ADVANCED COMPUTER NETWORKS - GROUP 7

Network-Based Remote Game Deployment System

on Arduino Using a Raspberry Pi

Sebastián Pérez - F11315118

Marco Verón - F11315123

Sebastián Nuñez - F11315117

Carlos Gernhofer - F11315110

NTUST - Dept. of Computer Science and Information Engineering

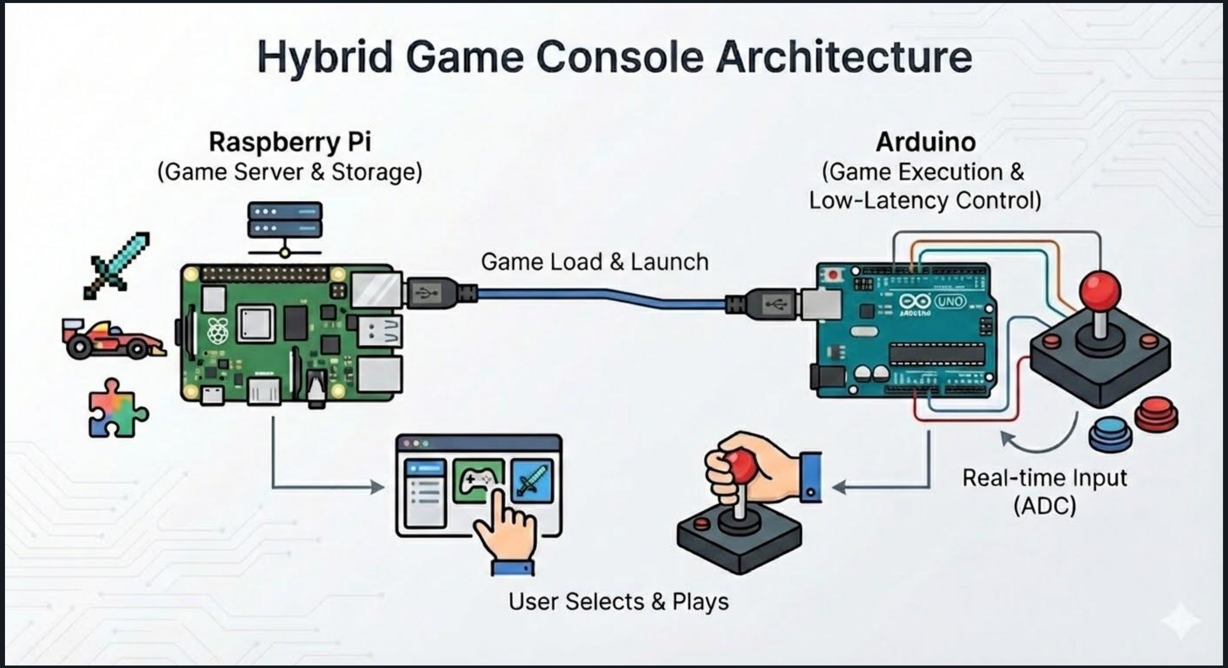
Motivation & Objective

Motivation

- Arduino offers excellent latency and analog-to-digital converter (ADC) readouts for controllers like joysticks, but lacks in storage and network connectivity.
- Raspberry Pi offers powerful network management and massive storage, ideal for hosting a library of multiple games.
- Synergy: Combining these two worlds allows you to separate the storage logic (RPi) from the real-time execution (Arduino) and get the benefits from both.

The Goal

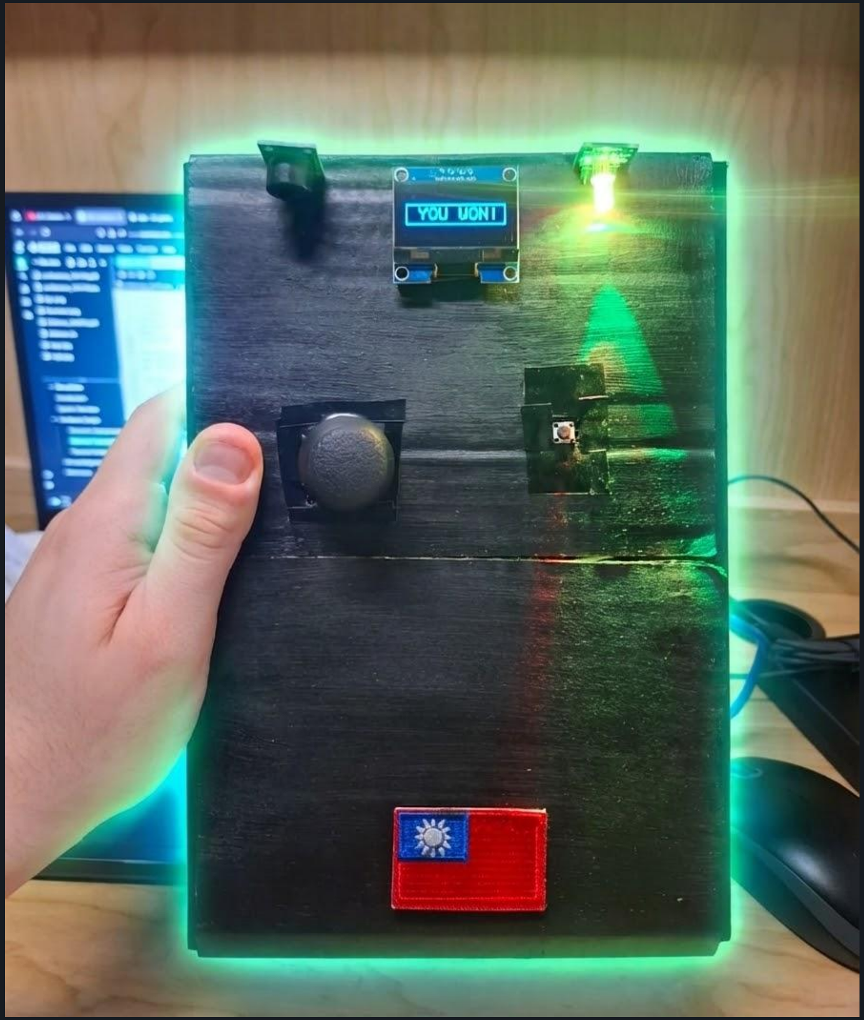
A console where the Raspberry Pi acts as the deployment server (host) and the Arduino as the execution engine, allowing you to dynamically select and load games over the network.



