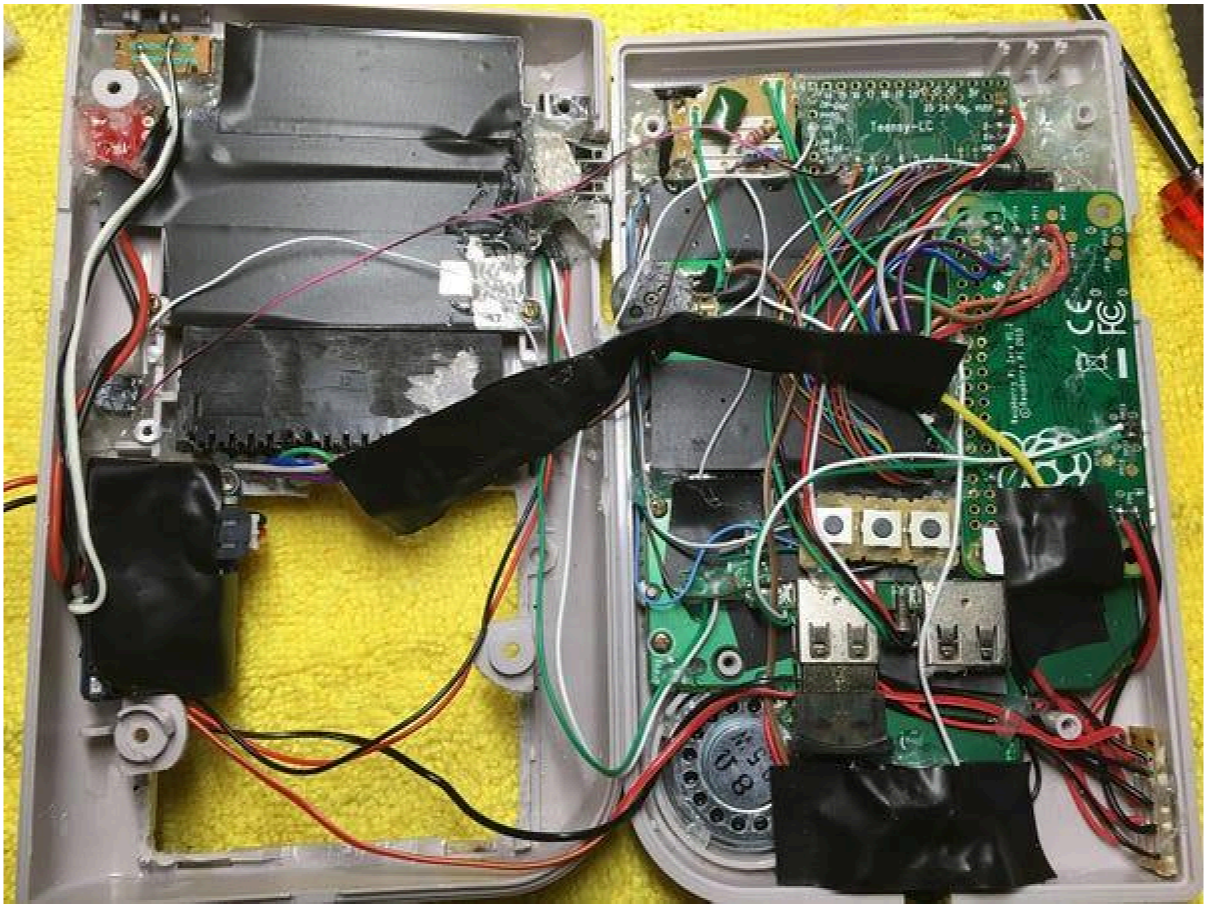
Coding Tutorial: Build a Raspberry Pi Game Boy–Style Handheld Console


RETROFLAG™
GPi CASE
RETROFLAG GPi CASE 2W
SELECT
START


Teensy-LC


SONIC
THE HEDGEHOG
2
2 PLAYER VS © 1992 SEGA


ATTENTION
KS - 1000
Contents
Static Shield Bag
Static
Sensitive
DOT MATRIX WITH STEREO SOUND
RETROFLAG GPi CASE 2W
SELECT
START
B
A
GCASE
COMPATIBLE WITH RASPBERRY PI ZERO, Z
PHONES

In this tutorial, you will build a **portable Game Boy–style retro gaming console using a Raspberry Pi Zero 2 W**. The project combines hardware assembly, Linux configuration, retro-game emulation, and Python programming.

The easiest version for a first build is the **Retroflag GPi Case 2W**, because it provides the screen, buttons, speaker, rechargeable battery, and Game Boy-style enclosure rather than requiring you to design all of those circuits yourself.

The Raspberry Pi Zero 2 W is particularly well suited to this project because it is only **65 × 30 mm**, has a quad-core 1 GHz CPU, 512 MB RAM, Wi-Fi, Bluetooth, and a microSD slot. Raspberry Pi currently lists it as remaining in production until at least January 2030. (Raspberry Pi)

---

1. What You Are Building

The finished system will look approximately like this:


DOT MATRIX WITH STEREO SOUND
GPi ZERO
2W
2023 Sliver X
2023 Retropie Project
RETROFLAG GPi CASE 2W
SELECT
START
Y
X
B
A


RETROFLAG GPi CASE
SELECT
START


SCORE 200
TIME 0:19
RINGS 22
SONIC
X 3
SELECT
START
X
Y
A
B

The architecture is:

```
┌──────────────────────────────┐
│  Raspberry Pi Zero  │
│       2 W          │
│                 │
│  CPU + RAM + Wi-Fi  │
└──────────────┬───────────────┘
       │
  microSD / Linux
       │
┌──────────────▼───────────────┐
│  Recalbox /        │
│  Retro Gaming OS   │
└──────────────┬───────────────┘
       │
┌───────────────────┼──────────────────────┐
│         │         │
┌───▼───┐ ┌───▼───┐ ┌───▼───┐
│ LCD  │ │Buttons│ │Audio │
│Screen │ │D-Pad │ │Speaker│
└───────┘ └───────┘ └───────┘
```

```
        |
┌───────▼───────┐
│  Battery    │
│  & Power    │
└───────────────┘
```

---

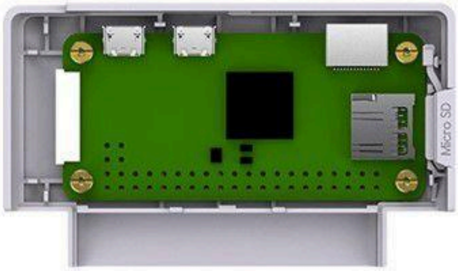
2. Parts You Need

Recommended beginner configuration

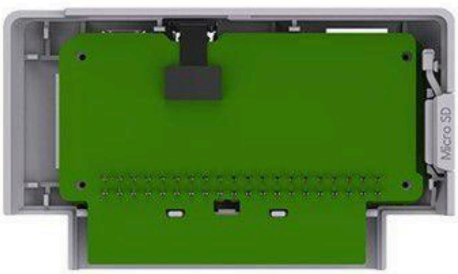
Component	Purpose
Raspberry Pi Zero 2 W	Main computer
Retroflag GPi Case 2W	Case, display, controls and power system
microSD card, 32 GB or larger	Operating system and games
Computer with microSD reader	Install the software
USB/micro-USB cable as appropriate	Initial setup
Wi-Fi network	Transfer software/games
Legal game files	Games you have the right to use

