

Collision Detection

- We can use the ball's and the paddle's **x** and **y** values to compare their positions

Collision Detection Illustrated.pdf

Collision Detection

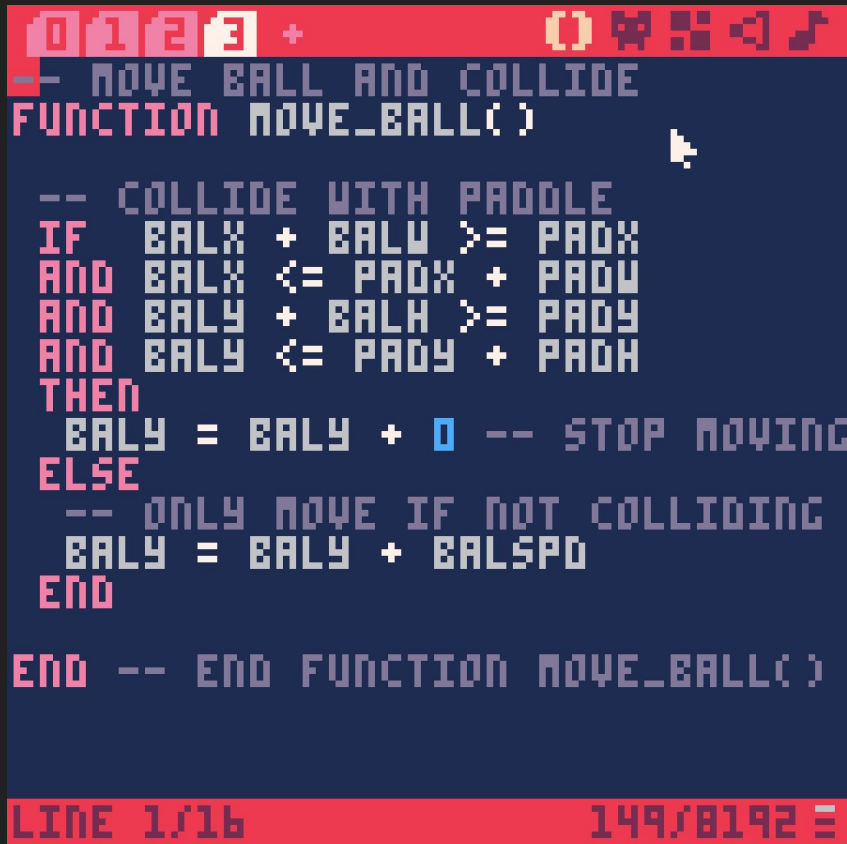
- We can use the ball's and the paddle's **x** and **y** values to compare their positions
- We'll also need to account for their width and height, so ***we'll need variables for the width and height***

Collision Detection

- I've added variables for the paddle's and ball's **width** and **height** (**PADW**, **PADH**, **BALW**, **BALH**)
- *The values are equal to the sprite dimensions*

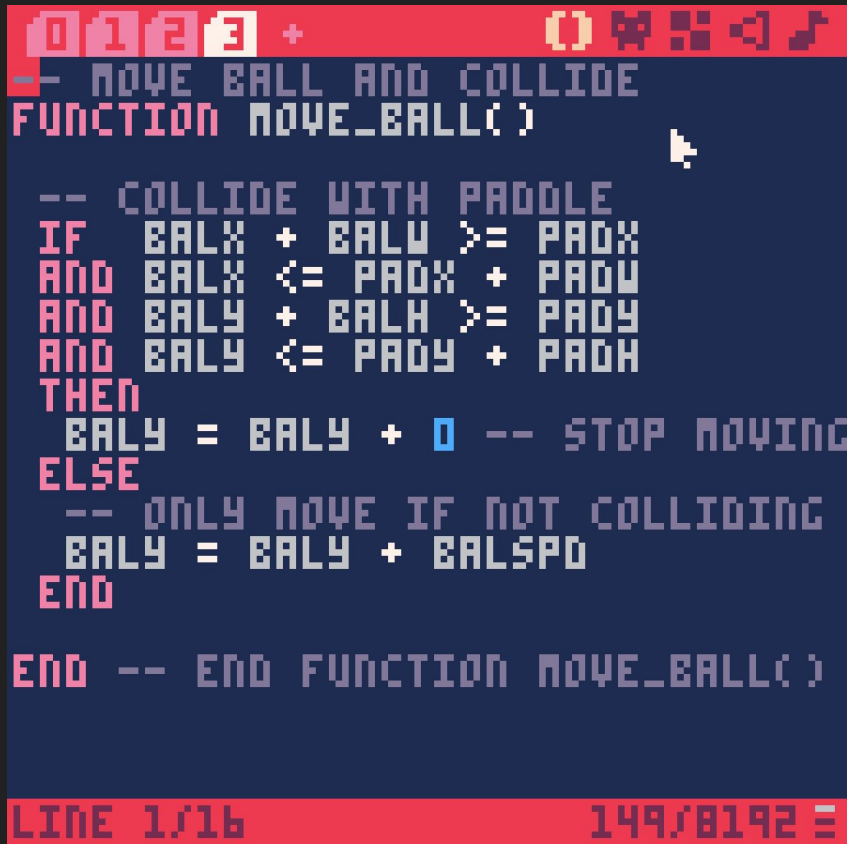
Collision Detection

- Next, we'll write code to determine whether the ball and paddle are touching
- I'll create a new function called **move_ball()** and write the code there, because *we'll want the ball to move in a different direction when it touches the paddle*

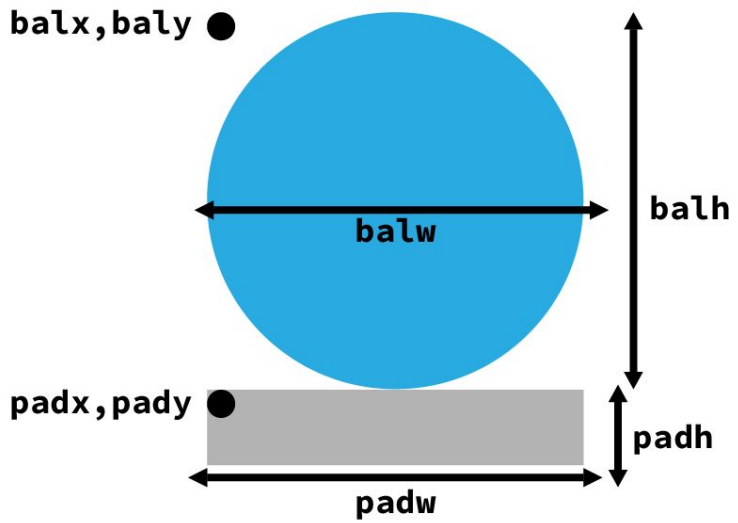


Collision Detection

Basically, we ***compare the position of opposite sides of each object against one another***



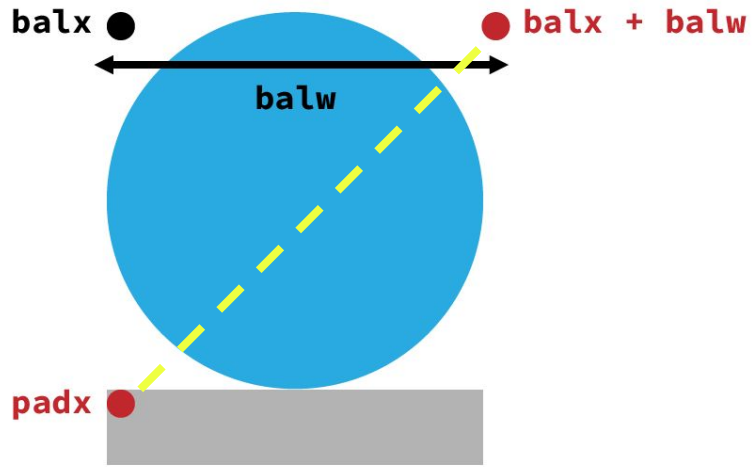
```
if balx + balw >= padx
and balx <= padx + padw
and baly + balh >= pady
and baly <= pady + padh
then there's collision!
```



We'll need **multiple conditions** to be met in order for the objects to be considered colliding

Let's break this down, one condition at a time . . .

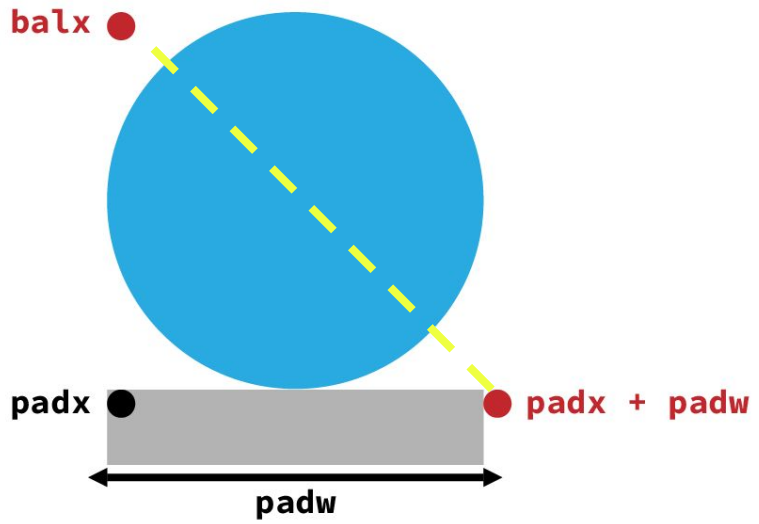
balx + balw >= padx



0 ——— 128

The right side of the ball (*measured as* ***balx + balw***) must have an **x** value greater than or equal to (**>=**) the left side of the paddle (*measured as* ***padx***)

balx <= padx + padw

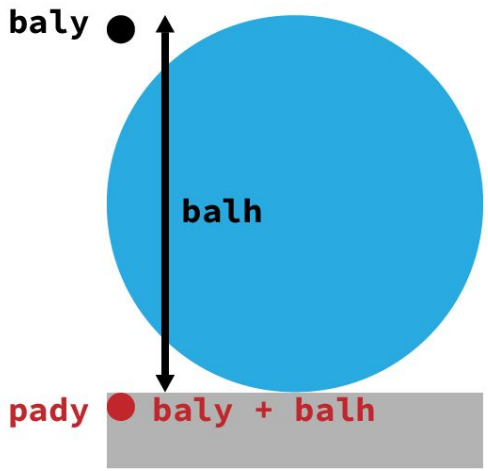


0 ——— 128

The left side of the ball (*measured as* ***balx***) must have an **x** value less than or equal to (**<=**) the right side of the paddle (*measured as* ***padx + padw***)

0

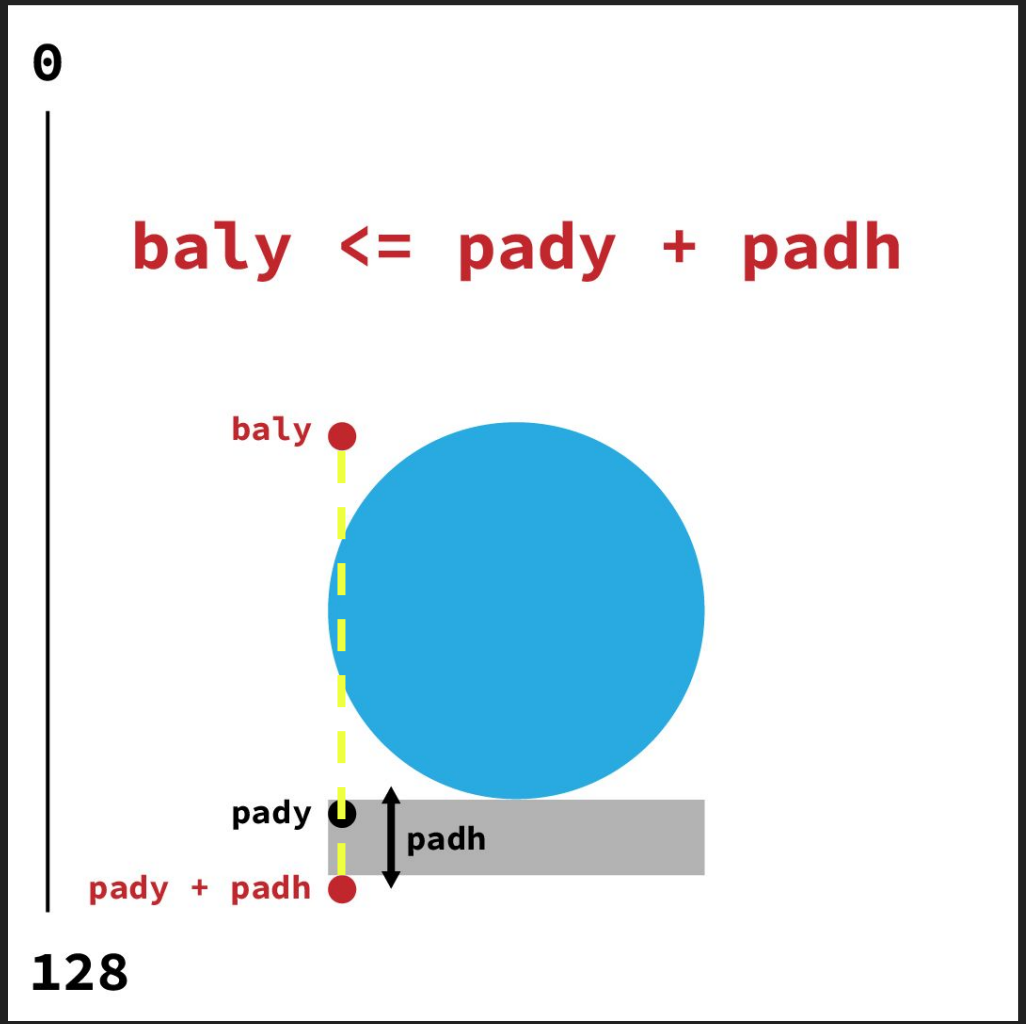
baly + balh >= pady



128

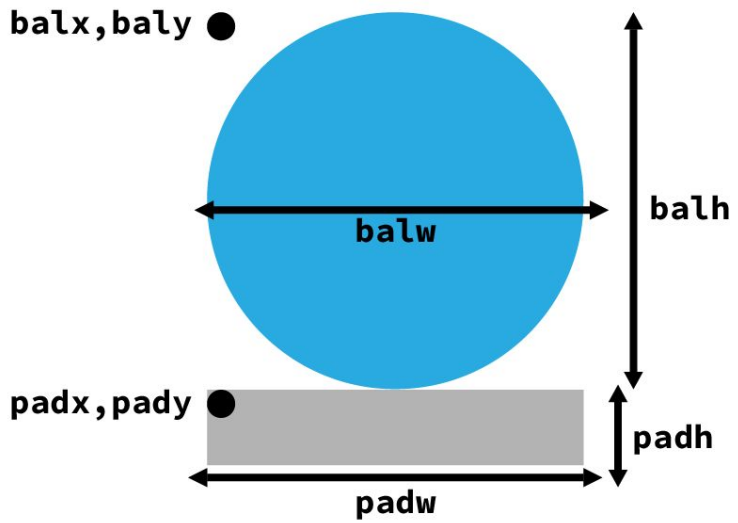
The bottom of the ball (*measured as* ***baly + balh***) must have a **y** value greater than or equal to (**>=**) the top of the paddle (*measured as* ***pady***)