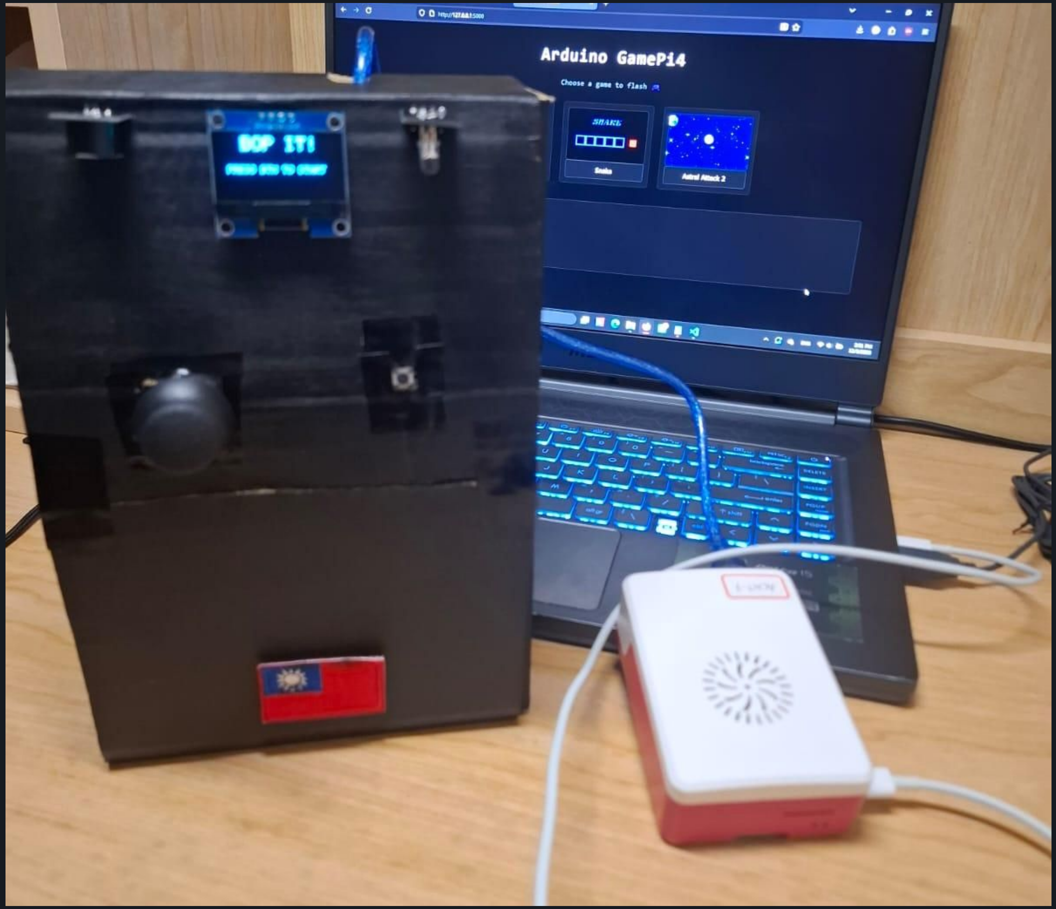
Introduction

What We Built

- A small-scale game console ecosystem using an Arduino UNO and a Raspberry Pi 4.
- The Raspberry Pi stores a library of precompiled game binaries and hosts a Flask-based web interface from which users can select games.
- The Arduino runs the game and allows the user to interact with it through the connected components.
- This architecture demonstrates network integration and the effective cooperation of two microcontrollers with distinct roles.



System Overview



How the Components Connect

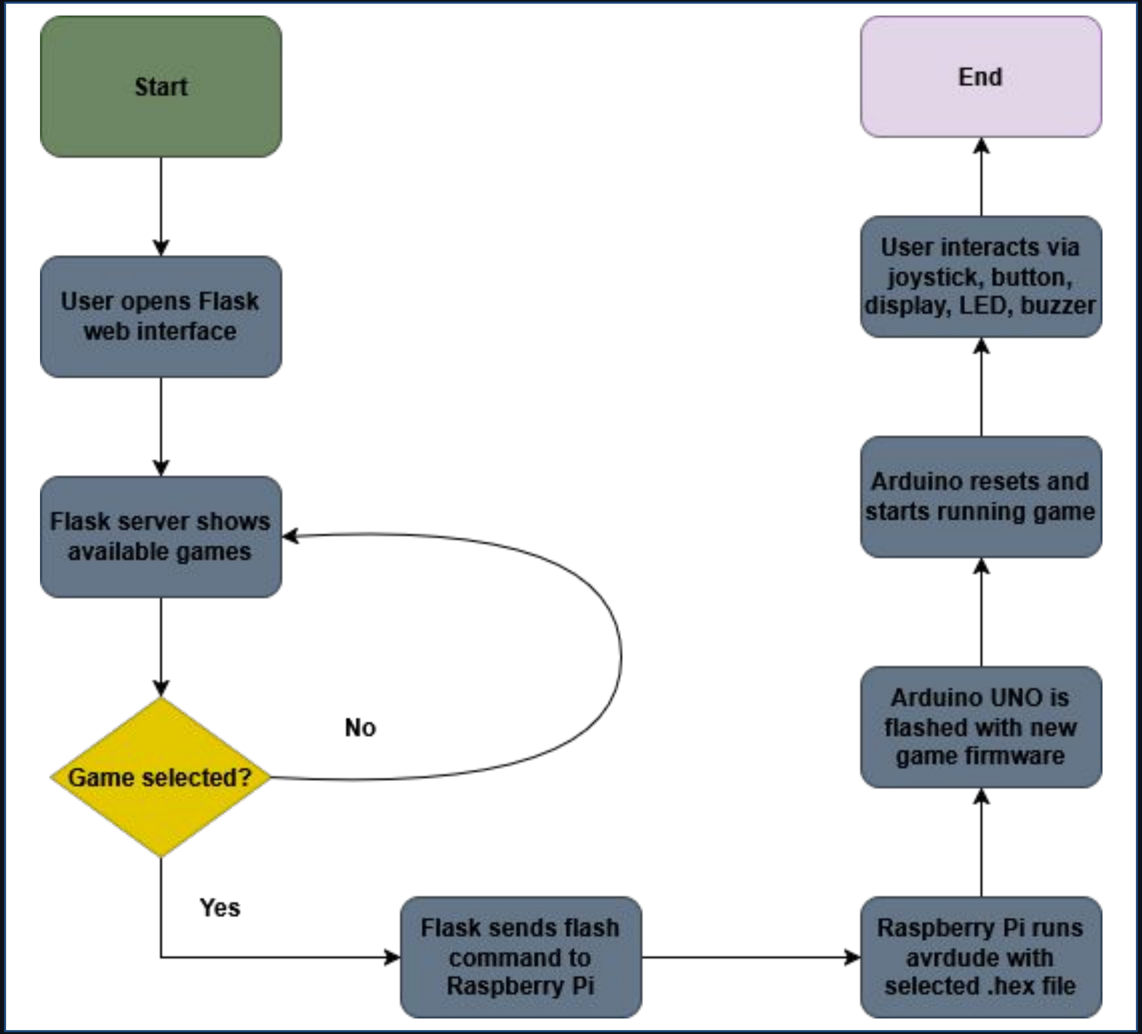
1. **Arduino UNO & Raspberry Pi 4:** They are connected over a USB serial connection. The Raspberry Pi 4 flashes the Arduino UNO through this connection.