The GPi Case 2W is designed specifically for the Raspberry Pi Zero family and supports the **Zero, Zero W and Zero 2 W**. It has a **3-inch 640×480 IPS display, 2800 mAh rechargeable lithium-ion battery, speaker, 3.5-mm audio output, Game Boy-style controls, power switch, volume controls and shoulder buttons**. (RetroFlag)


DOT MATRIX WITH STEREO SOUND
RETROFLAG GPi CASE 2W
SELECT
START
2019-7-22



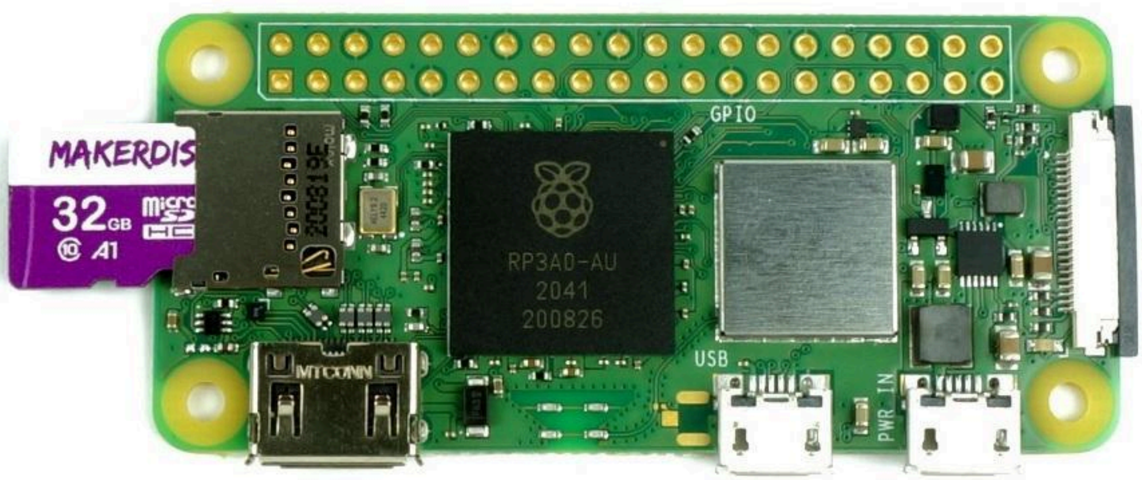
1. Install Raspberry
Pi Zero

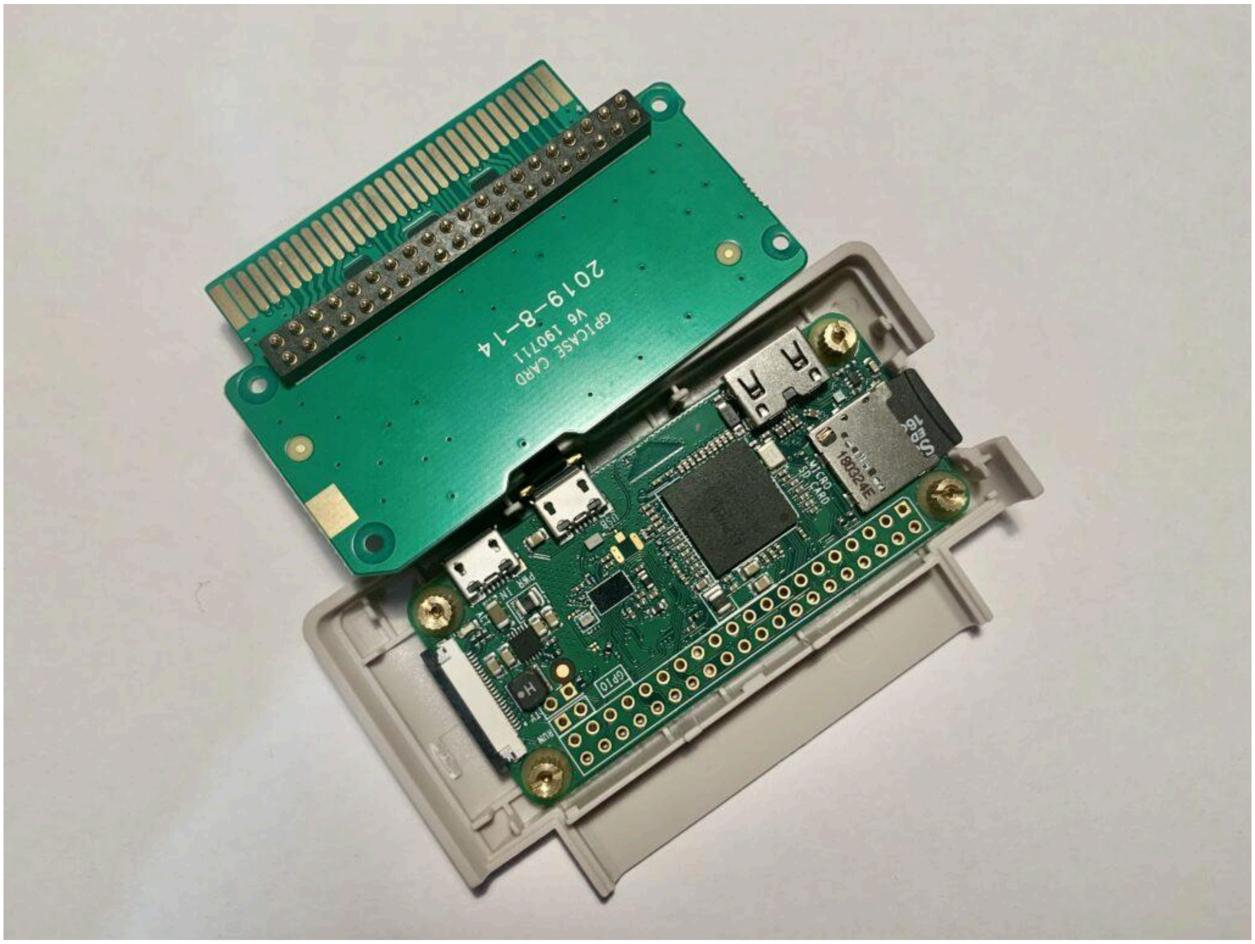


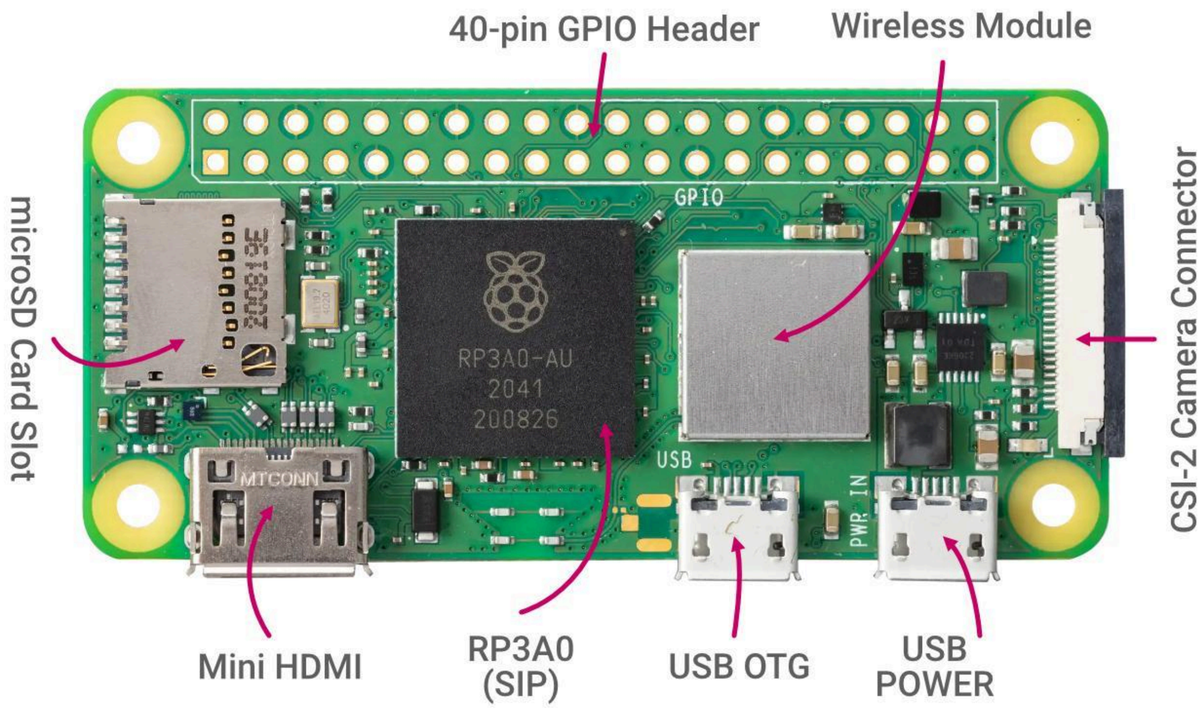
2. Install the IO
Extension Board

3. Install the back
cover

4. Insert SD Card




2019-8-14
V6 190711
GPICASE CARD



Where to order

For the computer, start with the official Raspberry Pi product page:

Raspberry Pi Zero 2 W official product page

For the enclosure:

Retroflag GPi Case 2W official product page