Here, they're equal



The top of the ball (*measured as **baly***) must have a **y** value less than or equal to (**>=**) the bottom of the paddle (*measured as **pady** + **padh***)

```
 if balx + balw >= padx
and balx <= padx + padw
and baly + balh >= pady
and baly <= pady + padh
then there's collision!
```



If all four conditions are met, the ball and paddle must be touching (colliding)

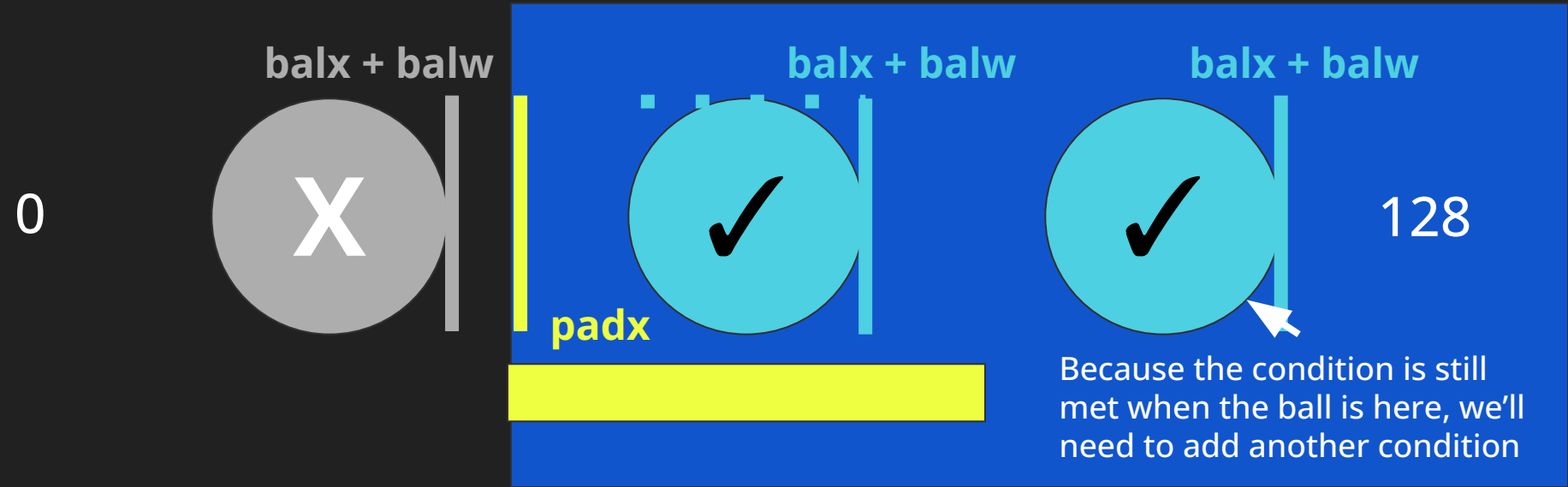
[Collision Detection Illustrated.pdf]

Let’s look at this a different way . . .

Collision Detection

balx + balw >= padx

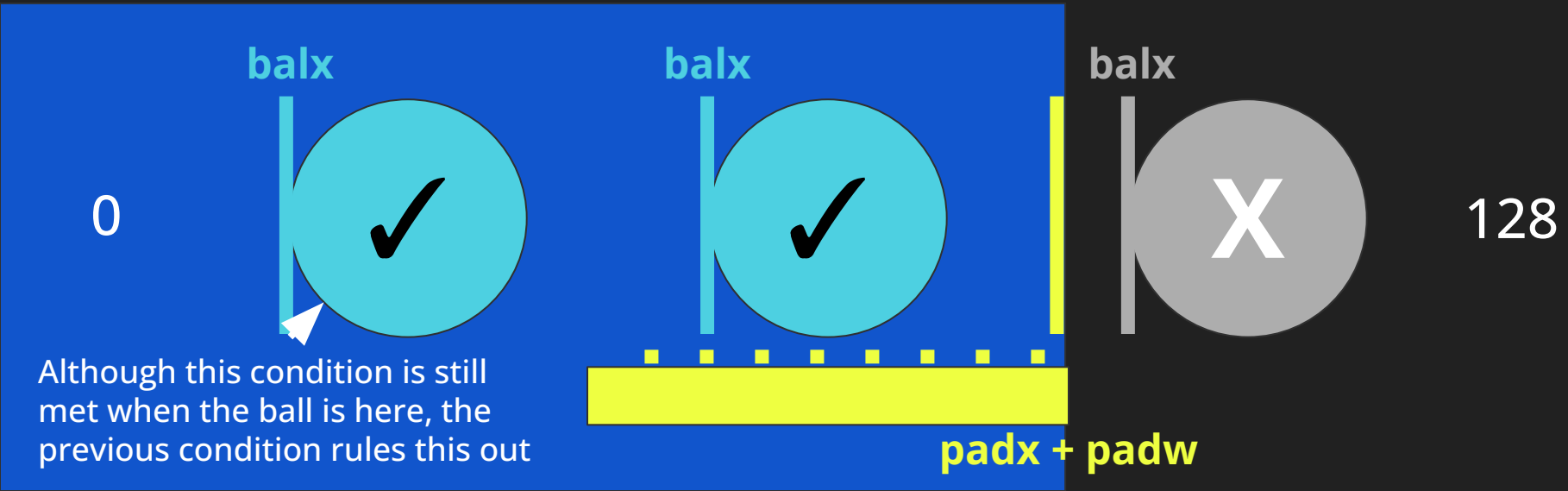
The right side of the ball (balx + balw) must be at or to the right of (>=) the left side of the paddle (padx)



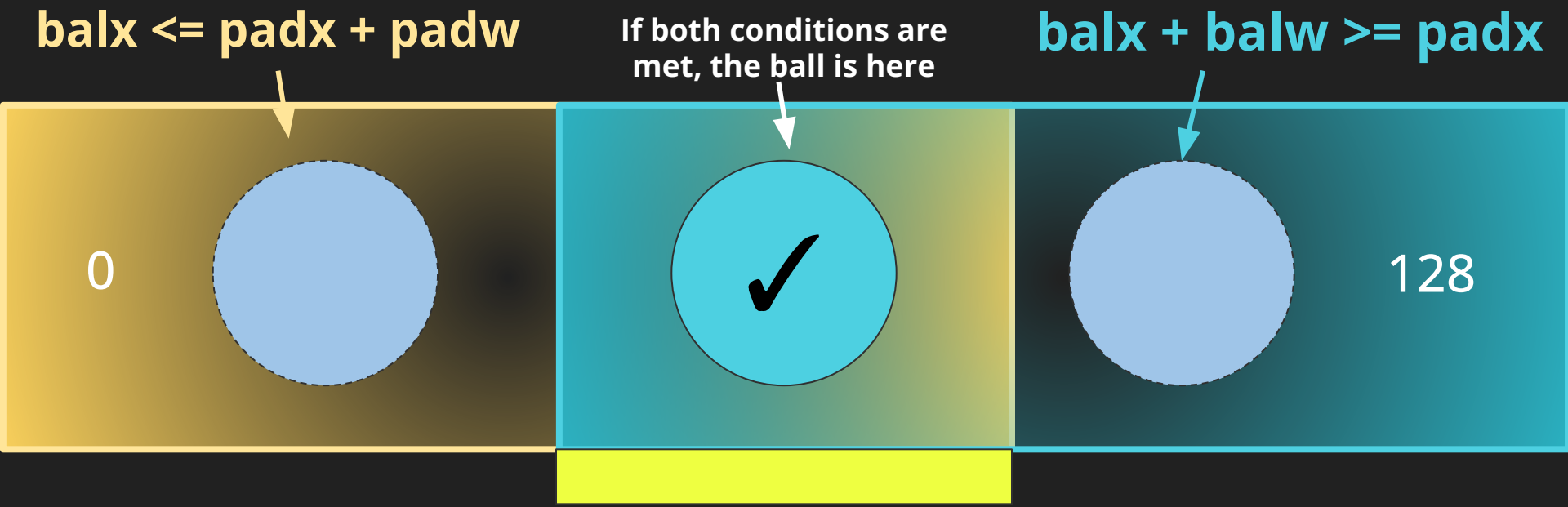
Collision Detection

balx <= padx + padw

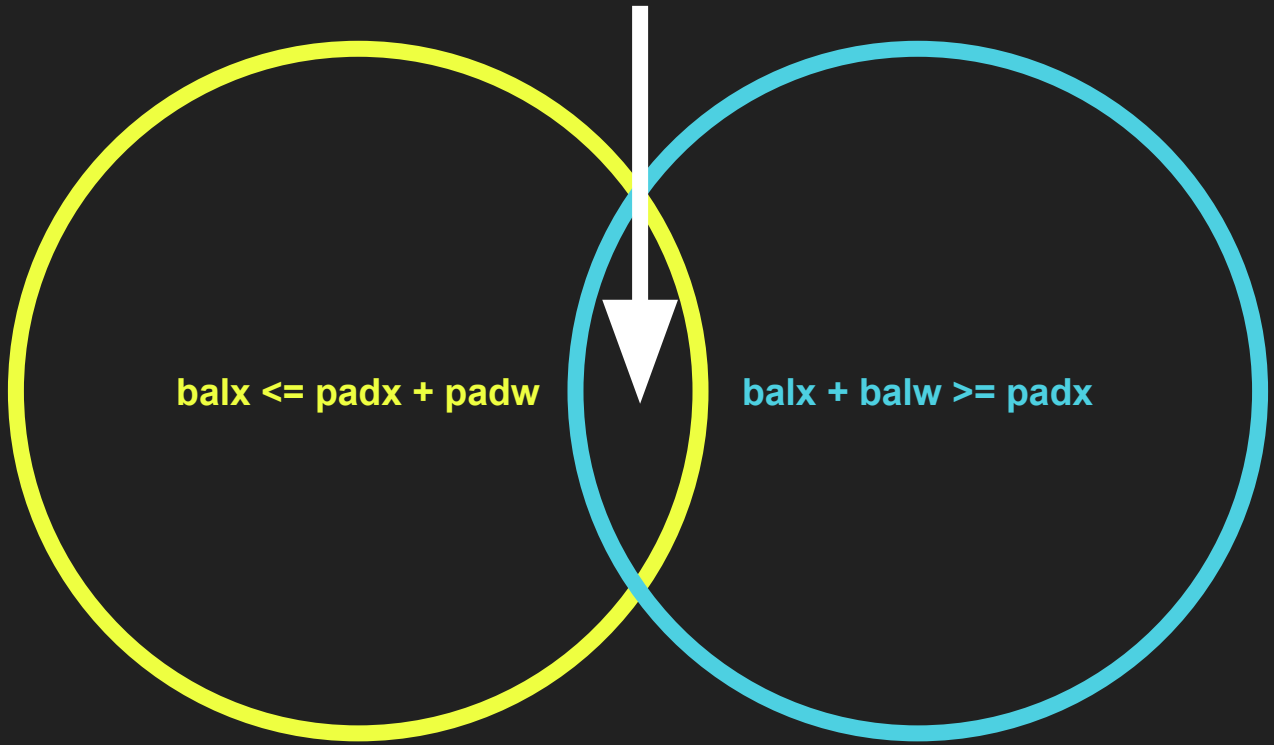
The left side of the ball (balx) must be at or to the left of (<=) the right side of the paddle (padx + padw)



Collision Detection



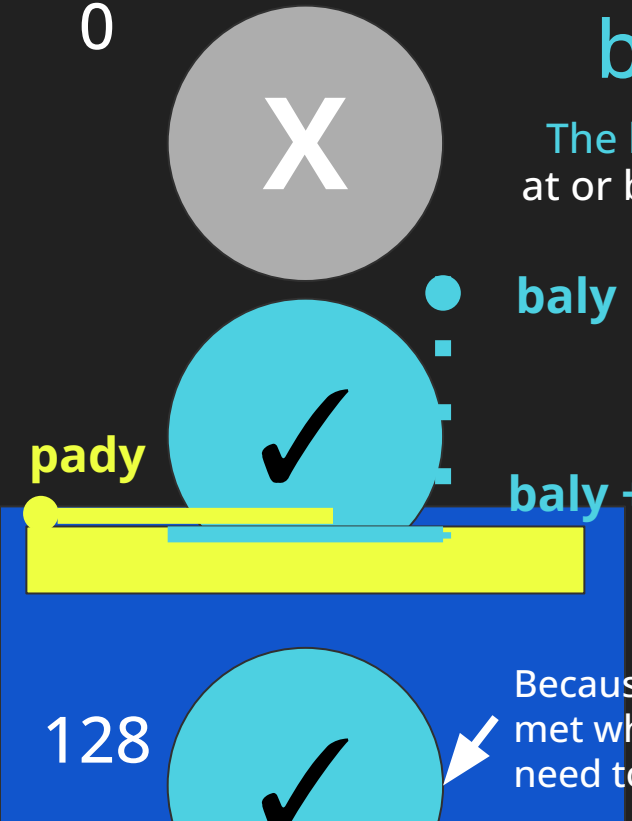
If both conditions are met, the ball is horizontally aligned with the paddle



Venn diagram

Collision Detection

0

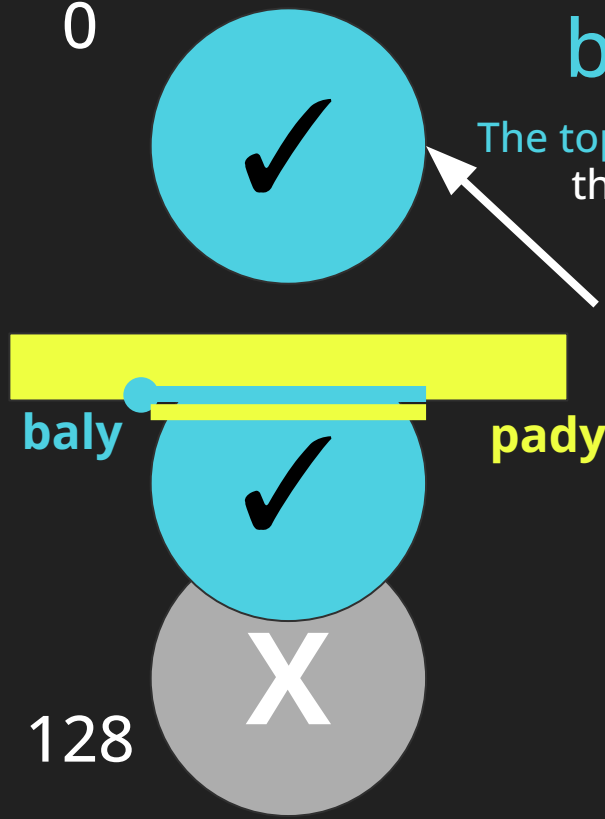


baly + balh >= pady

The bottom of the ball (baly + balh) must be at or beneath (>=) the top of the paddle (pady)