2. **The Server:** The Raspberry Pi 4 hosts a Flask web server and must remain connected to a power source and the local Wi-Fi network to properly operate.

3. **The Client:** The user connects to the system through the Flask web server. They may select a game to load from the interface provided.

4. **Game Execution:** Once the game is loaded, the user may interact with it through the input and output modules provided connected to the Arduino (joystick, button, display, LED, buzzer).

System Overview 2



Start
User opens Flask web interface
Flask server shows available games
Game selected?
No
Yes
Flask sends flash command to Raspberry Pi
Raspberry Pi runs avrdude with selected .hex file
Arduino UNO is flashed with new game firmware
Arduino resets and starts running game
User interacts via joystick, button, display, LED, buzzer
End

System Flowchart

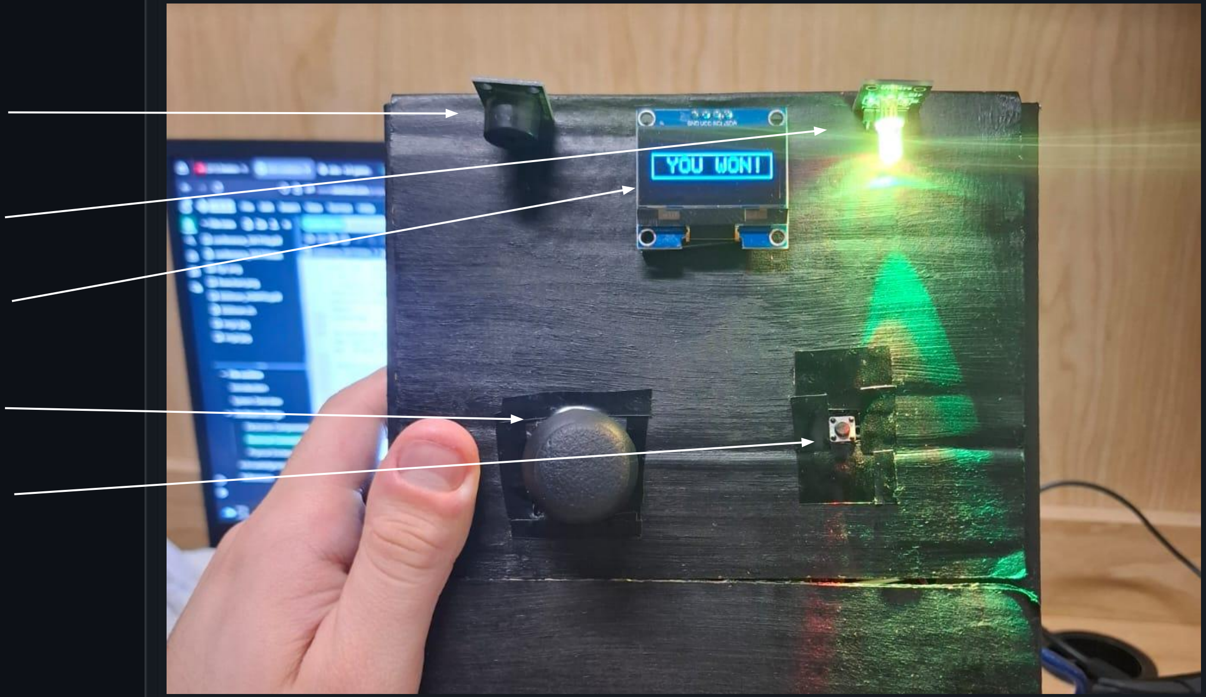
1. **User** opens the Flask web interface and sees the list of available games.
2. **User** selects a game, sending a POST request to the Flask server.
3. **Raspberry Pi** runs AVRDUDE with the chosen .hex file.
4. **Arduino UNO** is flashed and automatically resets.
5. The new game runs, and **the user** interacts via the joystick, button, display, LED, and buzzer.

Key Benefit of this structure:
Games can be switched remotely over Wi-Fi without a direct USB connection or the Arduino IDE.