The Raspberry Pi documentation also explains that the Pi requires boot media such as a microSD card and recommends using Raspberry Pi Imager to prepare it. (Raspberry Pi)

---

3. Why Use a GPi Case Instead of Building Every Circuit?

You could build everything from individual components:

```
Pi Zero 2 W
     │
     ├── LCD
     ├── D-pad
     ├── A/B buttons
     ├── X/Y buttons
     ├── Start/Select
     ├── Speaker
     ├── Audio amplifier
     ├── Battery
     ├── Battery charger
     ├── Power switch
     └── Safe-shutdown circuit
```

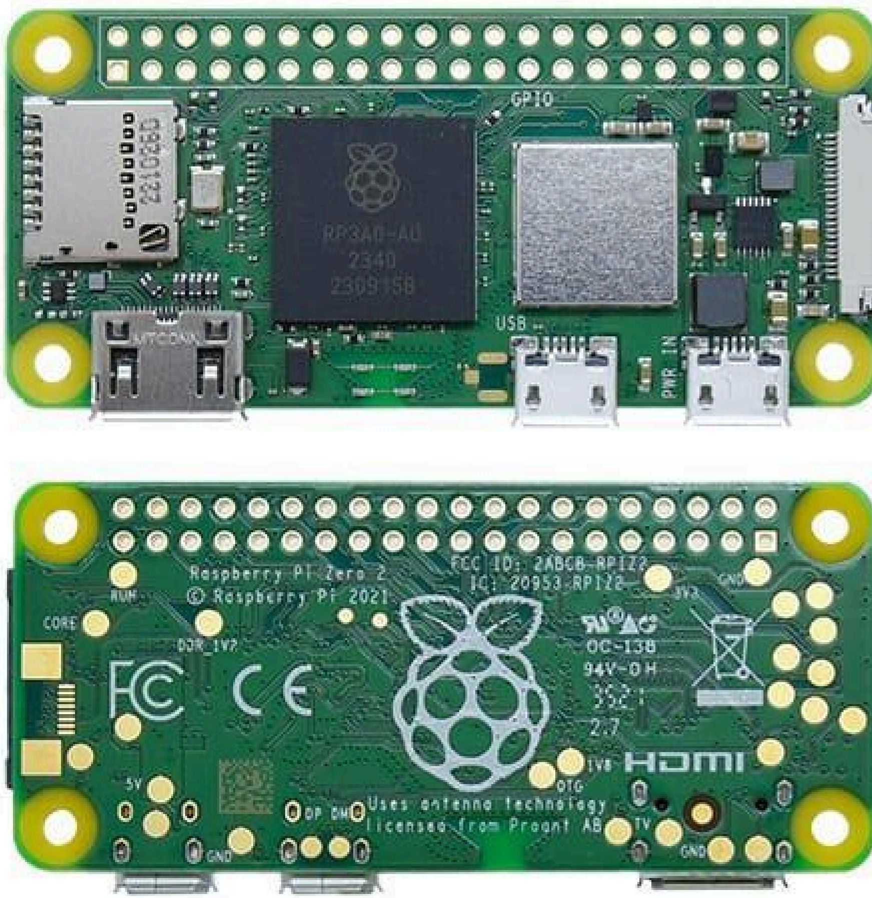
That is a much more advanced electronics project.

The GPi Case 2W combines many of those components into one enclosure. This makes it an excellent **first Raspberry Pi handheld project** while still allowing you to learn Linux, Python, GPIO, emulation and embedded programming.