Because the condition is still met when the ball is here, we'll need to add another condition

Collision Detection



baly <= pady + padh

The top of the ball (baly) must be at or above (<=) the bottom of the paddle (pady + padh)

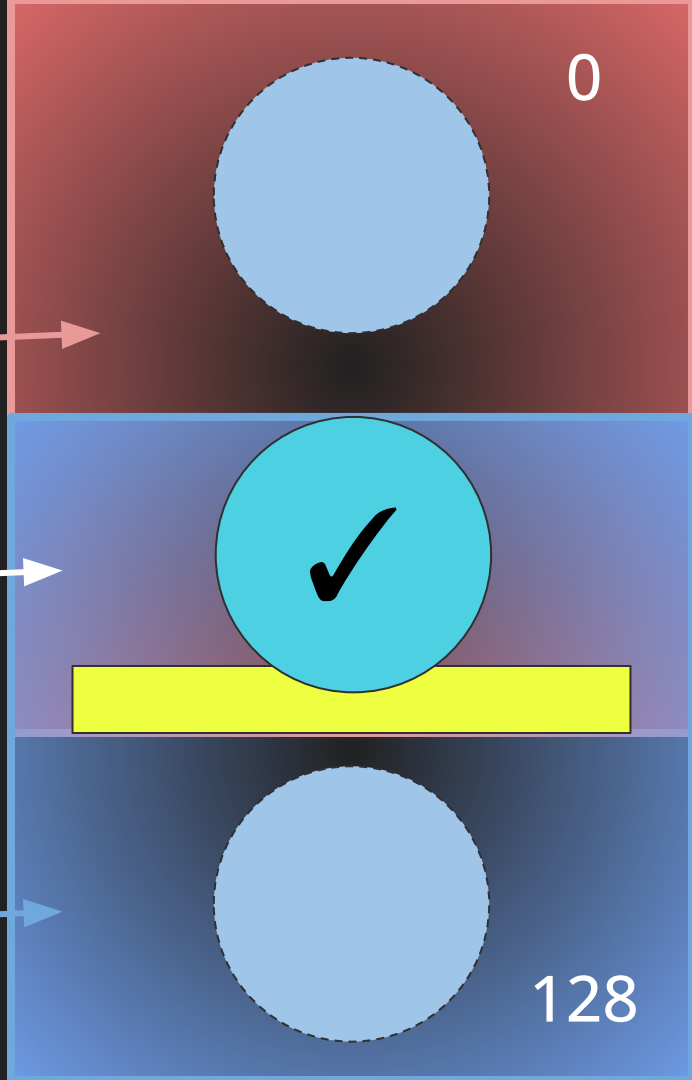
Although this condition is still met when the ball is here, the previous condition rules this out

Collision Detection

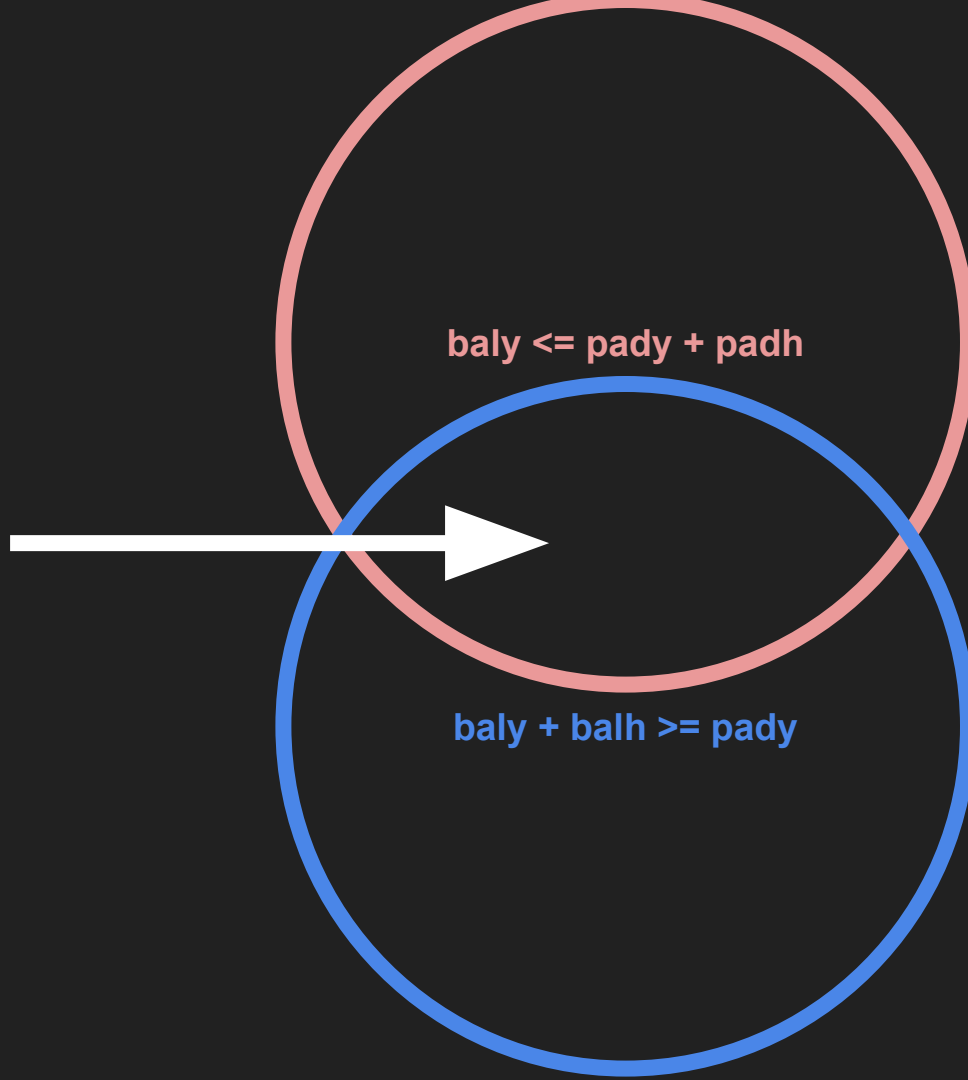
baly <= pady + padh

If both conditions are met, the ball is here

baly + balh >= pady



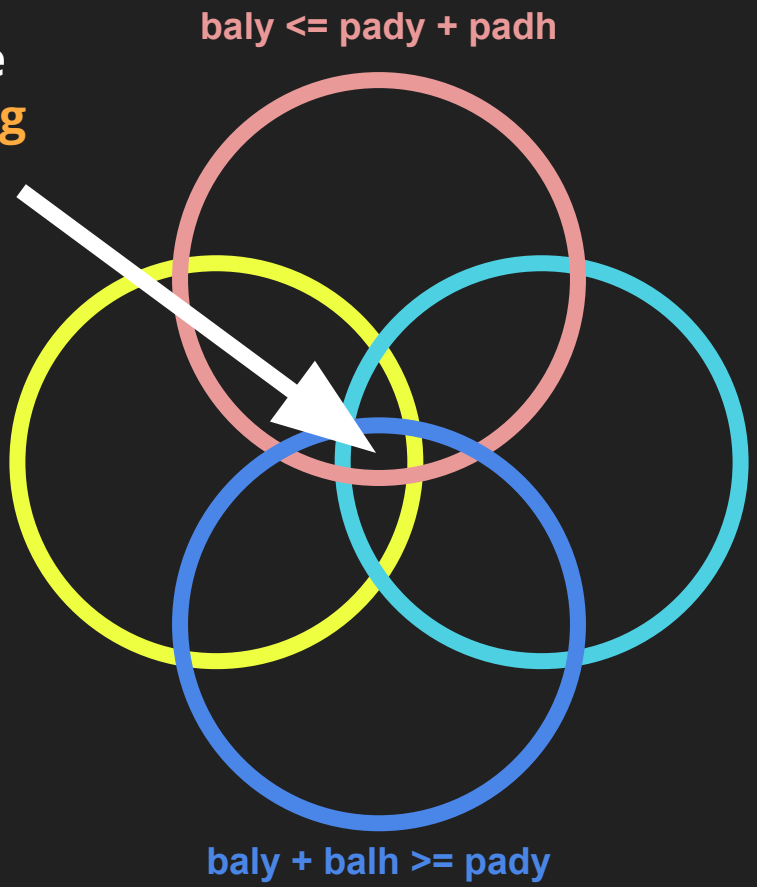
If both conditions are met, the ball is vertically aligned with the paddle



Venn diagram


baly <= pady + padh
balx <= padx + padw
balx + balw >= padx
If ALL conditions are met, the ball is here
baly + balh >= pady

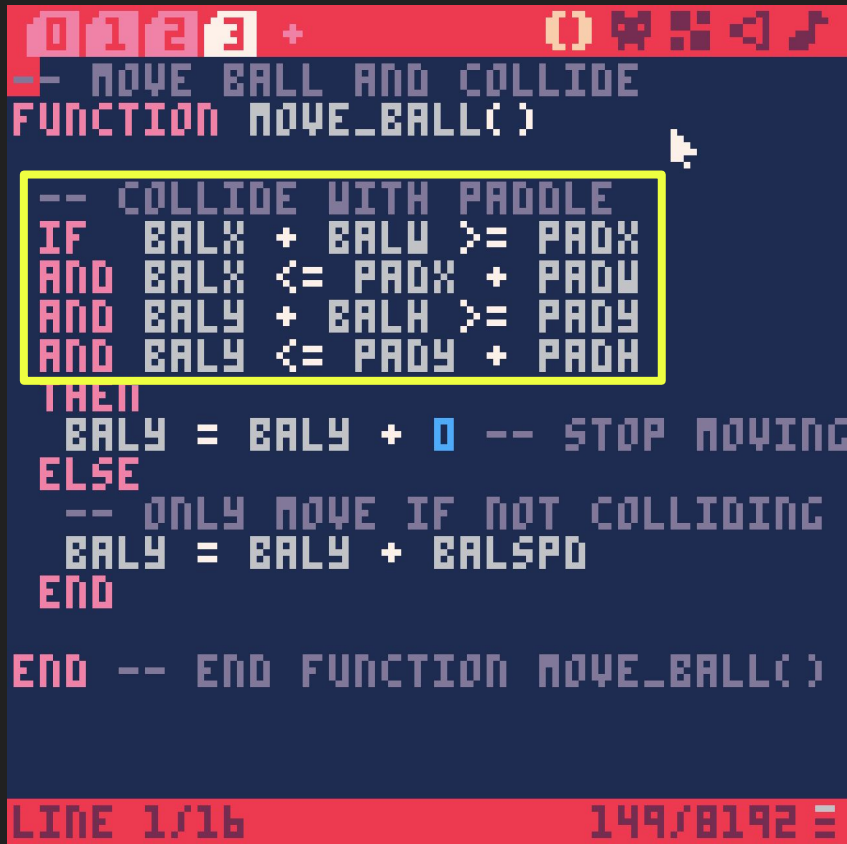
If ALL 4 conditions are met, the ball is colliding with the paddle



Venn diagram

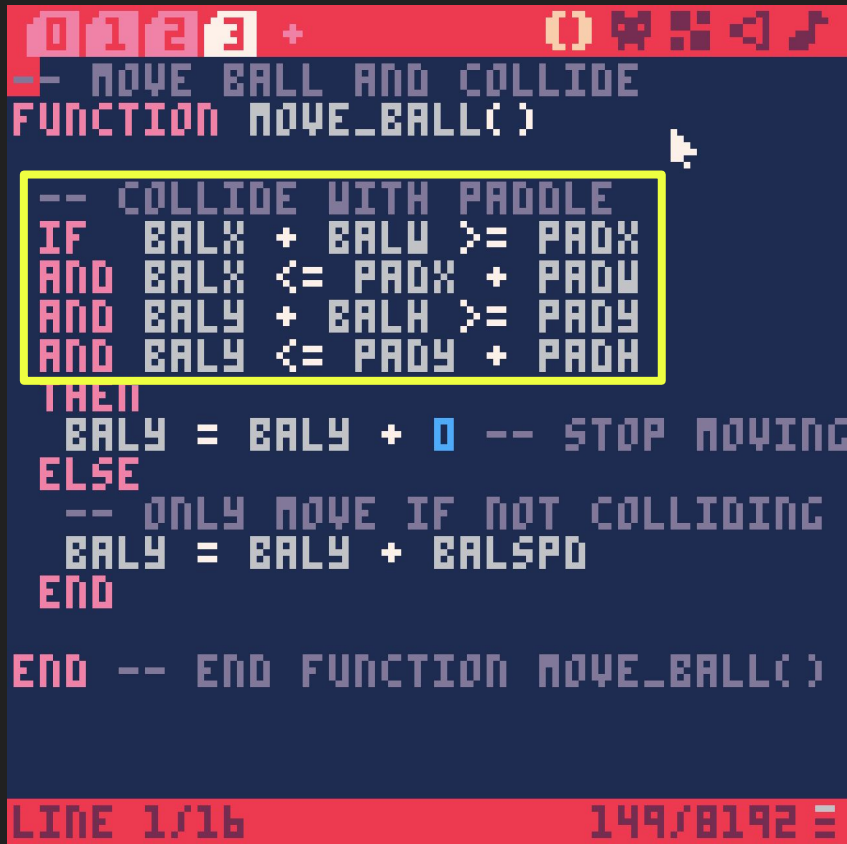
Collision Detection

If all four conditions are met, the ball is in collision with the paddle



Collision Detection

We can use the keyword **AND** to stack conditions (*require multiple conditions to be met*)

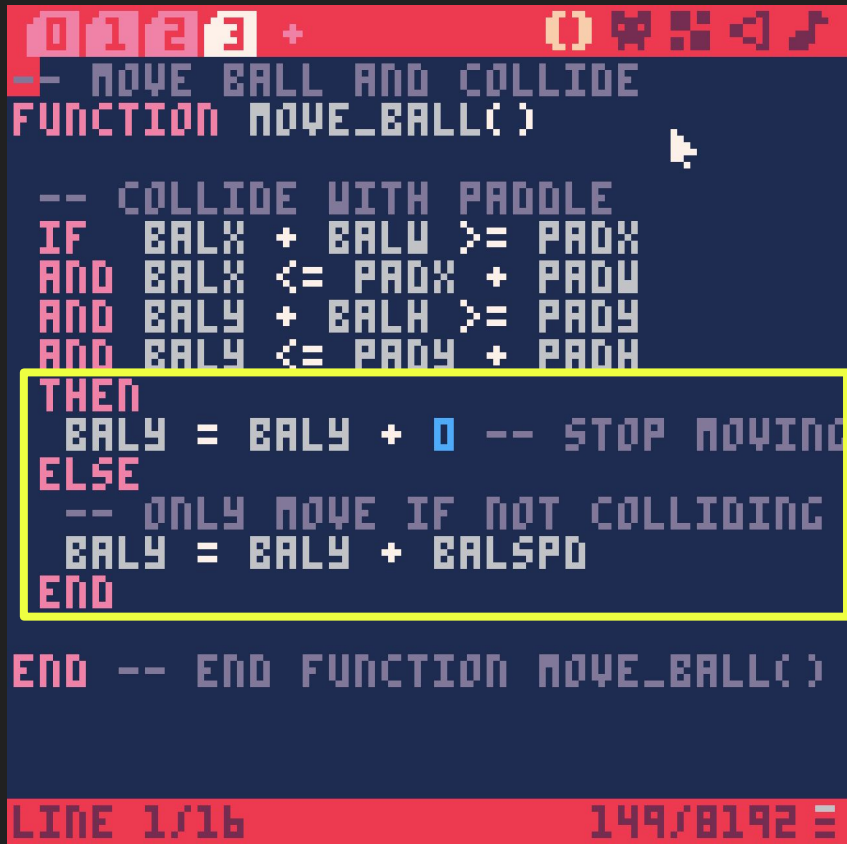


What do we do if all conditions are met and the ball is colliding with the paddle?