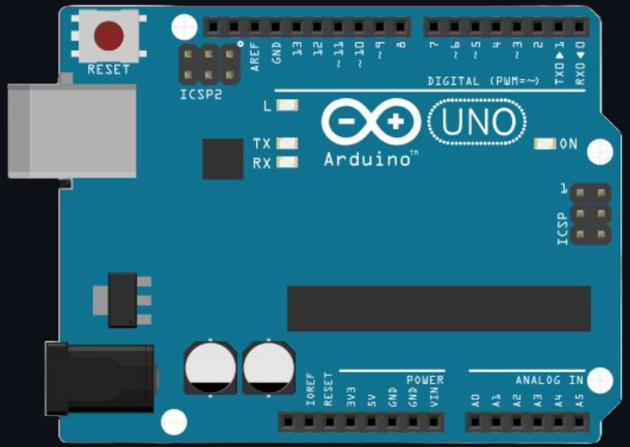
Hardware Integration - Components

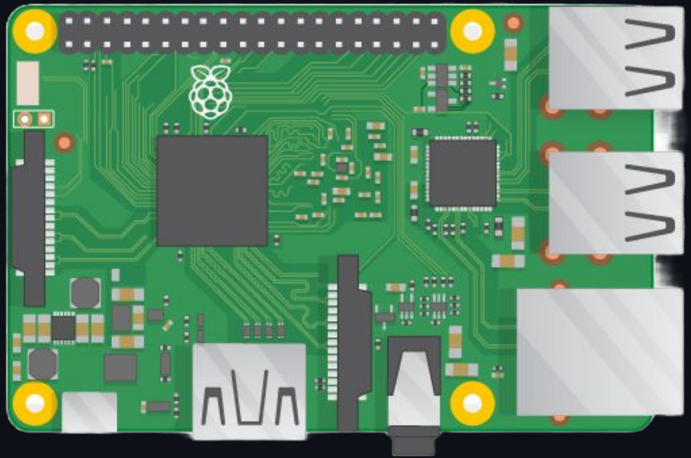
Hardware Components

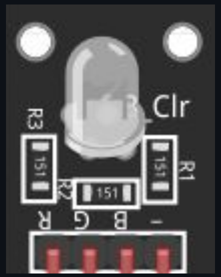
- **Arduino UNO –** Executes the game logic and handles all I/O.
- **Raspberry Pi 4 –** Stores game binaries, hosts the Flask server, and flashes the Arduino.
- **Passive Buzzer –** Provides sound effects (events, actions, alerts).
- **RGB LED –** Indicates game states such as player health, difficulty, or cooldowns.
- **1.3-inch SH1106 OLED Display –** Shows game graphics and UI.
- **Joystick Module –** 2-axis analog input + push switch for gameplay control.
- **Push Button –** Used for firing, selecting, or triggering actions depending on the game.
- **Breadboard + Jumper Wires –** Internal wiring for all modules.
- **USB Cable & USB-C PSU –** Power and data connection between Raspberry Pi and Arduino.