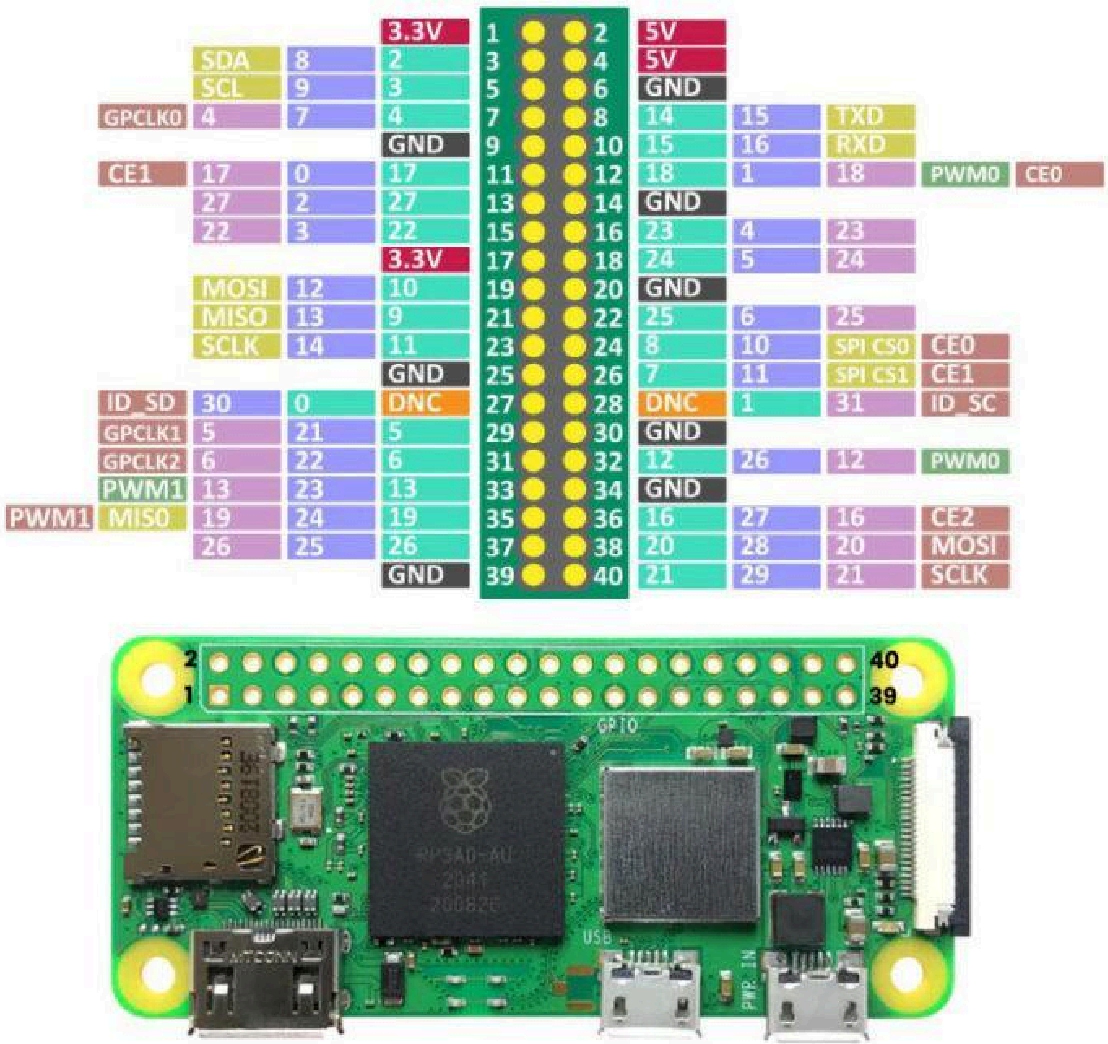
---

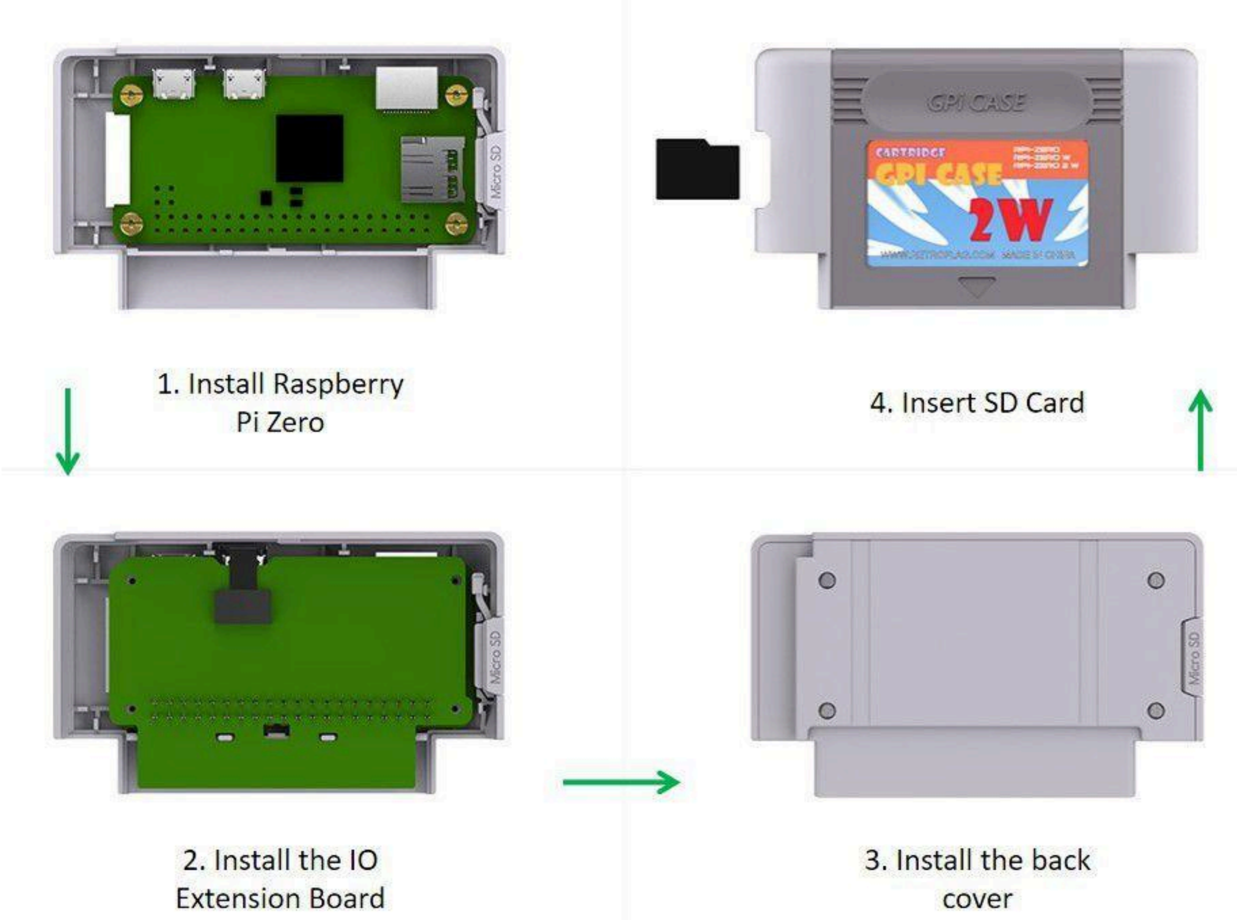
4. Assemble the Raspberry Pi

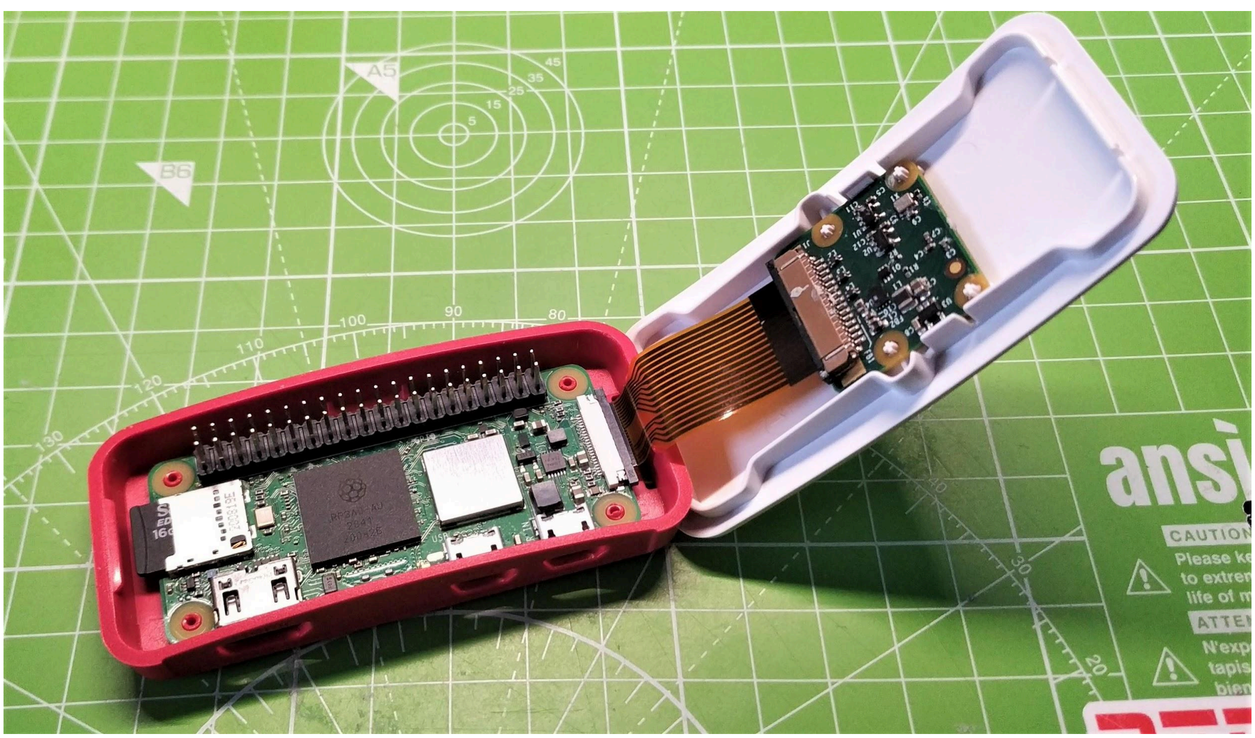
Before installing the Pi into the case, inspect the board.


GPIO
USB
PWR IN
Raspberry Pi Zero 2
© Raspberry Pi 2021
FCC ID: 2ABCB-RPIZ2
IC: 20953-RPIZ2
HDMI
Uses antenna technology
licensed from Proant AB

PINOUT
Raspberry Pi Zero 2 W




1. Install Raspberry Pi Zero
Micro SD
4. Insert SD Card
GPi CASE
CARTRIDGE
GPI CASE
2W
2. Install the IO Extension Board
Micro SD
3. Install the back cover
Micro SD


CAUTION

The Zero 2 W provides:

- 1 GHz quad-core ARM processor
- 512 MB RAM
- Wi-Fi
- Bluetooth
- microSD
- mini HDMI
- micro-USB
- 40-pin GPIO footprint

The 40-pin GPIO header is normally **not populated** on a bare Zero 2 W, so be careful when buying a board intended for a case that uses GPIO/pogo-pin connections. (Raspberry Pi)

Important

Do **not** force the Raspberry Pi into the case.

The GPi Case uses a connector/pogo-pin arrangement. Incorrect alignment can damage the pins or prevent the controls/display from working.

The manufacturer's documentation specifically provides installation and support information for the GPi Case 2W. (Retroflag Support)

---

5. Install the Operating System

For this tutorial, I recommend **Recalbox** because it is designed for retro gaming and the GPi Case 2W manufacturer specifically recommends it.

Retroflag states that Recalbox is recommended for the GPi Case 2W and that its configuration is intended to work with the case's display and safe-shutdown functionality. (RetroFlag)

You can alternatively use RetroPie, but Retroflag notes that additional display and safe-shutdown patches are required for the GPi Case 2W when using RetroPie. (RetroFlag)

---

6. Prepare the microSD Card

Install Raspberry Pi Imager on your Windows computer.

[Raspberry Pi Imager and official installation instructions]()

Insert your microSD card into your computer.

Then:

1. Open Raspberry Pi Imager.
2. Select the appropriate Raspberry Pi device.
3. Select the operating system.
4. Select your microSD card.
5. Configure the required settings.
6. Write the image.
7. Allow Imager to verify the card.