Collision Detection

To test whether your code is detecting collision correctly, you could choose to only move the ball if there is NOT collision

You can use the keyword **ELSE** to specify what should happen if the *opposite* of the condition(s) occurs



Collision Detection

Another way to stop the ball would be to set its speed to zero, so that nothing is being added to the y coordinate value

But what we really want is for the ball to bounce back up when it hits the paddle

Making the Ball Bounce

Download Example File: paddleball_basic_08_reverse_direction.p8

Making the Ball Bounce

I'm going to create a new variable for the ball's vertical (y-axis) direction (*is it moving up or down?*)

I'll name it **BALYDIR**

And I'll set its value to **"DOWN"**

You'll need to include quotation marks for a text value

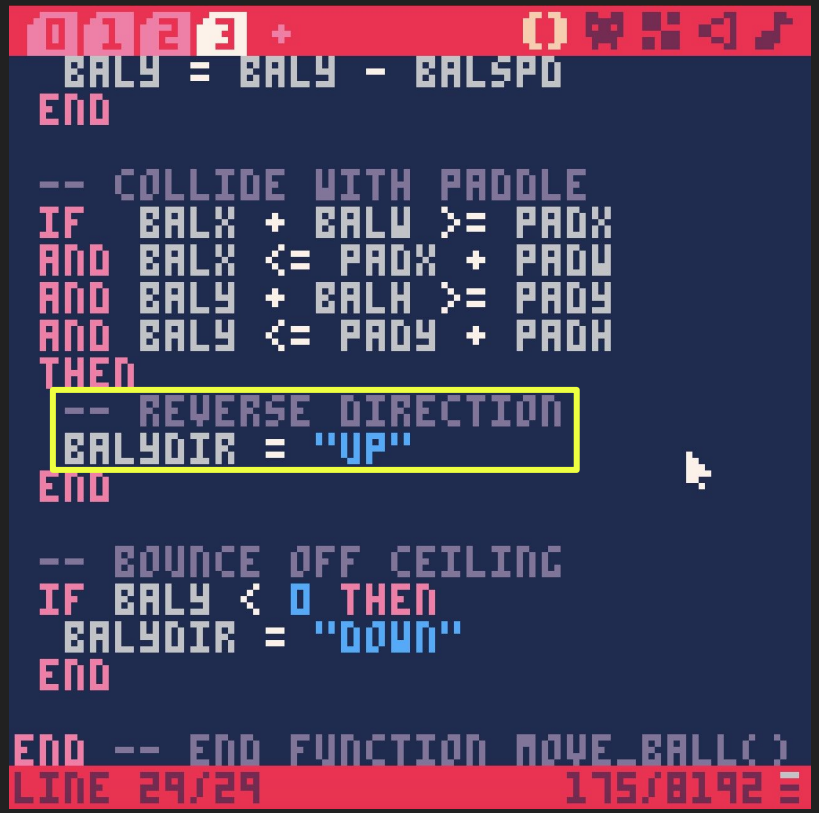
Making the Ball Bounce

While I'm at it, I'll create a variable for the horizontal direction called **BALXDIR**

I'll leave its value as just empty quotes for now, since the ball just drops straight down at the start (not moving left/right yet)

Making the Ball Bounce

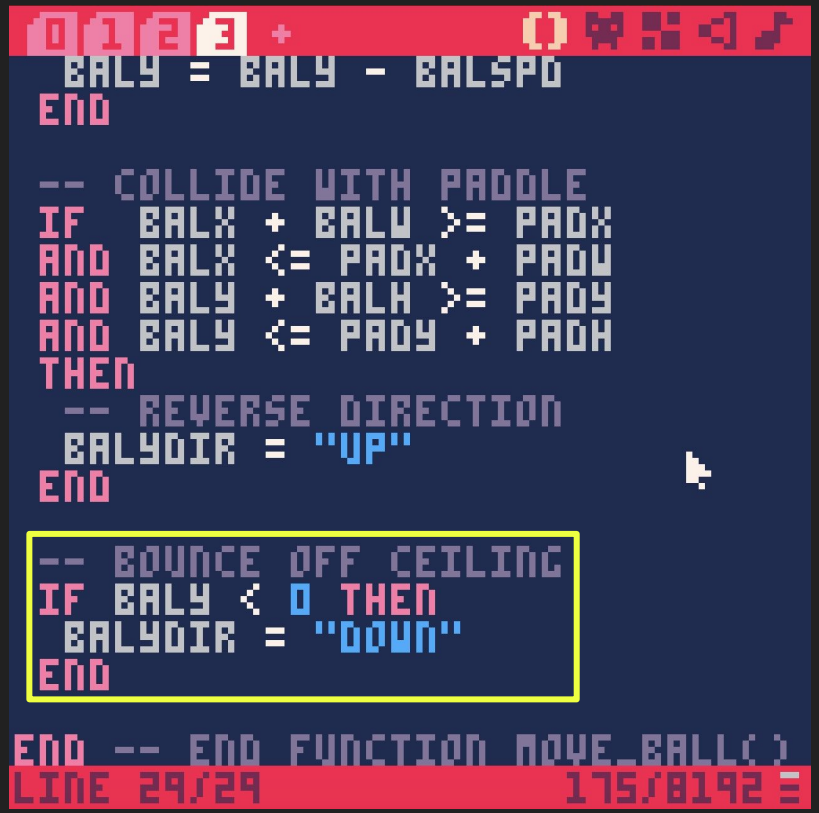
When the ball and paddle collide, I'll switch **BALYDIR** from **"DOWN"** to **"UP"**



Making the Ball Bounce

If the ball hits the ceiling, I'll switch its direction back to **"DOWN"** so it will bounce off the ceiling and start falling again

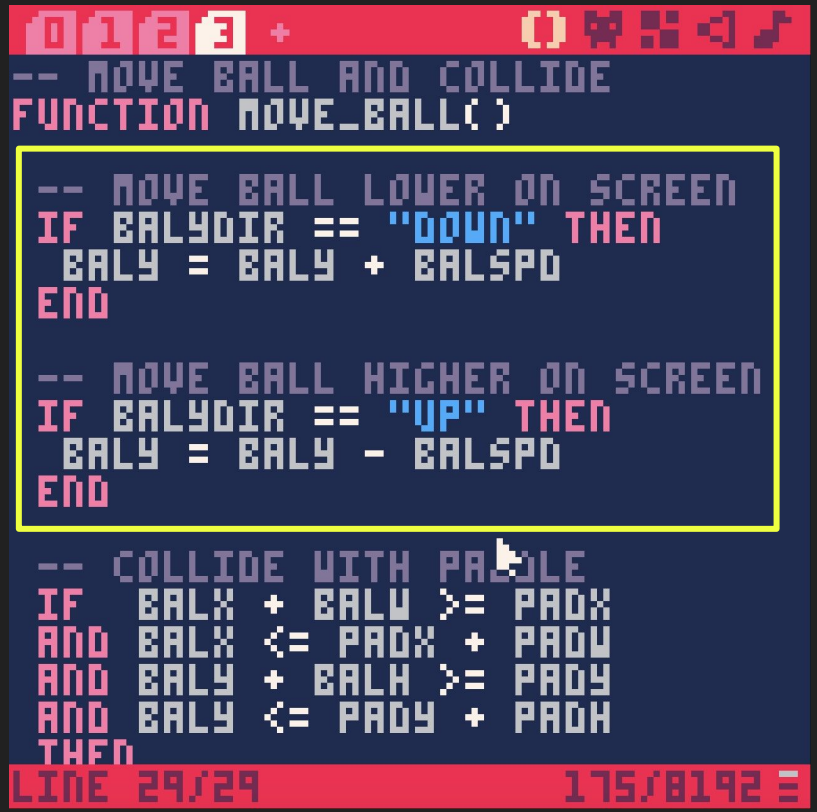
We know the ceiling is at **Y=0**, so we can compare the ball's y position against 0



Making the Ball Bounce

Finally, we can control which way the ball is moving (up or down) based on the value of **BALLYDIR**

Add the speed to **BALY** to move down; **subtract** to move **up**



Making the Ball Bounce Sideways

Download Example File: paddleball_basic_09_bounce.p8

Making the Ball Bounce Sideways

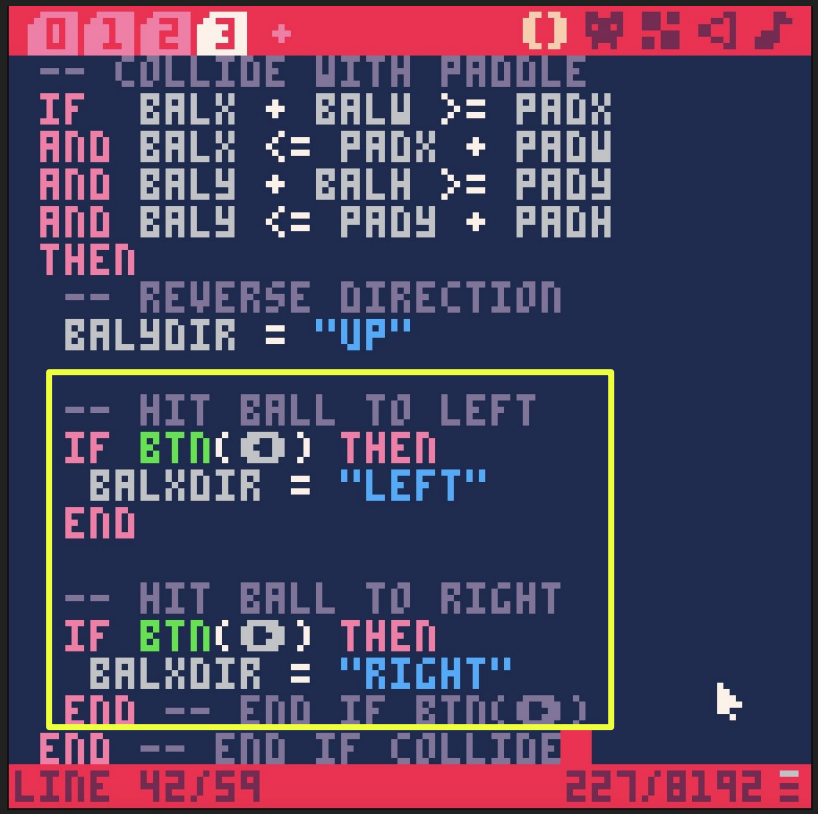
If the paddle is moving when it hits the ball, we want the ball to move in the same direction (**left** or **right**) *as well as* **up**

This means *we'll have to* ***combine if/then statements***

Making the Ball Bounce Sideways

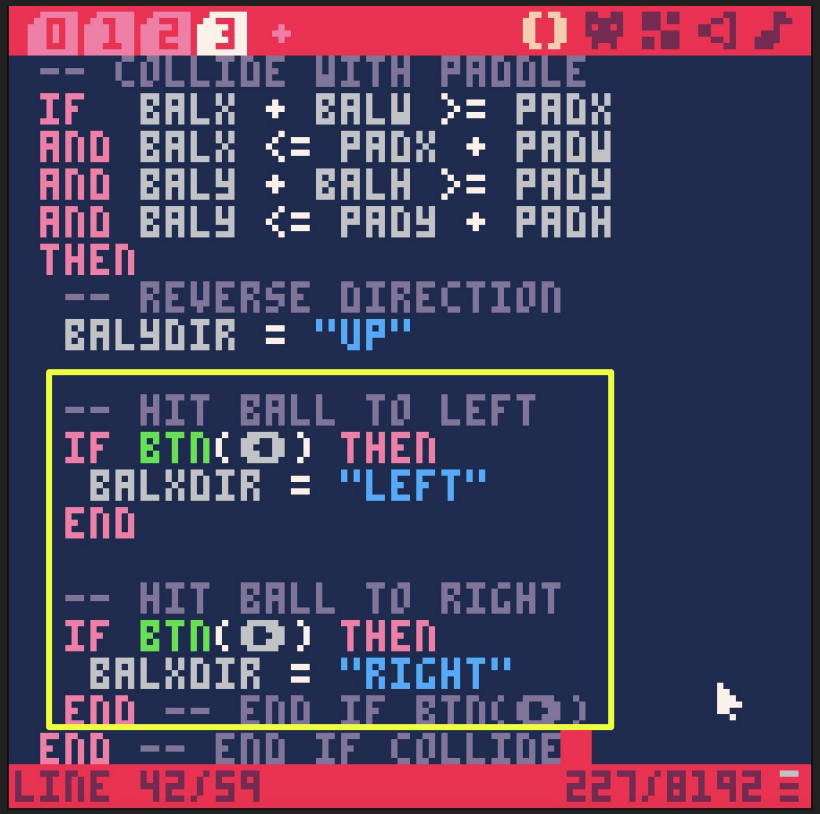
If the paddle is moving, that means a button is being pressed

So we can use **BTN()** for the left and right arrow keys as our condition to measure whether the paddle is moving



Making the Ball Bounce Sideways

If a button is being pressed, we'll set **BALXDIR** to **"LEFT"** or **"RIGHT"** accordingly



Making the Ball Bounce Sideways

The button-press IF/THEN code must be *nested* within the IF/THEN block for collision

Because we need **TWO conditions to be met** for the ball to bounce at an angle:

1) Collision with paddle
2) An arrow keypress

Making the Ball Bounce Sideways

Think of each IF/THEN block like slices of bread in a sandwich

An IF/THEN within an IF/THEN is like a double-decker sandwich

Why can’t we just use AND to stack conditions here?