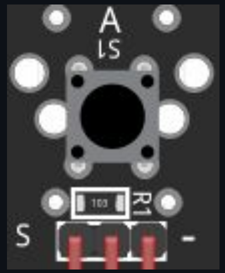
Hardware Integration - Components

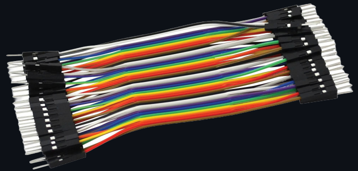

Arduino Uno


Raspberry Pi 4

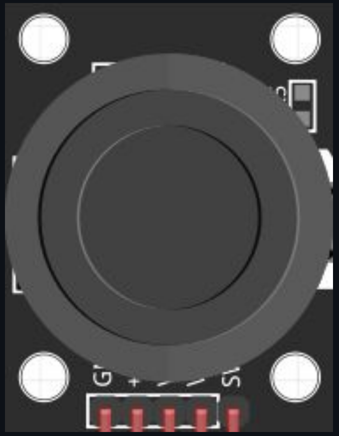

LED

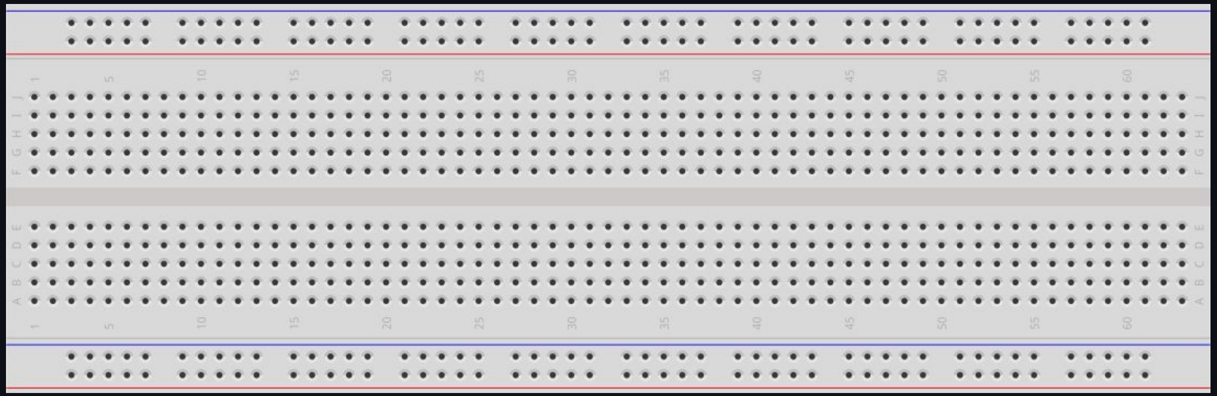
Buzzer


Button


Jumper Wires

OLED 1.3 128×64

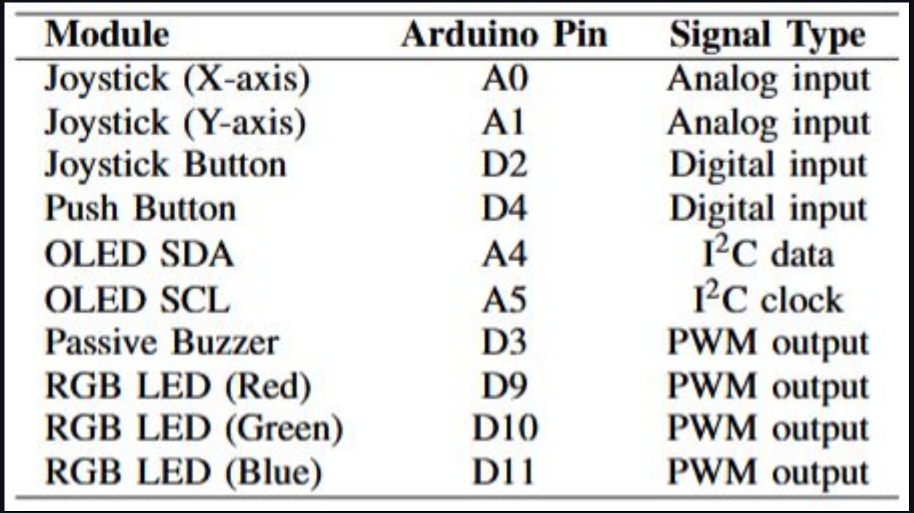

Joystick


Breadboard

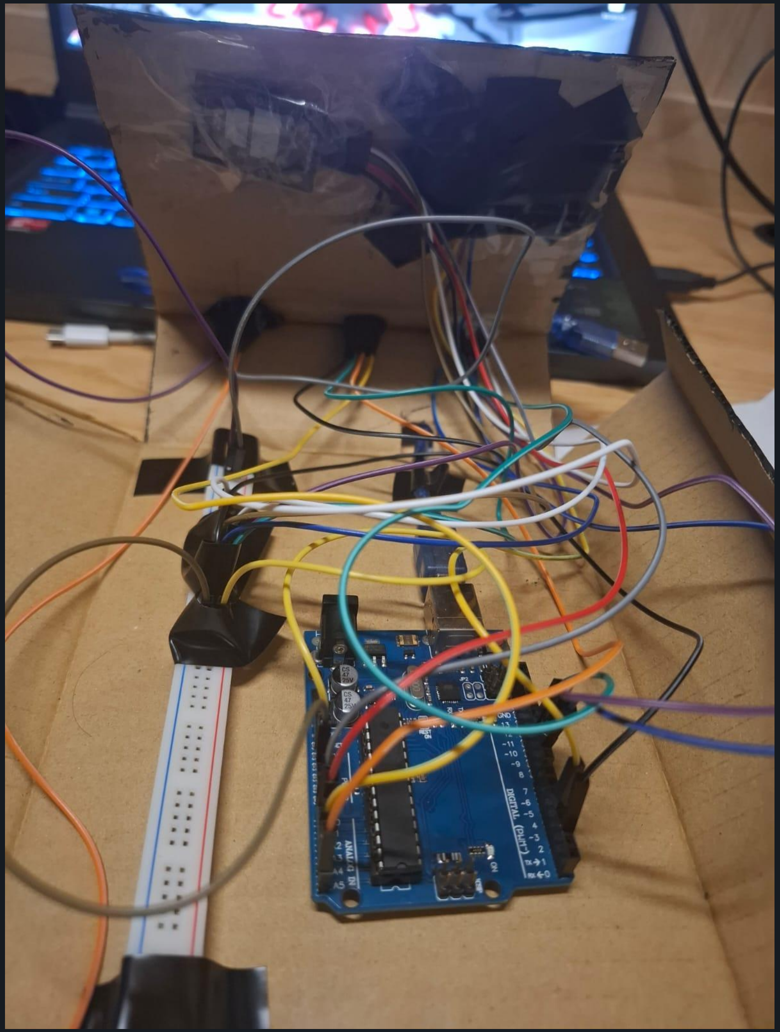
Hardware Integration - Electrical Connection & Wiring

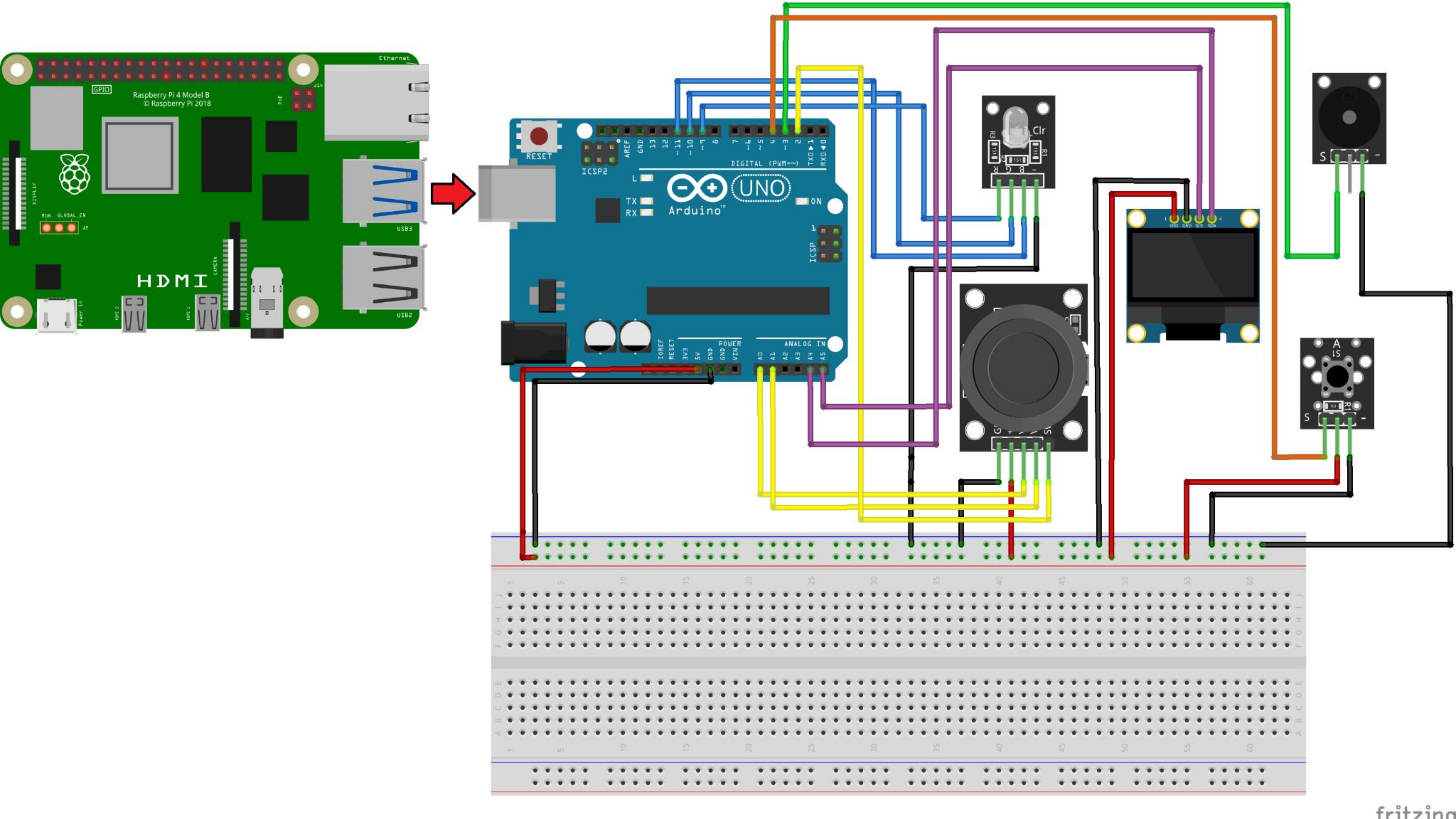
Component Connections

- All modules are mounted inside the cardboard enclosure, except for the Raspberry Pi.
- Components and wiring are secured using tape to avoid damage (borrowed components).
- A small opening in the enclosure allows the USB cable to reach the Raspberry Pi.



Module	Arduino Pin	Signal Type
Joystick (X-axis)	A0	Analog input
Joystick (Y-axis)	A1	Analog input
Joystick Button	D2	Digital input
Push Button	D4	Digital input
OLED SDA	A4	I^2C data
OLED SCL	A5	I^2C clock
Passive Buzzer	D3	PWM output
RGB LED (Red)	D9	PWM output
RGB LED (Green)	D10	PWM output
RGB LED (Blue)	D11	PWM output