Raspberry Pi recommends completing the verification step because it confirms that the image was correctly written. ([Raspberry Pi]())

---

7. Insert the microSD Card

Once the software installation has completed:

```
Raspberry Pi Zero 2 W

   ┌──────────┐
   │ Wi-Fi  │
   └──────────┘
   ┌──────────┐
   │     │
   │ CPU   │
   │     │
   └──────────┘
     │
     │
  ┌──────▼──────┐
  │ microSD  │
  │  slot   │
  └─────────────┘
```

Insert the card into the Raspberry Pi.

Do not power the unit while inserting or removing the microSD card.

---

8. Install the Pi Into the GPi Case

Open the GPi Case.

You will have approximately this arrangement:

```
┌─────────────────────────────┐
│                           │
│     LCD DISPLAY           │
│                           │
│                           │
├─────────────────────────────┤
│                           │
│ D-PAD        A   B        │
│                           │
│          X   Y            │
│                           │
│     SELECT   START        │
└─────────────────────────────┘
```

The Pi Zero 2 W sits inside the rear section.

Make sure:

- The Pi is aligned correctly.
- The connector pins are aligned.
- The Pi is not tilted.
- No wires are being pinched.
- The microSD card is accessible.

---

9. Power the Console

The GPi Case 2W has its own rechargeable battery and power circuitry.

This is much safer for a beginner than designing a lithium-ion charging circuit from individual components.

Do not connect a bare lithium-ion cell directly to the Raspberry Pi.

The case provides the battery and charging system as part of the enclosure. Retroflag specifies a 2800 mAh lithium-ion battery for the GPi Case 2W. (RetroFlag)

---

10. First Boot

Turn the power switch on.

The sequence should look approximately like:

POWER ON
│
▼
Raspberry Pi
bootloader
│
▼
Operating system
│
▼
EmulationStation /
Recalbox interface
│
▼
Game selection

The first boot can take longer than subsequent boots.

If nothing appears on the screen:

1. Turn the device off.
2. Remove the microSD card.
3. Verify the operating-system image.
4. Check the Pi's alignment in the case.
5. Check the GPi display configuration.
6. Reinsert the card.
7. Try again.