Making the Ball Bounce Sideways

We can use **AND** to stack conditions when what happens when all conditions are met is the ***same***

But in this instance, if collision is occurring, we will then do 1 of 2 things: move the ball left or move it right

IF BTN(L)
THEN
MOVE
LEFT

COLLISION BTWN PADDLE & BALL

IF BTN(R)
THEN
MOVE
RIGHT

Think of this as a Venn diagram

Truth Tables

Another way to look at it

AND (Both conditions must be met)			
X	Y	OUTCOME	
FALSE	FALSE	**FALSE**	
FALSE	TRUE	**FALSE**	
TRUE	FALSE	**FALSE**	
TRUE	TRUE	**TRUE**	
AND: In Paddleball (2 Conditions Must be Met to Bounce at an Angle			
Collision	**Arrow Key Pressed**	**OUTCOME**	**RESULT**
FALSE	FALSE	**FALSE**	No action taken
FALSE	TRUE	**FALSE**	Paddle moves
TRUE	FALSE	**FALSE**	Ball bounces straight up
TRUE	TRUE	**TRUE**	Ball bounces at an angle

Making the Ball Bounce Sideways

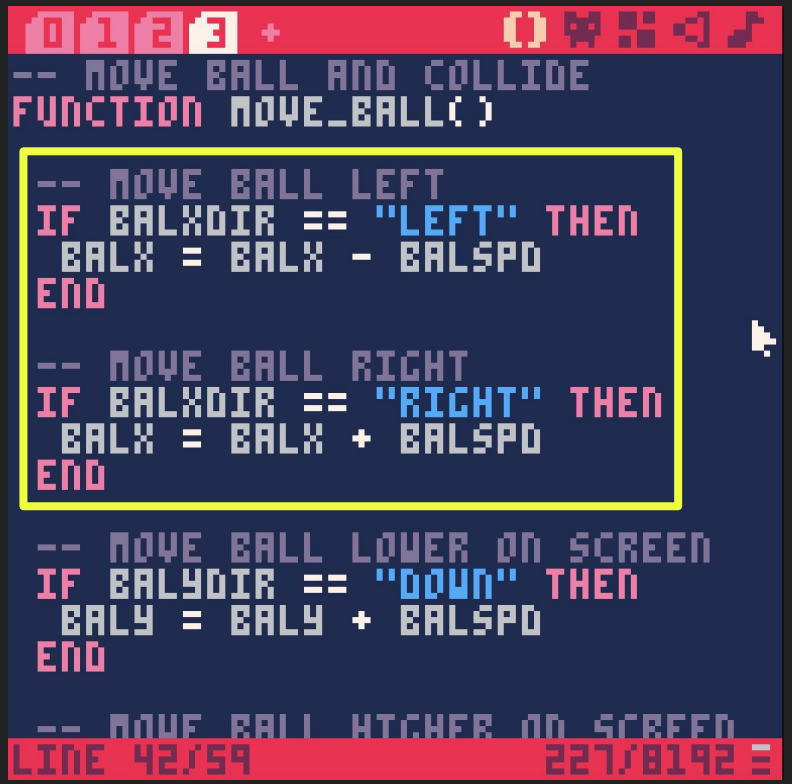
A common error is that the ball moves left/right whenever an arrow key is pressed, regardless of whether there's collision

If this is happening, you likely placed the BTN IF/THEN statements ***outside*** the collision IF/THEN block

Place these ***within*** the collision IF/THEN block instead >>

Making the Ball Bounce Sideways

Finally, we need to adjust the ball's **X** position based on its horizontal direction



Making the Ball Bounce Sideways

You can't see these cases all at once in PICO-8's editor, so I'm showing it in VS Code

There's an IF/THEN to define how the ball should move for each of the 4 directions

```
-- move ball left
if balxdir == "left" then
    balx = balx - balspd
end

-- move ball right
if balxdir == "right" then
    balx = balx + balspd
end

-- move ball lower on screen
if balydir == "down" then
    baly = baly + balspd
end

-- move ball higher on screen
if balydir == "up" then
    baly = baly - balspd
end
```

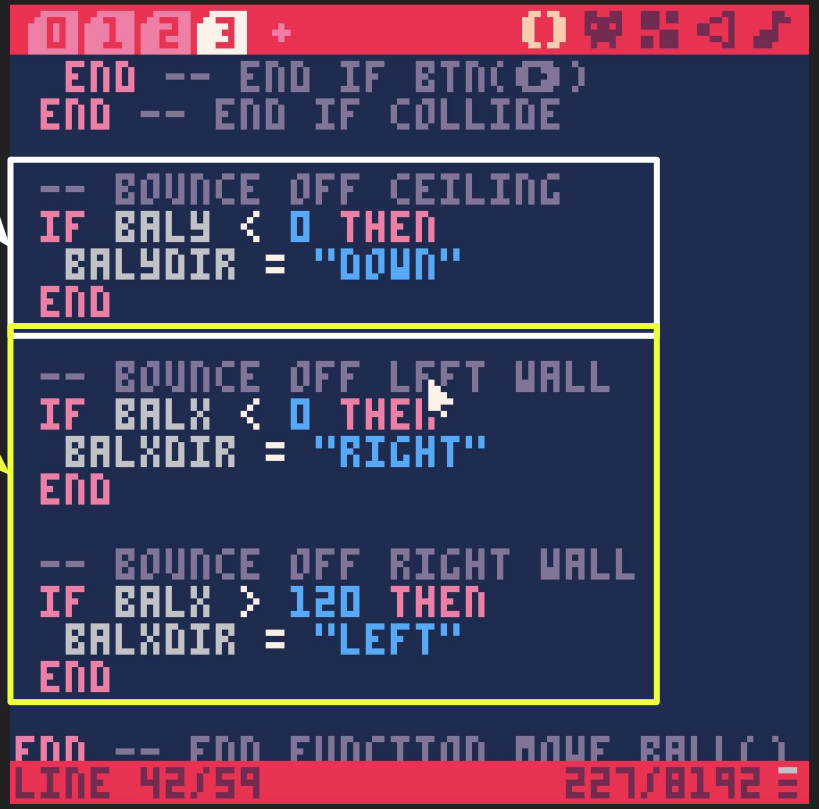
Bouncing off the Walls & Ceiling

Bouncing off the Walls and Ceiling

We already added this to make the ball switch direction when reaching the ceiling

We can do the same thing for the walls

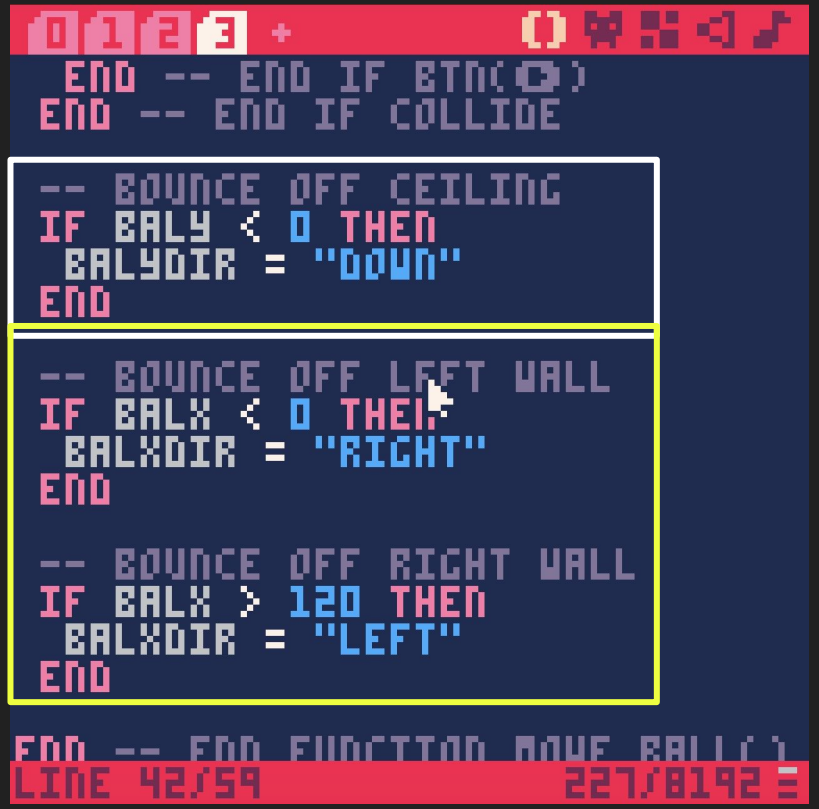
*A more complete solution would be to use **128 minus the ball's width** to check against the right edge of the screen*

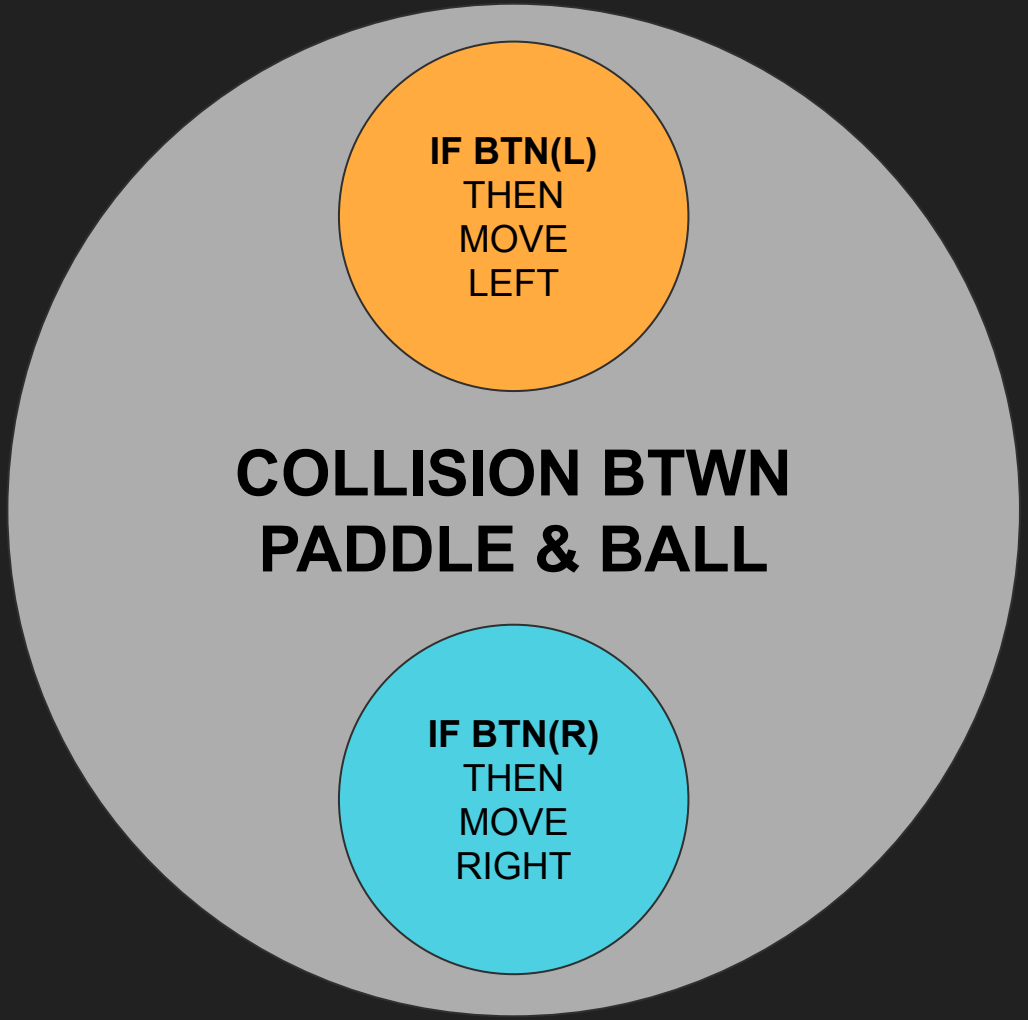


Bouncing off the Walls and Ceiling

Make sure to place these IF/THEN statements ***OUTSIDE*** the collision IF/THEN block

Otherwise the ball would only switch direction if BOTH collision is true and the other conditions are met





IF Y <=0
THEN
SWITCH
VERTICAL
DIRECTION

IF X <=0
THEN
SWITCH
HORIZONTAL
DIRECTION

IF X >=120
THEN
SWITCH
HORIZONTAL
DIRECTION

A Venn diagram of our IF/THEN logic so far

The entire move_ball() function so far:

Change X/Y based on direction

Collision and bouncing at an angle

Bouncing off walls and ceiling

```
-- move ball and collide
function move_ball()

    -- move ball left
    if balxdir == "left" then
        balx = balx - balspd
    end

    -- move ball right
    if balxdir == "right" then
        balx = balx + balspd
    end

    -- move ball lower on screen
    if balydir == "down" then
        baly = baly + balspd
    end

    -- move ball higher on screen
    if balydir == "up" then
        baly = baly - balspd
    end

    -- collide with paddle
    if  balx + balw >= padx
    and balx <= padx + padw
    and baly + balh >= pady
    and baly <= pady + padh
    then
        -- reverse direction
        balydir = "up"

        -- hit ball to left
        if btn(⬅️) then
            balxdir = "left"
        end

        -- hit ball to right
        if btn(➡️) then
            balxdir = "right"
        end

    end -- end if collide

    -- bounce off ceiling
    if baly < 0 then
        balydir = "down"
    end

    -- bounce off left wall
    if balx < 0 then
        balxdir = "right"
    end

    -- bounce off right wall
    if balx > 120 then
        balxdir = "left"
    end
```

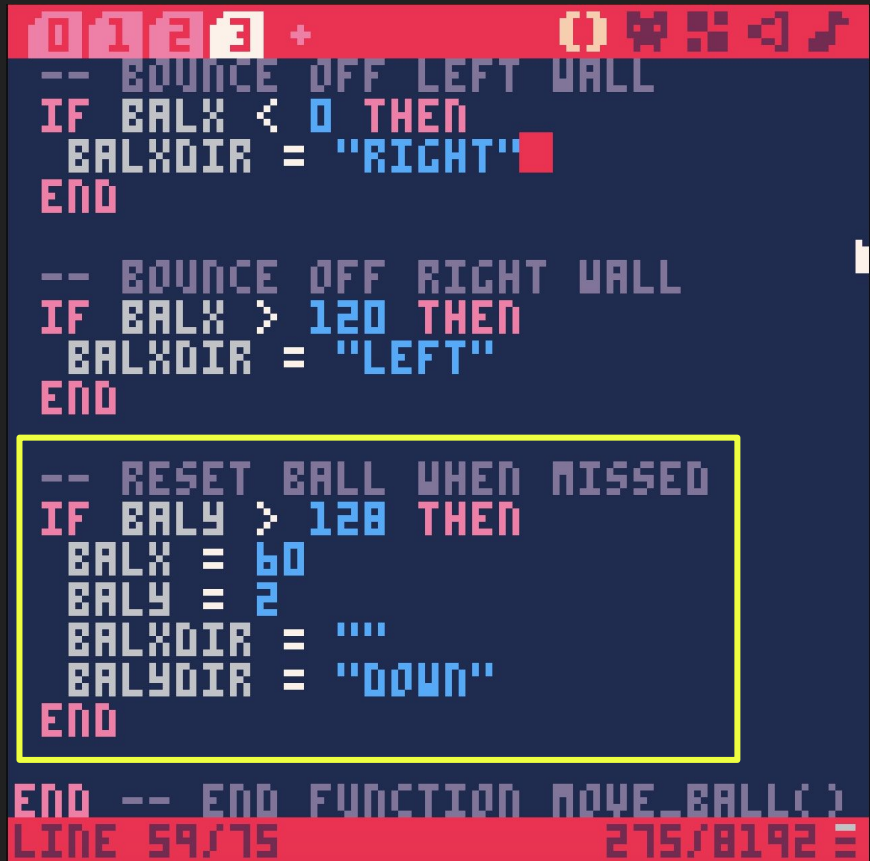
Resetting the Ball After a Miss

Resetting the Ball After a Miss

If the player misses the ball and it falls off the bottom of the screen, we need to move the ball back to its initial position