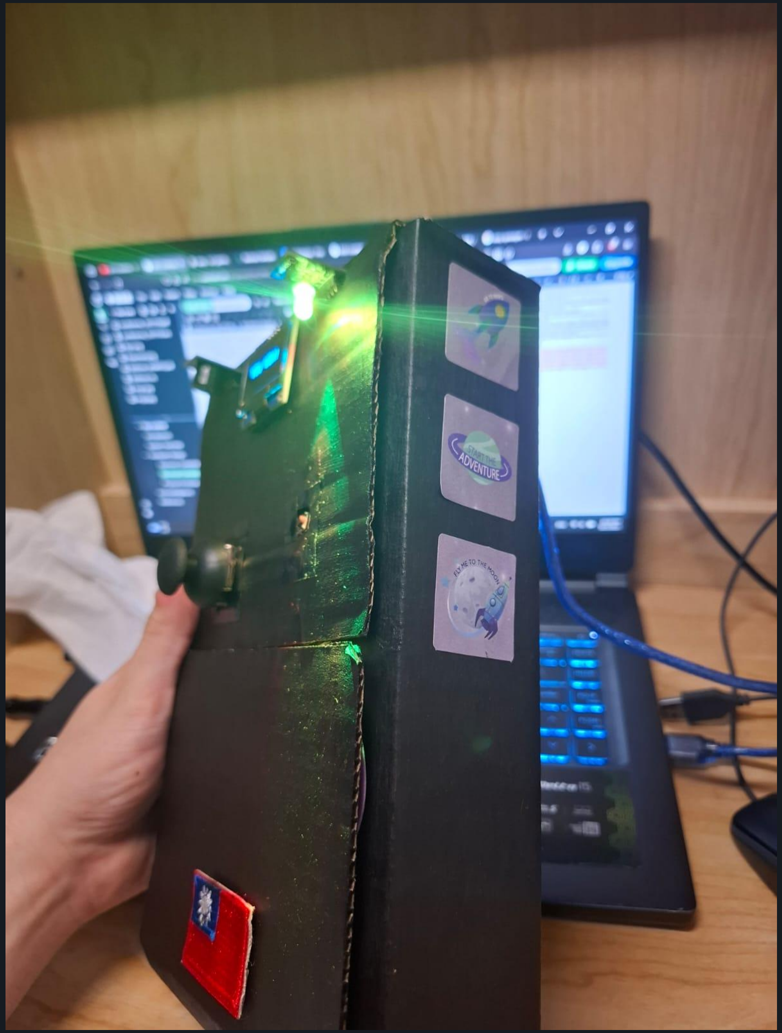


Raspberry Pi 4 Model B
© Raspberry Pi 2018
GPIO
Ethernet
HDMI
USB3
USB2
RESET
ICSP2
DIGITAL (PWM=~)
UNO
Arduino
ON
TX
RX
L
ICSP
POWER
ANALOG IN
IOREF RESET 3V3 5V GND GND VIN
A0 A1 A2 A3 A4 A5
Clr
VDD GND SCK SDA
fritzing

Hardware Integration - Physical Enclosure

Physical Enclosure

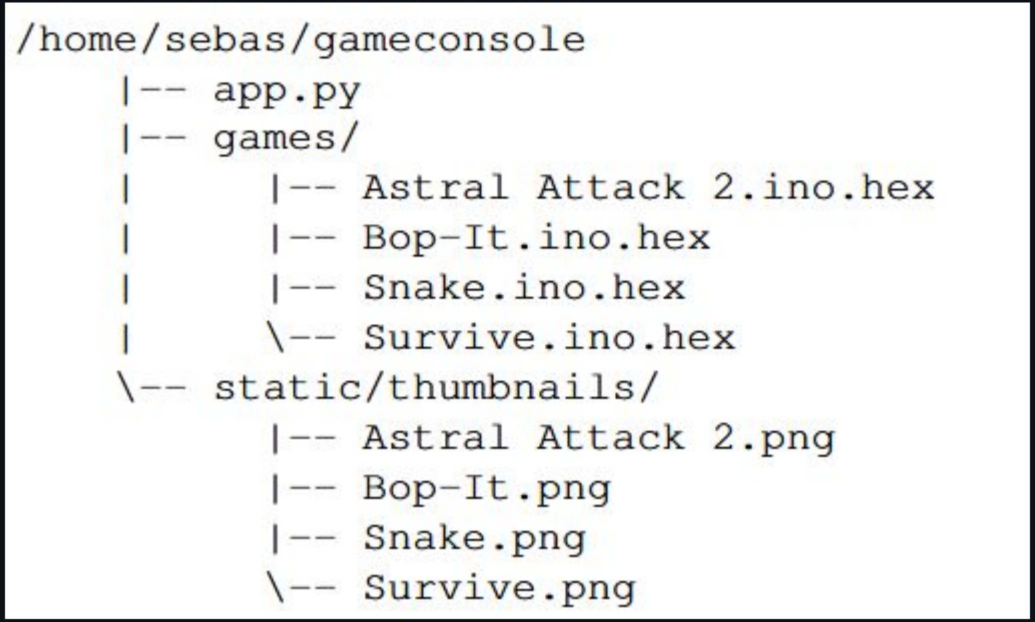
- Enclosure built from layered cardboard panels due to it being lightweight, cheap, and easy to modify.
- Openings were cut in the top for joystick and button.
- Output modules (OLED, RGB LED, buzzer) were mounted by inserting pins through cardboard.
- Internal wiring is reinforced with tape. Other methods were discarded as to avoid permanent changes to the borrowed components.
- Enclosure painted black for a console-like appearance, and some aesthetic additions were made (stickers + flag).
- Raspberry Pi was kept outside to maintain easy accessibility to it and to reduce enclosure weight.