---

11. Configure the Buttons

Once the operating system starts, configure the controls.

You should see something similar to:

```
GAME CONTROLLER

   D-PAD

UP
 ▲
◄ ● ►
 ▼

 SELECT   START

   X   Y
   A   B
```

Assign:

- D-pad → movement
- A → primary action
- B → secondary action
- X/Y → additional functions
- Start → pause/start
- Select → menu/hotkey

The GPi Case 2W also includes L/R shoulder buttons. ([RetroFlag]())

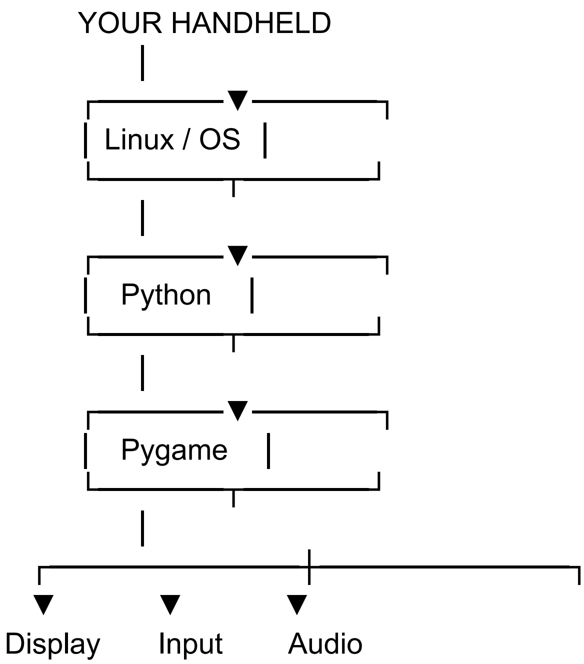
---

12. How the Coding Fits Into the Project

Now we get to the programming portion.

The Raspberry Pi isn't just an emulator. You can use it as a **small Linux computer for writing your own games**.

A useful development architecture is:



This allows you to write your own games rather than simply playing existing games.

---

13. Install Python Development Tools

If you're using Raspberry Pi OS for the programming portion, open Terminal:

sudo apt update
sudo apt upgrade -y

Then install Python and Pygame:

sudo apt install python3 python3-pygame -y

Verify Python:

python3 --version

Then verify Pygame:

python3 -c "import pygame; print(pygame.version.ver)"

If Pygame prints a version number, it is installed.

---

14. Create Your First Handheld Game

Create a directory:

```
mkdir ~/gameboy-project
cd ~/gameboy-project
```

Create a Python file:

```
nano game.py
```

Enter:

```
import pygame
import sys

pygame.init()

WIDTH = 640
HEIGHT = 480

screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("My Raspberry Pi Game")

clock = pygame.time.Clock()

player_x = 300
player_y = 220

speed = 5

running = True

while running:

    for event in pygame.event.get():
```

```
        if event.type == pygame.QUIT:
            running = False

    keys = pygame.key.get_pressed()

    if keys[pygame.K_LEFT]:
        player_x -= speed

    if keys[pygame.K_RIGHT]:
        player_x += speed

    if keys[pygame.K_UP]:
        player_y -= speed

    if keys[pygame.K_DOWN]:
        player_y += speed

    screen.fill((20, 20, 20))

    player = pygame.Rect(
        player_x,
        player_y,
        40,
        40
    )

    pygame.draw.rect(
        screen,
        (0, 200, 100),
        player
    )

    pygame.display.flip()

    clock.tick(60)

pygame.quit()
sys.exit()
```

Save the file.

Run it:

```
python3 game.py
```

You now have a basic graphical game.

---

15. Understanding the Game Code

The most important section is:

```
keys = pygame.key.get_pressed()
```

This reads the controller/keyboard input.

For example:

```
if keys[pygame.K_LEFT]:
    player_x -= speed
```

means:

> If the left control is pressed, move the player left.

Similarly:

```
if keys[pygame.K_RIGHT]:
    player_x += speed
```

moves the player right.

And:

```
if keys[pygame.K_UP]:
    player_y -= speed
```

moves the player upward.

---

16. Add an Enemy

Let's make the game more interesting.

Add:

```
enemy_x = 100
enemy_y = 100
```

Then inside the game loop:

```
enemy = pygame.Rect(
    enemy_x,
    enemy_y,
    40,
    40
)

pygame.draw.rect(
    screen,
    (200, 50, 50),
    enemy
)
```

Now you have:

```
┌──────────────────────────────┐
│                    │
│    RED                 │
│   ENEMY                  │
│                    │
│                    │
│            GREEN     │
│            PLAYER    │
│                    │
└──────────────────────────────┘
```

---

17. Add Collision Detection

Pygame makes collision detection straightforward.

Add:

```
if player.colliderect(enemy):
    print("Collision!")
```

Now the program detects when the player touches the enemy.

A simple game loop becomes:

```
Read controller
    ↓
Move player
    ↓
Update enemies
    ↓
Check collisions
    ↓
Calculate score
    ↓
Draw graphics
    ↓
Play sound
    ↓
Repeat
```

This is the fundamental architecture behind many 2D games.

---

18. Add a Score

Create:

```
score = 0
```

Create a font:

```
font = pygame.font.Font(None, 36)
```

Then render it:

```
score_text = font.render(
    f"Score: {score}",
    True,
    (255, 255, 255)
)

screen.blit(score_text, (20, 20))
```

Now your handheld can display a score.

---

19. Add a Game Over Screen

For example:

```
if player.colliderect(enemy):

    game_over = font.render(
        "GAME OVER",
        True,
        (255, 255, 255)
    )

    screen.blit(
        game_over,
        (250, 200)
    )

    pygame.display.flip()

    pygame.time.wait(2000)

    running = False
```

Your program now has a basic game state.

---

20. Turn Your Game Into a Real Handheld Game

Once the basic game works, organize your project:

```
gameboy-project/
│
├── game.py
├── player.py
├── enemy.py
├── settings.py
│
├── assets/
│   ├── player.png
│   ├── enemy.png
│   ├── background.png
│   └── coin.png
│
├── sounds/
│   ├── jump.wav
│   ├── hit.wav
│   └── gameover.wav
│
└── saves/
    └── savegame.dat
```

This is much better than putting an entire game into one Python file.

---

21. Create a Game Menu

A handheld console should have a menu.

For example:

```
╔══════════════════════╗
║   MY RETRO CONSOLE   ║
╠══════════════════════╣
║                      ║
```

```
║      > MY GAME           ║
║        OPTIONS           ║
║        CREDITS           ║
║                          ║
╚══════════════════════════╝
```

Python can implement this using a simple state machine:

```
state = "MENU"
```

Then:

```
if state == "MENU":
    # Draw menu

elif state == "GAME":
    # Run game

elif state == "GAMEOVER":
    # Display game-over screen
```

This is an important programming concept because it separates different parts of your application.