We also need to reset its direction

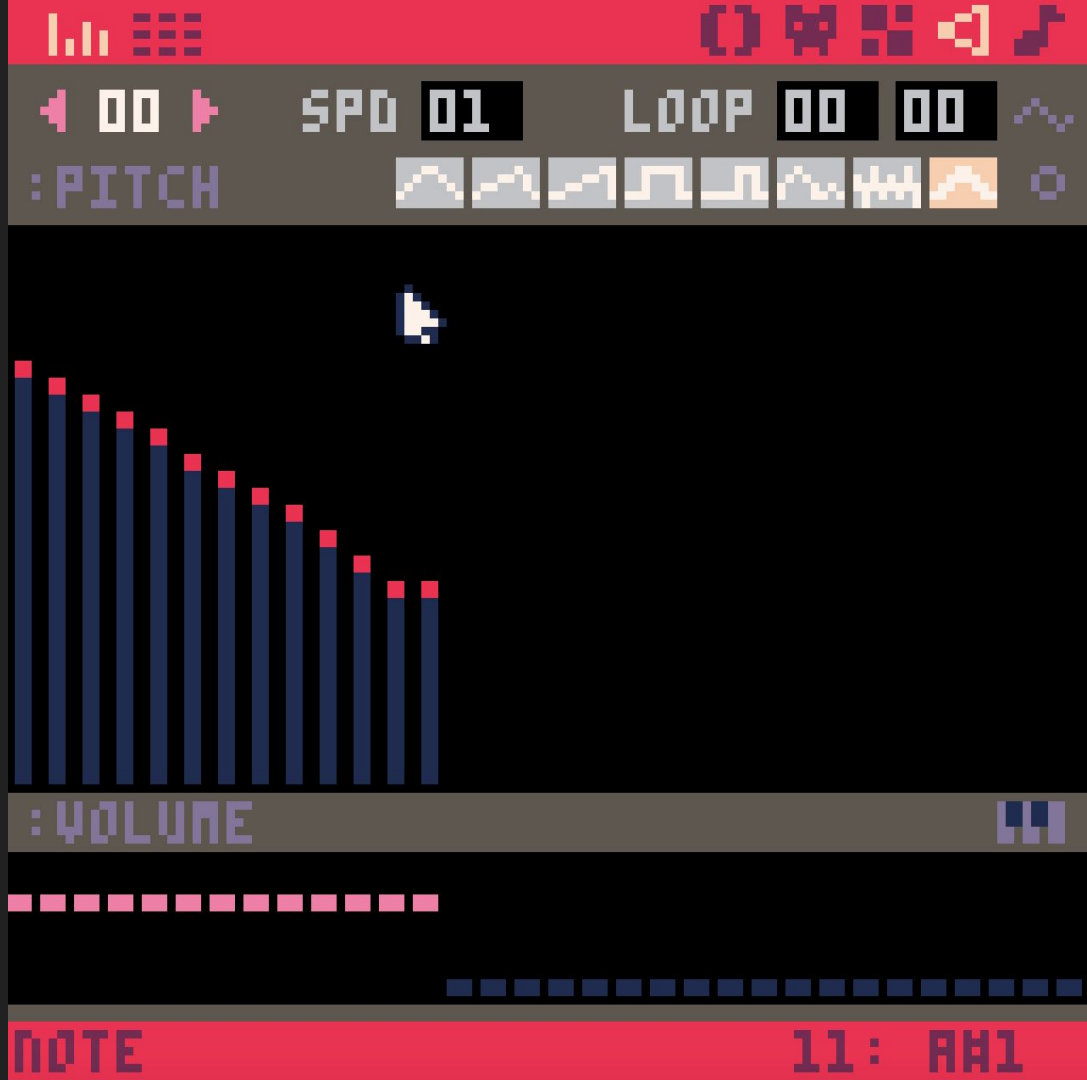
All the values being set here are the same as the initial values used in make_ball()



Adding Sounds

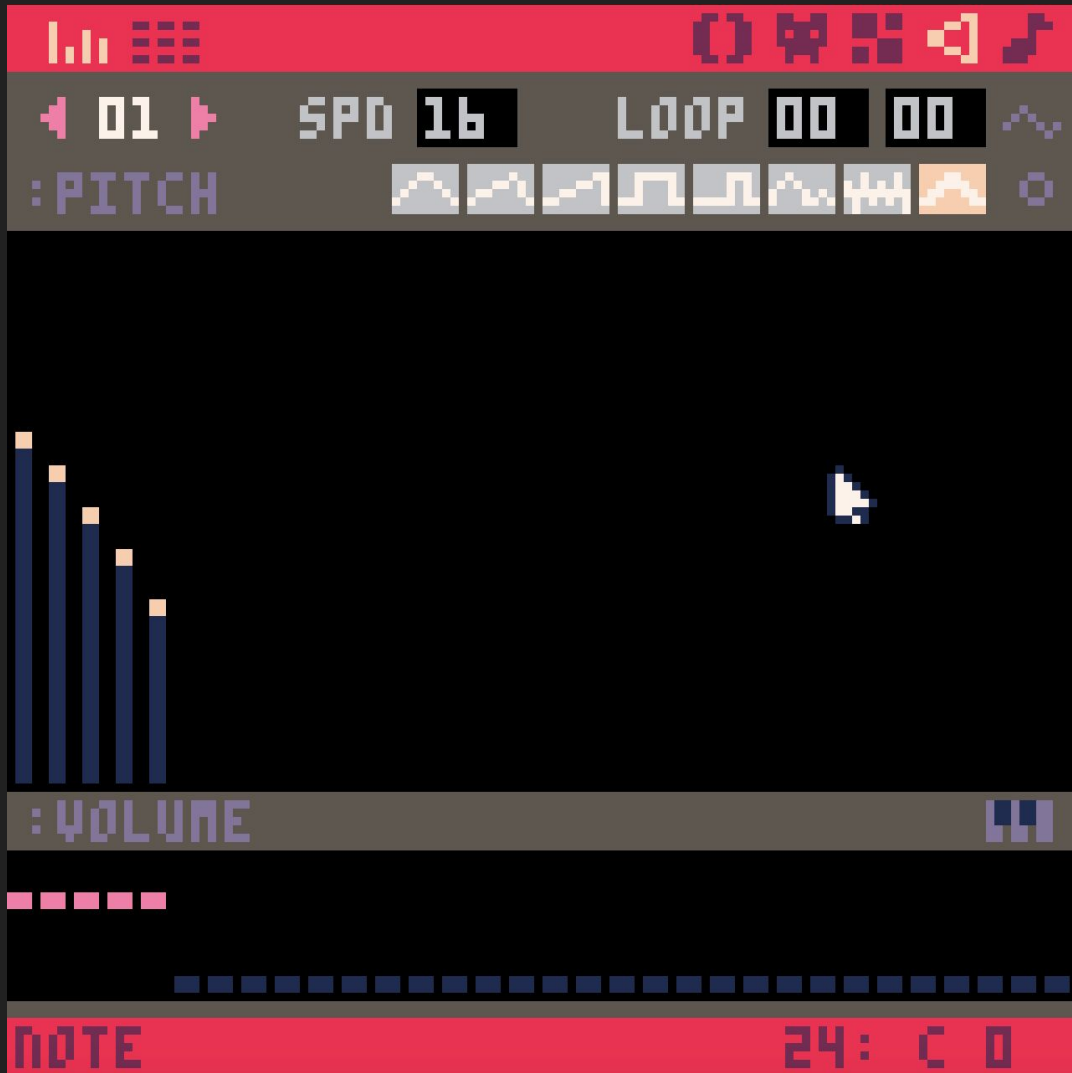
Adding Sounds

I created a cool squishy laser boinky sound for when the ball bounces off the paddle, walls, or ceiling



Adding Sounds

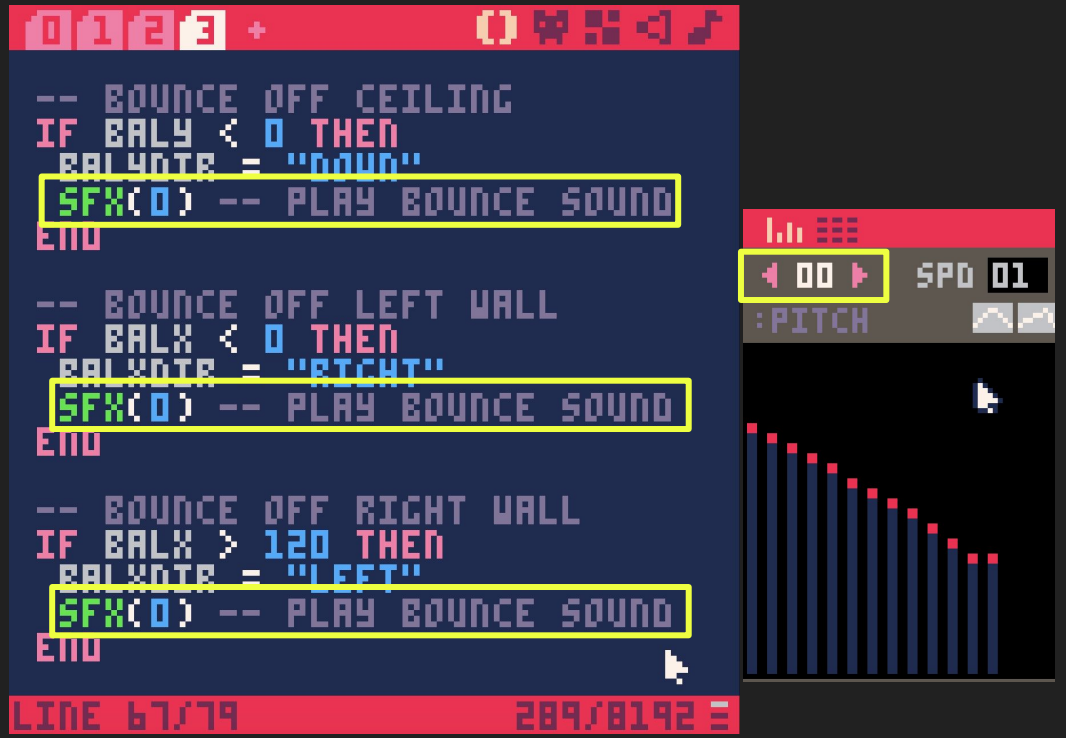
Also a sad chime for when the ball goes off the bottom of the screen



Adding Sounds

You can use the SFX() function, with the number for the sound inside the parentheses, to play a sound effect

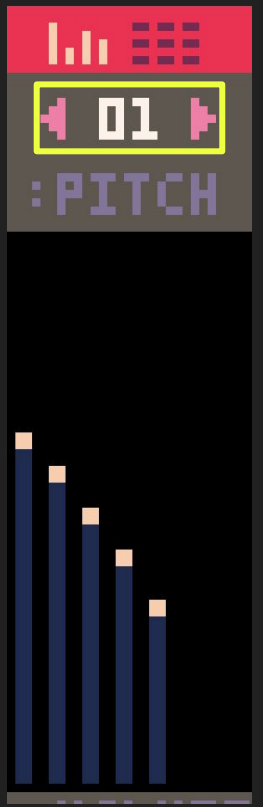
I'll add SFX(0) anywhere the ball bounces



Adding Sounds

You can use the SFX() function, with the number for the sound inside the parentheses, to play a sound effect

I'll add SFX(1) in the if/then block for handling a miss



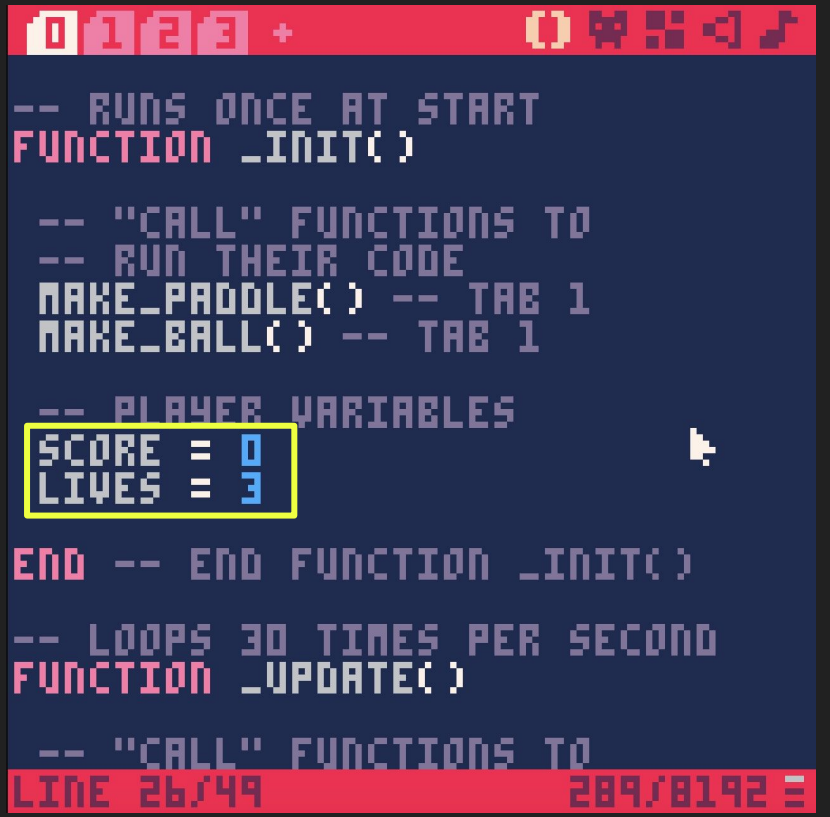
Adding Score & Lives

Download Example File: paddleball_basic_10_score_and_lives.p8

Adding Score and Lives

We'll need to introduce some new variables for the player's score and remaining lives

I'll do this in _init()



Adding Score and Lives

We can add a quick HUD by using the PRINT() function in _draw()