Software Architecture - Flask Server

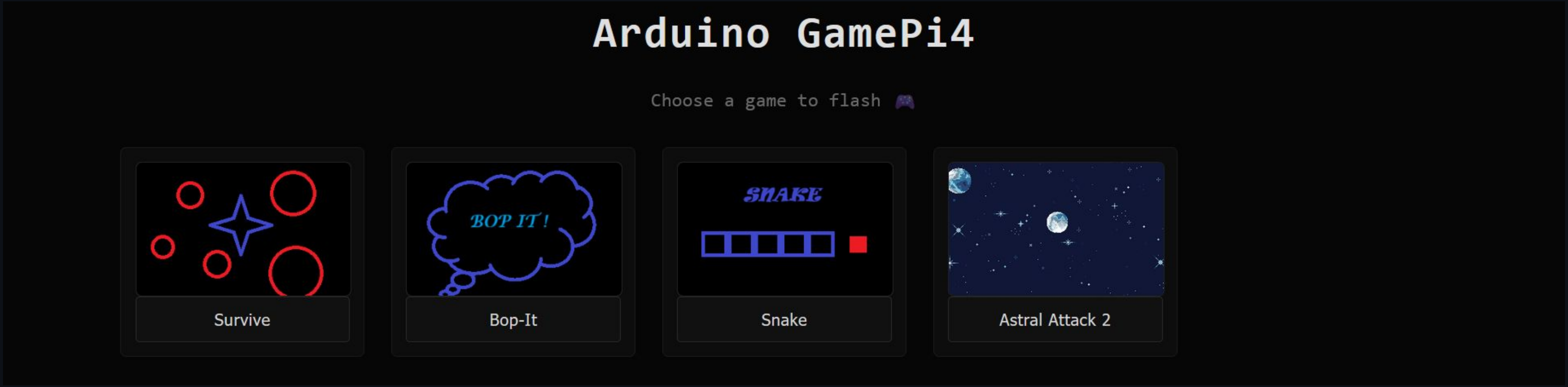
Raspberry Pi File Structure

- */home/sebas/gameconsole* is the directory where the entire system is implemented.
- The server is implemented in the *app.py* file.
- The games are located within the *games/* folder. Any file in the proper format within this folder gets automatically added to the web interface.
- Game thumbnails are located inside the *static/thumbnails/* directory. The file names inside this folder have to match the names of the games in order for the thumbnail to be applied in the interface.



```
/home/sebas/gameconsole
    |-- app.py
    |-- games/
    |       |-- Astral Attack 2.ino.hex
    |       |-- Bop-It.ino.hex
    |       |-- Snake.ino.hex
    |       \-- Survive.ino.hex
    \-- static/thumbnails/
            |-- Astral Attack 2.png
            |-- Bop-It.png
            |-- Snake.png
            \-- Survive.png
```

Software Architecture - Flask Server



Flask Server

For each game, the server constructs a display name, a thumbnail image if provided, and fills in the HTML interface with each component.

Behind the Scenes

Whenever the user selects a game, it triggers a POST signal to the /flash endpoint. This signal contains the name of the selected game and starts the remote flashing pipeline.

```
POST /flash data: { game: "snake.hex" } ⟶
Triggers Python Subprocess
```

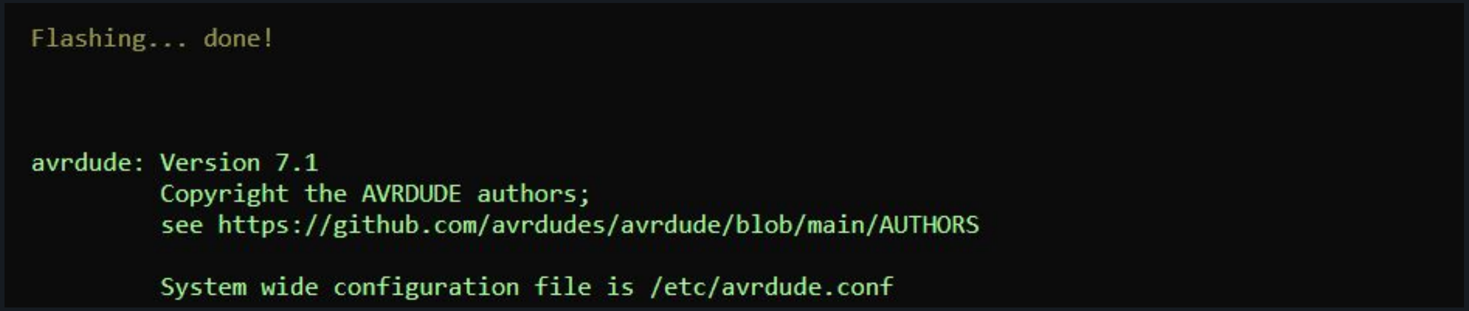
Software Architecture - Remote Deployment Pipeline

Once the signal is received, the pipeline proceeds as follows:

1. **Selection:** The correct *.hex* game file is identified from the */games/* directory.
2. **Execution:** The server runs an **AVRDUDE** command with the appropriate *.hex* file, which starts the flashing process to the Arduino UNO over the USB serial connection.

```
avrdude -p atmega328p -c arduino -P /dev/ttyACM0 -U flash:w:game.hex:i
```

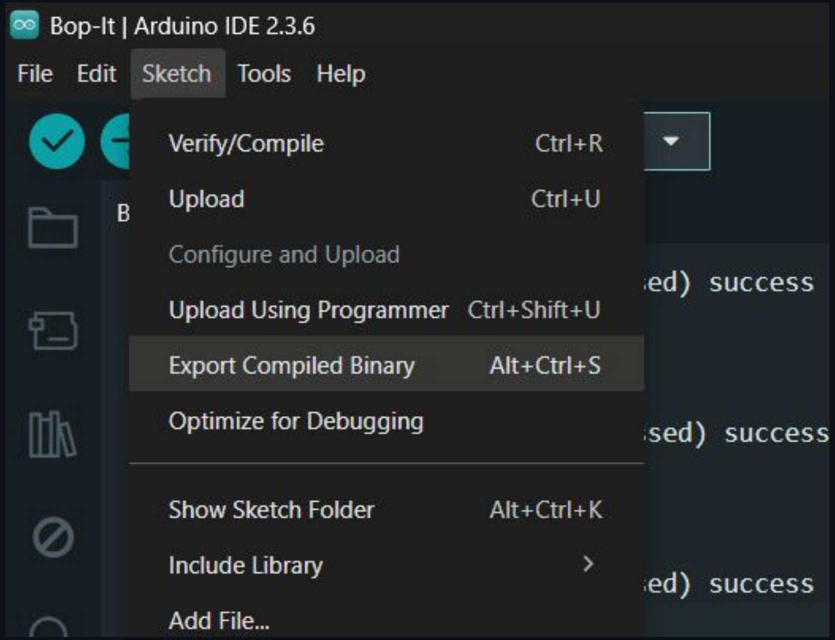
3. **Loading:** The **AVRDUDE** command places the Arduino into bootloader mode, erases the currently loaded firmware, and writes the new game binary into the Arduino. Once the flashing is complete, the Arduino resets and starts to execute the selected game.



```
Flashing... done!



avrdude: Version 7.1
         Copyright the AVRDUDE authors;
         see https://github.com/avrdudes/avrdude/blob/main/AUTHORS

         System wide configuration file is /etc/avrdude.conf
```