---

22. Create a Simple Launcher

You can eventually create your own launcher:

```
games = [
    "Snake",
    "Space Shooter",
    "Maze",
    "Platformer"
]

selected = 0
```

Then allow the D-pad to select a game.

Conceptually:

```
        MAIN MENU
            │
   ┌────────┼────────┐
   ▼        ▼        ▼
 Snake  Space Shooter  Maze
   │        │        │
   └────────┼────────┘
            ▼
          PLAY
```

This turns the Raspberry Pi into your own small gaming platform.

---

23. Retro Game Emulation

The other major function of the project is emulation.

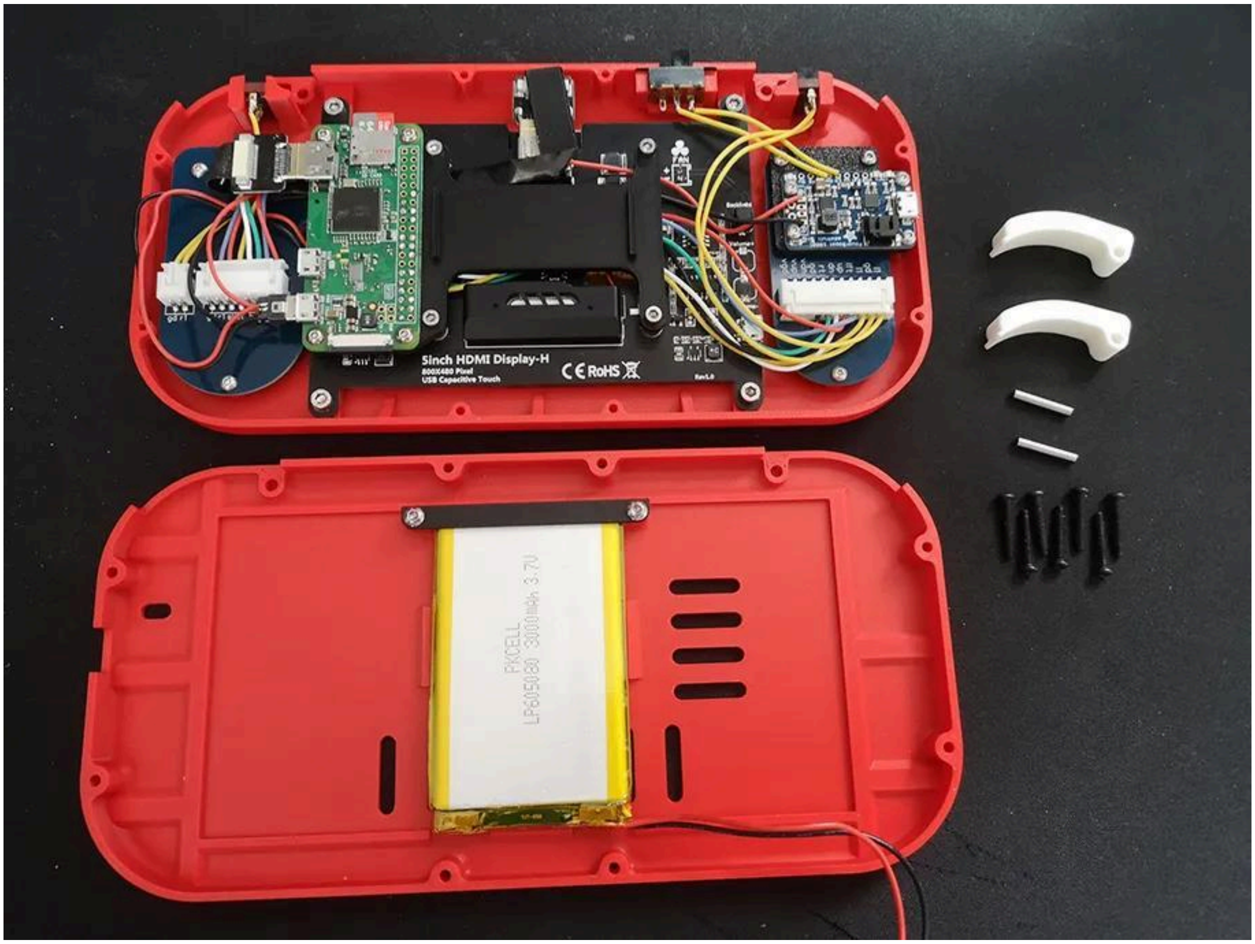
The Raspberry Pi Zero 2 W can be used for many classic systems, although performance varies by emulator and game.

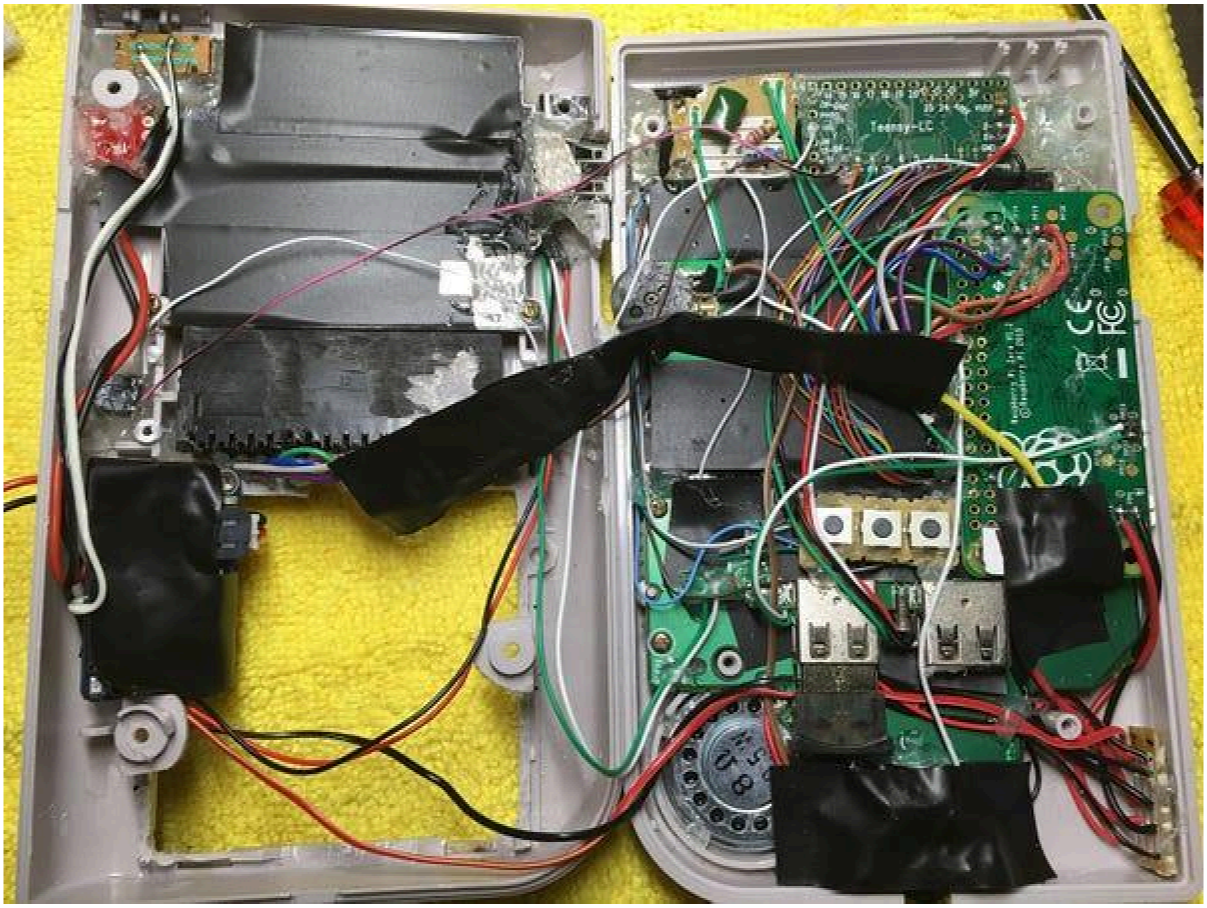
The GPi Case 2W is specifically designed for this type of retro-gaming use. (RetroFlag)