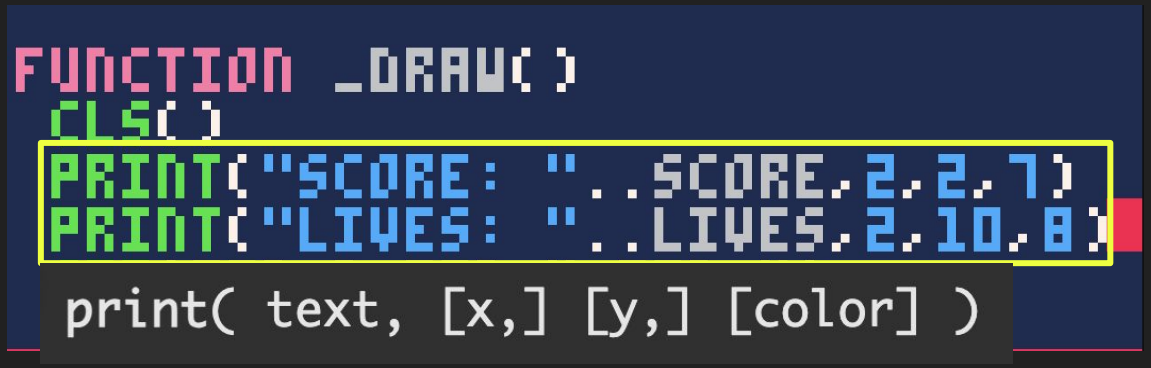
```
print( text, [x,] [y,] [color] )
```

Adding Score and Lives

- Remember, we can put plain text inside quotes
- But to combine text with a numeric value (the variable values for score, lives), we can put two periods . . between the text in quotes and the variable

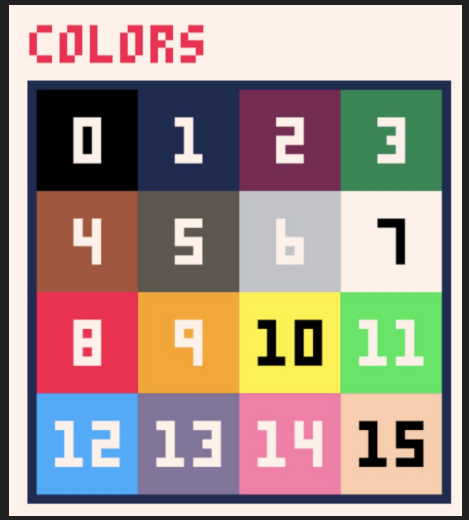
Text values are called "strings" in programming

Combining two "strings" together is called "concatenation"



Adding Score and Lives

- After the first value (the text to print), we can add values for the x and y coordinates of the text
- The last value is the numeric code for the color

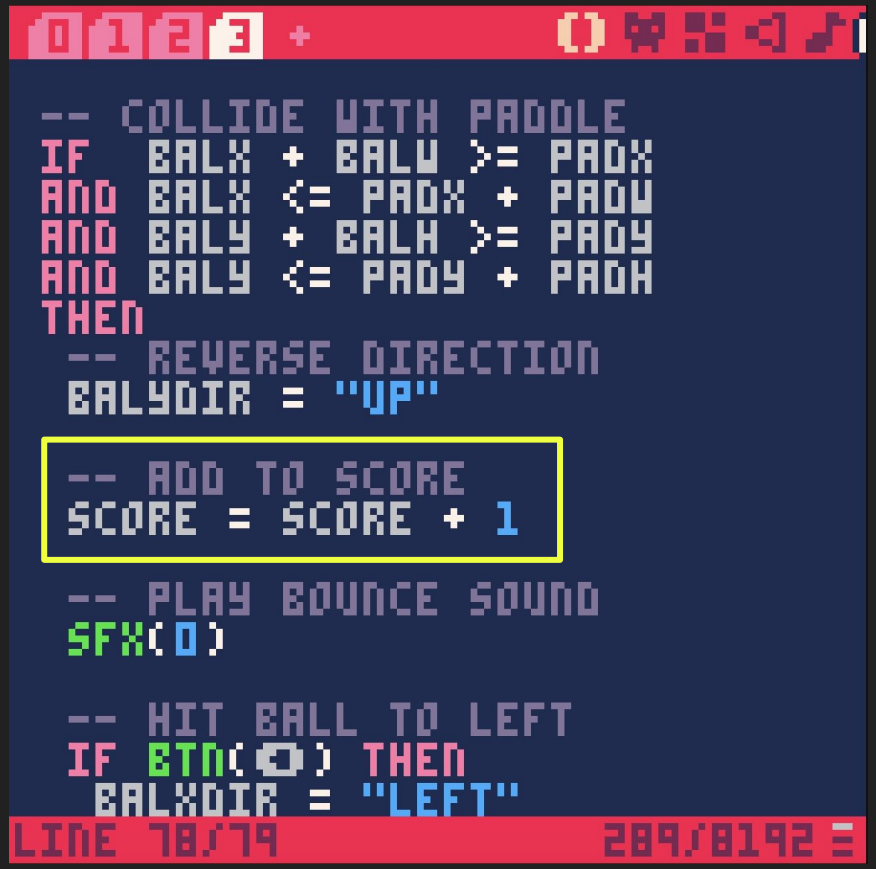



```
function _draw()
 cls()
 print("score: "..score,2,2,7)
 print("lives: "..lives,2,10,8)
```

print(text, [x,] [y,] [color])

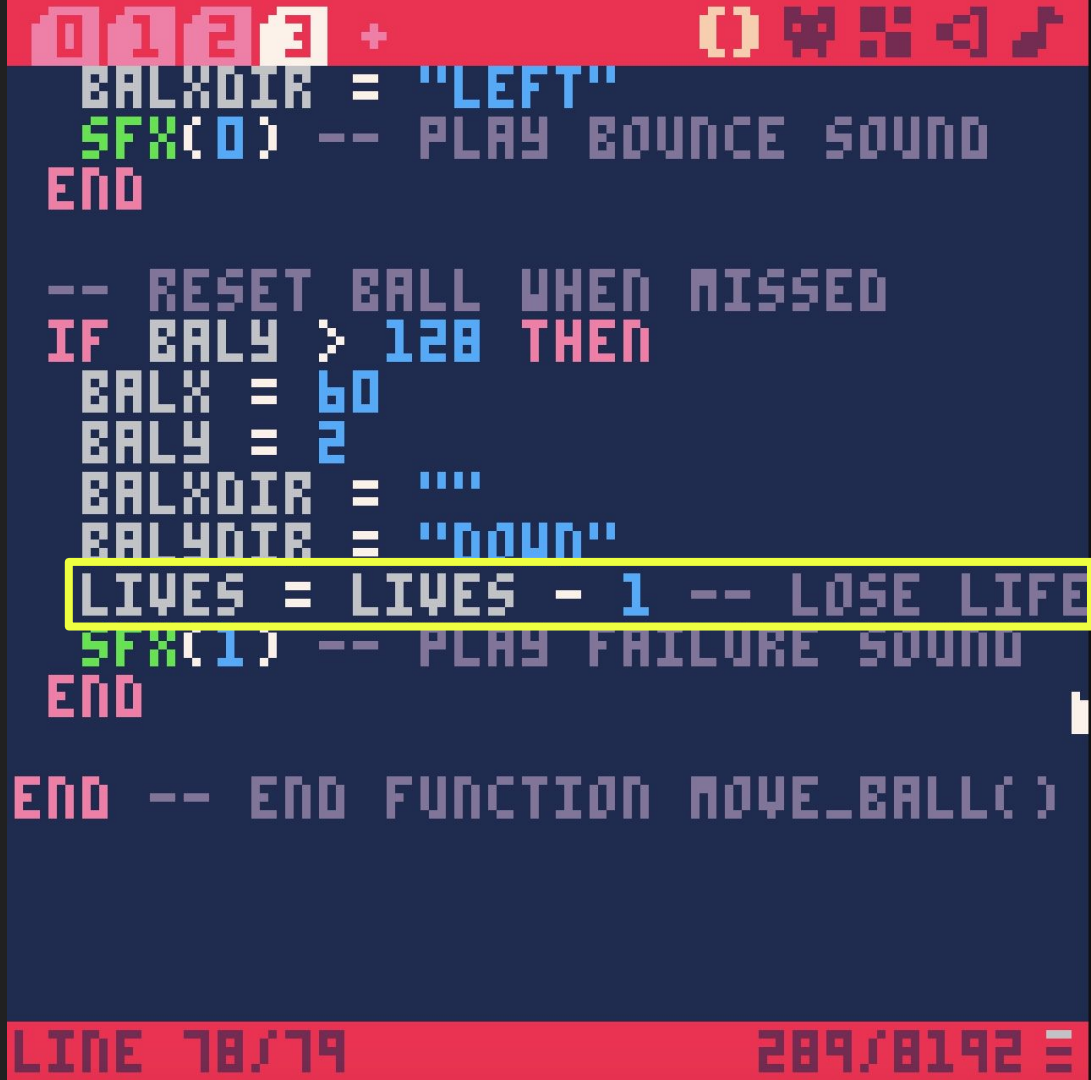
Adding Score

To change the score when the player hits the ball, we'll add to the score variable's value when collision between the ball and paddle is true, in the move_ball() function



Managing Lives

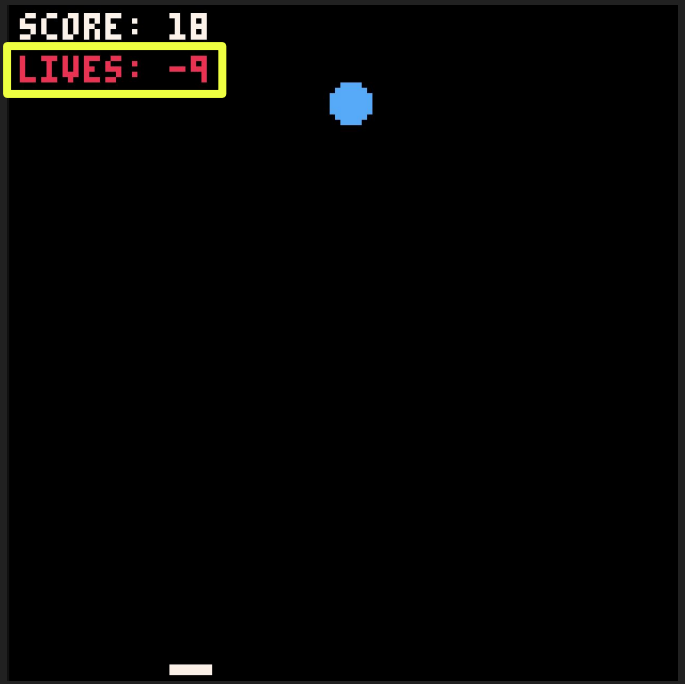
Inside the **IF/THEN** block for resetting the ball, we can subtract a life when the ball goes off screen



Managing Lives

But look what happens if we keep losing lives . . .

The value turns negative, but we can still keep playing



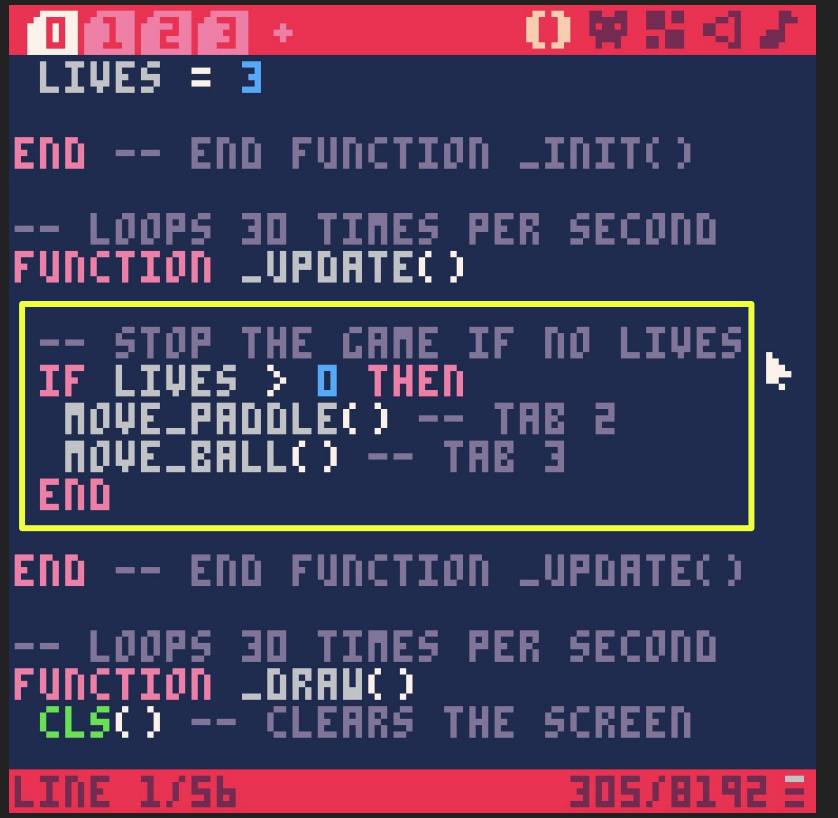
Implementing a Game Over Screen

Download Example File: paddleball_basic_11_gameover.p8

Implementing a Game Over Screen

If the player's lives reach zero, we can stop the game

We can approach this backwards, by *only* running the game as long as lives are *above* zero



Implementing a Game Over Screen

Likewise, we can use **IF/THEN** statements in our **_DRAW()** loop to display a game over message

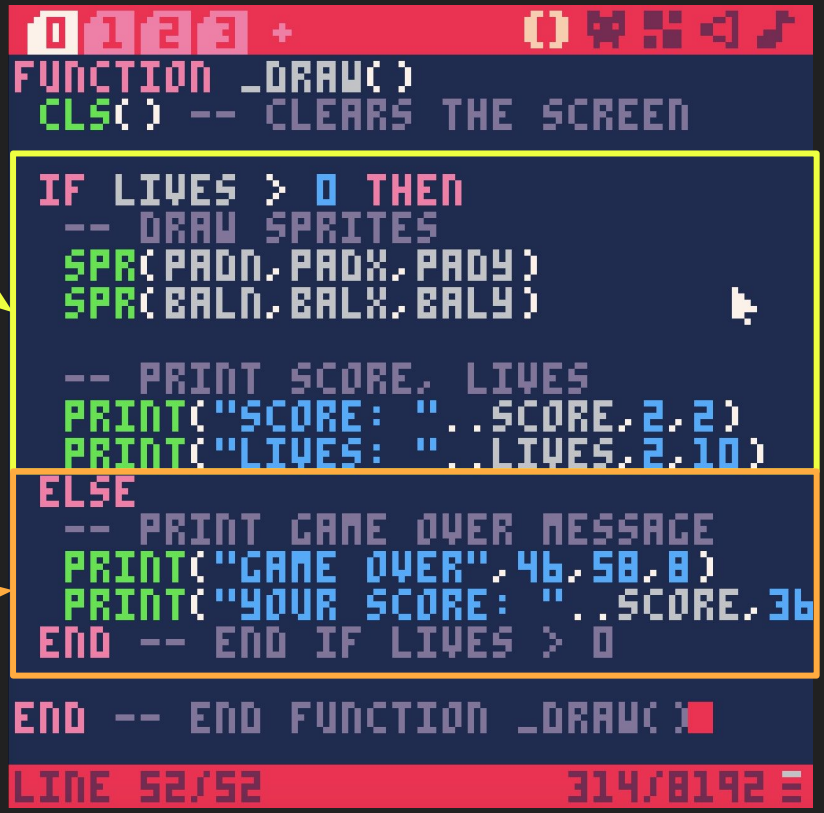
*There's **no text alignment feature** in PICO-8, so you have to find the right coordinates by trial & error*


```
SPR(PADN,PADX,PADY)

-- DRAW BALL SPRITE
SPR(BALN,BALX,BALY)

-- PRINT SCORE, LIVES
-- PLACE TEXT IN QUOTES
-- COMBINE TEXT WITH NUMBERS
-- BY TYPING .. BETWEEN THEM
PRINT("SCORE: "..SCORE,2,2)
PRINT("LIVES: "..LIVES,2,10)

-- SHOW GAME OVER MESSAGE
IF LIVES < 1 THEN
 PRINT("GAME OVER",46,64,8)
END

END -- END FUNCTION _DRAW()
```

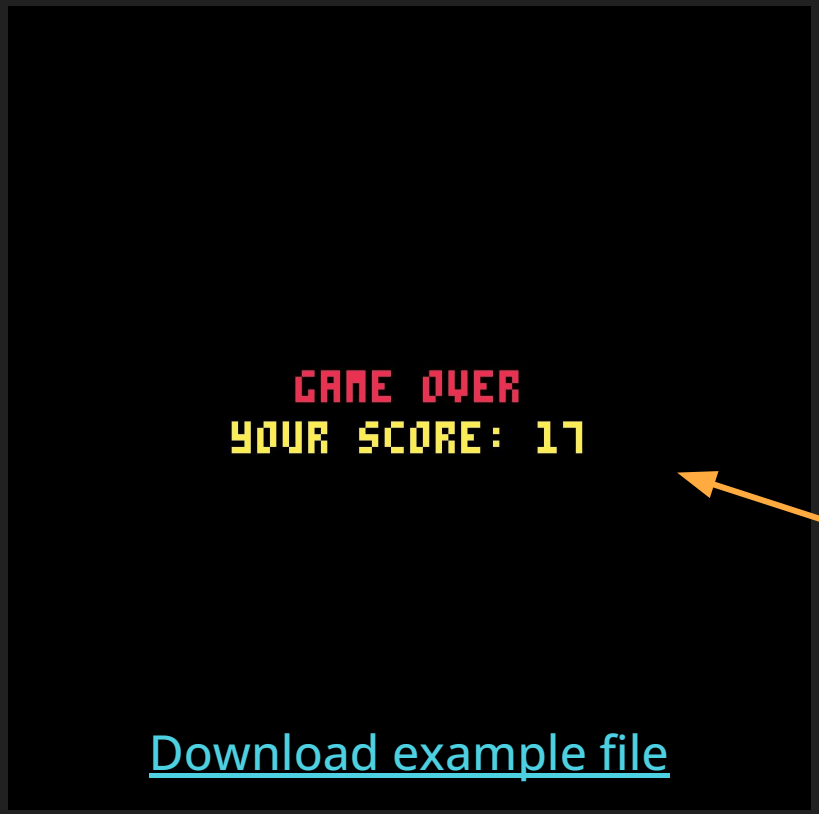
Implementing a Game Over Screen

If you want the game to disappear and be replaced by a game over screen, you can require that the player have lives left to draw the sprites and HUD

Use ELSE to display a game over screen when lives reach zero



Implementing a Game Over Screen



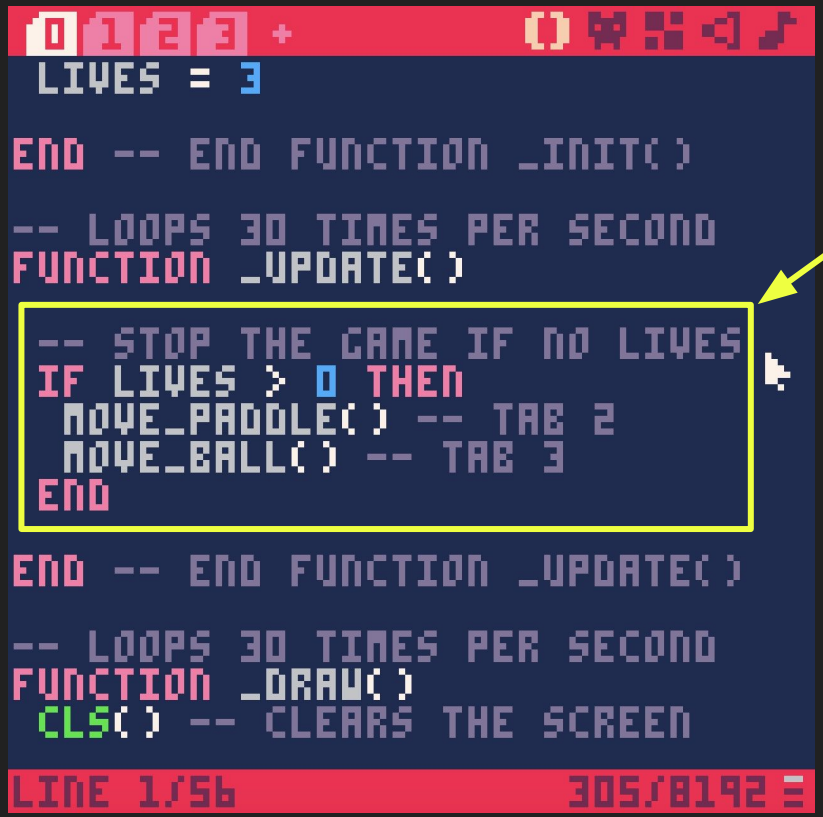
Download example file

```
FUNCTION _DRAW()
 CLS() -- CLEARS THE SCREEN

 IF LIVES > 0 THEN
  -- DRAW SPRITES
  SPR(PADN,PADX,PADY)
  SPR(BALN,BALX,BALY)

  -- PRINT SCORE, LIVES
  PRINT("SCORE: "..SCORE,2,2)
  PRINT("LIVES: "..LIVES,2,10)
 ELSE
  -- PRINT GAME OVER MESSAGE
  PRINT("GAME OVER",46,58,8)
  PRINT("YOUR SCORE: "..SCORE,36
 END -- END IF LIVES > 0

END -- END FUNCTION _DRAW()
```

Restarting the Game

Download Example File: paddleball_basic_12_restart.p8

Restarting the Game



In **_UPDATE()**, we are already stopping the game when lives reach zero

But we should also give the player a way to restart the game after getting a game over

We can add an **ELSE** case to this IF/THEN block to specify what inputs the player can perform after a game over – if they press the **X** key, we can call the **_INIT()** function, which *reverts the game to its starting conditions*

```
 LIVES = 3

END -- END FUNCTION _INIT()

-- LOOPS 30 TIMES PER SECOND
FUNCTION _UPDATE()

 -- STOP THE GAME IF NO LIVES
 IF LIVES > 0 THEN
  MOVE_PADDLE() -- TAB 2
  MOVE_BALL() -- TAB 3
 END

END -- END FUNCTION _UPDATE()

-- LOOPS 30 TIMES PER SECOND
FUNCTION _DRAW()
 CLS() -- CLEARS THE SCREEN
```

LINE 1/56 305/8192

```
END -- END FUNCTION _INIT()

-- LOOPS 30 TIMES PER SECOND
FUNCTION _UPDATE()

 -- STOP THE GAME IF NO LIVES
 IF LIVES > 0 THEN
  MOVE_PADDLE() -- TAB 2
  MOVE_BALL() -- TAB 3
 ELSE

  -- PRESS ❎ TO RESTART
  IF BTNP(❎) THEN
   _INIT()
  END -- END IF BTNP(❎)

 END -- END IF/ELSE LIVES > 0

END -- END FUNCTION _UPDATE()
```

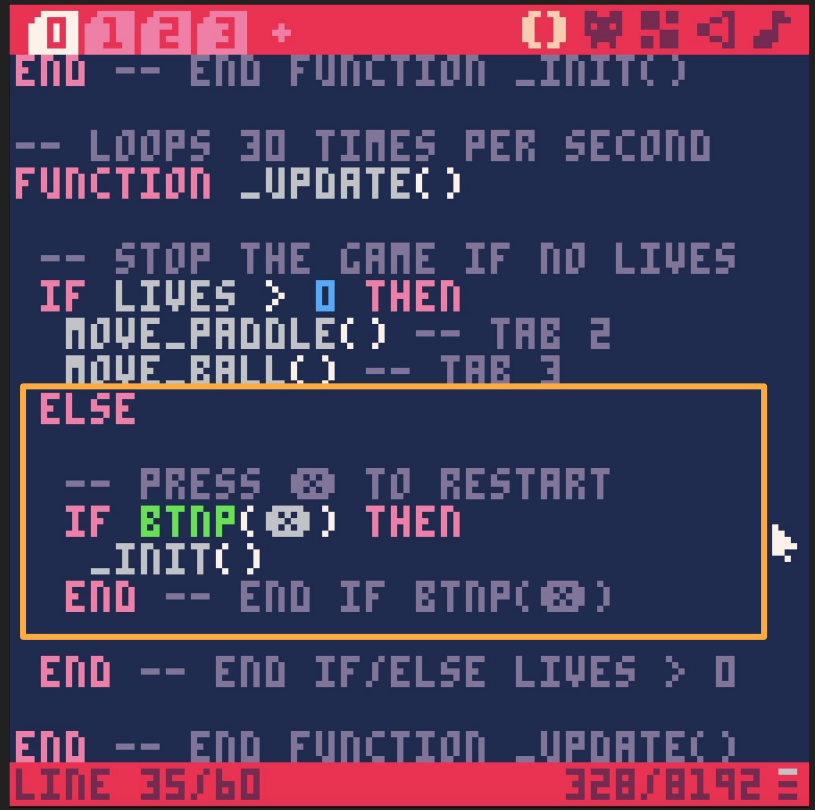
LINE 35/60 328/8192

Restarting the Game

BTNP() is a function similar to **BTN()**

But while **BTN()** requires the player to press and hold a key, **BTNP()** will trigger as long as a key is pressed

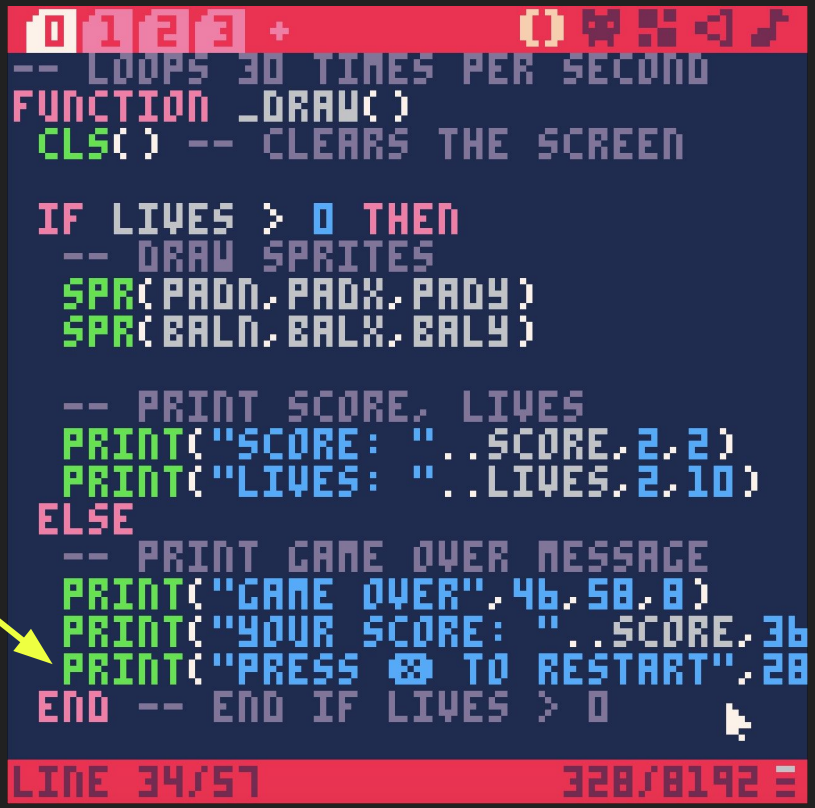
Type **Shift X** in the PICO-8 editor to generate an X-key symbol – ***or*** use the key code **5**, as in **BTNP(5)**



Restarting the Game

Finally, we should provide instructions to the player that they can press X to restart from the game over screen

I'll add a **PRINT()** statement in **_DRAW()** after displaying the game over message and score



Restarting the Game


```
-- LOOPS 30 TIMES PER SECOND
FUNCTION _DRAW()
 CLS() -- CLEARS THE SCREEN

 IF LIVES > 0 THEN
  -- DRAW SPRITES
  SPR(PADN,PADX,PADY)
  SPR(BALN,BALX,BALY)

  -- PRINT SCORE, LIVES
  PRINT("SCORE: "..SCORE,2,2)
  PRINT("LIVES: "..LIVES,2,10)
 ELSE
  -- PRINT GAME OVER MESSAGE
  PRINT("GAME OVER",46,50,8)
  PRINT("YOUR SCORE: "..SCORE,36
  PRINT("PRESS ❎ TO RESTART",28
 END -- END IF LIVES > 0
```

The complete PRINT() statement:

```
PRINT("PRESS ❎ TO RESTART",28,74,7)
```

New Game Plus: Paddleball w/Physics

Download Example Files: _helloworld_01a_paddleball_physics.zip

Learning Resources

PICO-8 Resources

- PICO-8 Home
- Official Resources
- Cheat Sheet
- Forum
- Games

My GitHub Repository, helloworld

Code Examples

- basics_00_helloworld.p8
- basics_01_shapes.p8
- basics_02_gameloop.p8
- basics_03_variables.p8
- basics_04_coordinates.p8

Code Examples

1. paddleball_basic_01_sprites.p8
2. paddleball_basic_02_variables.p8
3. paddleball_basic_03_variables2.p8
4. paddleball_basic_04_input_and_movement.p8
5. paddleball_basic_05_conditional_logic.p8
6. paddleball_basic_06_functions.p8
7. paddleball_basic_07_collision.p8
8. paddleball_basic_08_reverse_direction.p8
9. paddleball_basic_09_bounce.p8
10. paddleball_basic_10_score_and_lives.p8
11. paddleball_basic_11_gameover.p8 | paddleball_basic_11a_gameover_screen.p8
12. paddleball_basic_12_restart.p8

Download all paddleball examples: _helloworld_01_paddleball_basic.zip

PICO-8 Function Reference

PICO-8 Wiki – Functions Reference

- _init() Define starting conditions for your game
- _update() For input, calculations, and updating variable values
- _draw() For displaying text and graphics

- cls() Clears the screen
- spr() Draws a sprite to the screen
- print() Prints text to the screen
- btn() Checks whether a button is being held down
- btnp() Checks whether a button was pressed
- sfx() Plays a sound effect

</lesson>

Questions? Email me at mdimatteo@rider.edu anytime!