You can also look at the official Raspberry Pi **Game Boy Zero** project for inspiration. Raspberry Pi documented an earlier Pi Zero handheld that used RetroPie, a custom case, additional controls and a display. (Raspberry Pi)


Nintendo GAME BOY™
SELECT
START
B
A
PHONES


SUPER
MARIO WORLD
Nintendo GAME BOY
SELECT
START
B
A
Nintendo GAME BOY
DMG - PI - USA
THIS SIDE OUT
SanDisk
microSD
microSDHC
microSDXC
ADAPTER


5inch HDMI Display-H
800X480 Pixel
USB Capacitive Touch
CE RoHS
PKCELL
LP605080 3000mAh 3.7V


Teensy-LC


GAME BOY ZERO
SUPER
MARIO WORLD
© 1990·1991 Nintendo
Nintendo GAME BOY
SELECT
START

---

24. Adding Games

Use only game software that you have the legal right to use.

Do **not** download copyrighted ROM collections simply because they are available online.

For games you legally own or otherwise have permission to use, the emulator's game directories can be populated according to the operating system's documentation.

A typical organization looks like:

```
roms/
│
├── gameboy/
│   ├── game1.gb
│   └── game2.gb
│
├── nes/
│   ├── game1.nes
│   └── game2.nes
│
├── snes/
│   ├── game1.sfc
│   └── game2.sfc
│
└── arcade/
    └── ...
```

---

25. Transfer Files Over Wi-Fi

One of the advantages of the Zero 2 W is its built-in Wi-Fi. (Raspberry Pi)

You can therefore configure your handheld so that you don't have to remove the microSD card every time you want to transfer something.

For Linux-based systems, SSH/SFTP can be used for remote file transfers.

For example:

```
ssh username@your-handheld
```

You can also use an SFTP program on Windows.

The Raspberry Pi documentation supports configuring Wi-Fi and SSH through Raspberry Pi Imager during OS installation. (Raspberry Pi)

---

26. A More Advanced Project: Direct GPIO Programming

If you eventually build your **own enclosure and button circuit** instead of using the GPi Case, you can connect physical buttons directly to GPIO.

Conceptually:

```
BUTTON
  │
  ├────────── GPIO
  │
  └────────── GND
```

For example:

```
         Raspberry Pi
         Zero 2 W GPIO
         ┌──────────────────┐
LEFT  ────────▶ GPIO        │
RIGHT ────────▶ GPIO        │
UP    ────────▶ GPIO        │
DOWN  ────────▶ GPIO        │
A     ────────▶ GPIO        │
B     ────────▶ GPIO        │
START ────────▶ GPIO        │
SELECT────────▶ GPIO        │
         └──────────────────┘
```

A Python program can then monitor those GPIO inputs and convert them into game actions.

This is where the project moves from **assembling a retro console** into a genuine **embedded-systems programming project**.

---

27. Recommended Development Path

I recommend building the project in stages rather than trying to do everything simultaneously.

Level 1 — Hardware

Build:

Pi Zero 2 W
+
GPi Case 2W
↓
Working handheld

Level 2 — Operating System

Install:

Recalbox
↓
EmulationStation
↓
Retro gaming

Level 3 — Linux

Learn:

cd
ls
mkdir
cp
mv
sudo
apt
ssh

Level 4 — Python

Learn:

Variables
↓
Conditions
↓
Loops

↓
Functions
↓
Classes
↓
Pygame

Level 5 — Game Development

Build:

Input
↓
Player
↓
Enemies
↓
Collision
↓
Score
↓
Sound
↓
Levels
↓
Save system

Level 6 — Embedded Development

Eventually build:

Custom PCB
+
Custom buttons
+
Custom display
+
Battery management
+
Custom 3D-printed case
+
Custom Python/C/C++ software

28. Final Project

When finished, your project could look like:


PlayStation
OPTIONS
71 GAMES AVAILABLE


ES emulation station

And the software architecture could be:

```
        RASPBERRY PI ZERO 2 W
              |
              ▼
           Linux / OS
              |
    ┌─────────┴─────────┐
    ▼                ▼
Retro Emulator      Python
    |            Game Engine
    ▼                |
Retro Games              ▼
                 Your Games
                     |
                     ▼
              Display + Controls
                     |
                     ▼
               Audio + Saves
```

Suggested final feature list

Your finished console can include:

- 🎮 D-pad
- 🔴 A/B buttons
- 🔵 X/Y buttons
- ⏯ Start/Select
- L/R shoulder buttons
- 🖥️ 3-inch IPS display
- 🔊 Built-in speaker
- 🎧 Headphone output
- 🔋 Rechargeable battery
- 📶 Wi-Fi
- 🟦 Bluetooth
- 💾 microSD storage
- 🐍 Python game development
- 🕹️ Retro-game emulation
- 💻 SSH/SFTP development
- 💾 Save-game support

The **GPi Case 2W + Raspberry Pi Zero 2 W** route is the approach I'd recommend for your first build because it lets you concentrate on **coding and software development instead of spending most of the project designing a battery-management and display circuit**.

Raspberry Pi has also documented multiple Pi Zero handheld projects, demonstrating that the Zero platform is well suited to this form factor. (Raspberry Pi)

Useful official resources

Raspberry Pi Zero 2 W — hardware specifications and purchasing information.

Raspberry Pi Getting Started Documentation — OS installation and Raspberry Pi Imager instructions.

Retroflag GPi Case 2W — case specifications, compatibility and setup information.

Retroflag Support — display patches and safe-shutdown information.

One important distinction: the GPi Case approach gives you a professionally designed handheld enclosure, while a **completely custom version**—3D-printed Game Boy-style case, individual LCD, physical GPIO buttons, custom PCB, battery-management board, and your own Python launcher—would be a much deeper electronics/programming project.