Software Architecture - Game Implementations

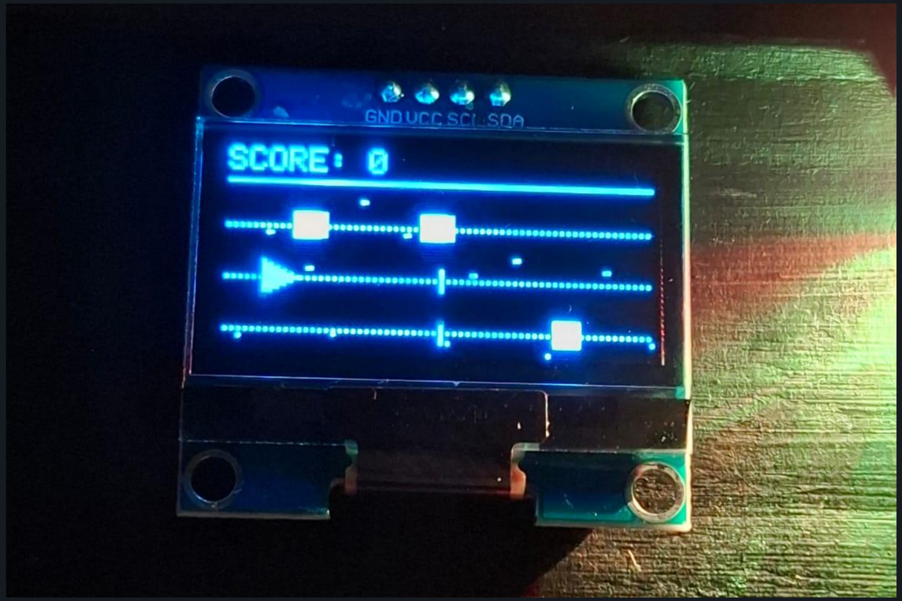
- All of the games were first developed as standard Arduino sketches in the Arduino IDE, where files are in .ino format.
- The SH110X and GFX libraries were used to interact with the OLED display.
- Since the Raspberry Pi cannot compile Arduino sketches directly, the games had to be exported as precompiled .hex binaries.
- The Arduino IDE provides this export functionality within the Sketch menu. These binaries contain the machine code which can be flashed directly to the Arduino from the Raspberry Pi using AVRDUDE.


Bop-It | Arduino IDE 2.3.6
File Edit Sketch Tools Help
Verify/Compile Ctrl+R
Upload Ctrl+U
Configure and Upload
Upload Using Programmer Ctrl+Shift+U
Export Compiled Binary Alt+Ctrl+S
Optimize for Debugging
Show Sketch Folder Alt+Ctrl+K
Include Library
Add File...
ed) success
sed) success
ed) success

Astral Attack 2

A horizontal space shooter game.

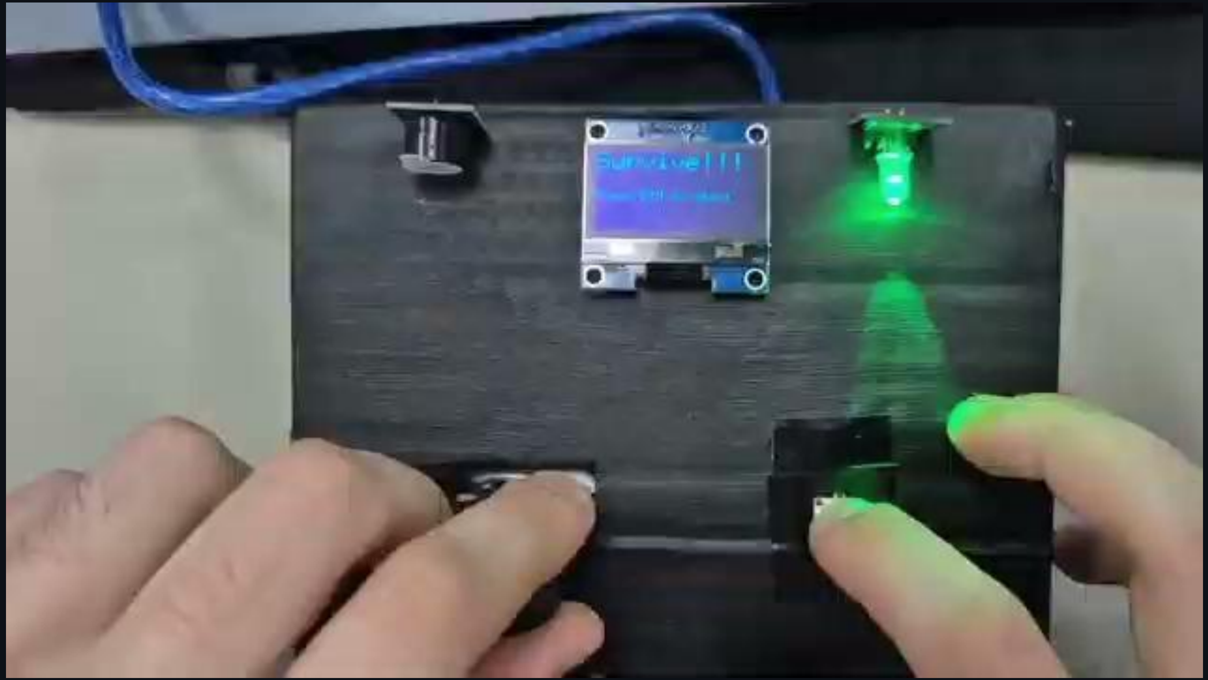
- **Inputs:** Joystick for vertical movement between lines, Button for shooting.

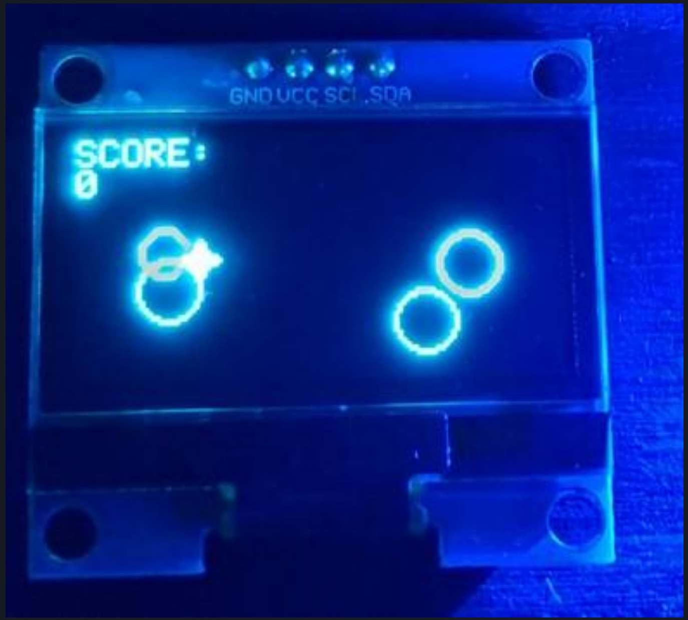


Bop-It

A speed-reaction game.

- **Inputs:** Joystick and button for commands given by the game.
- **Multiplayer:** Has functionality for up to four players.

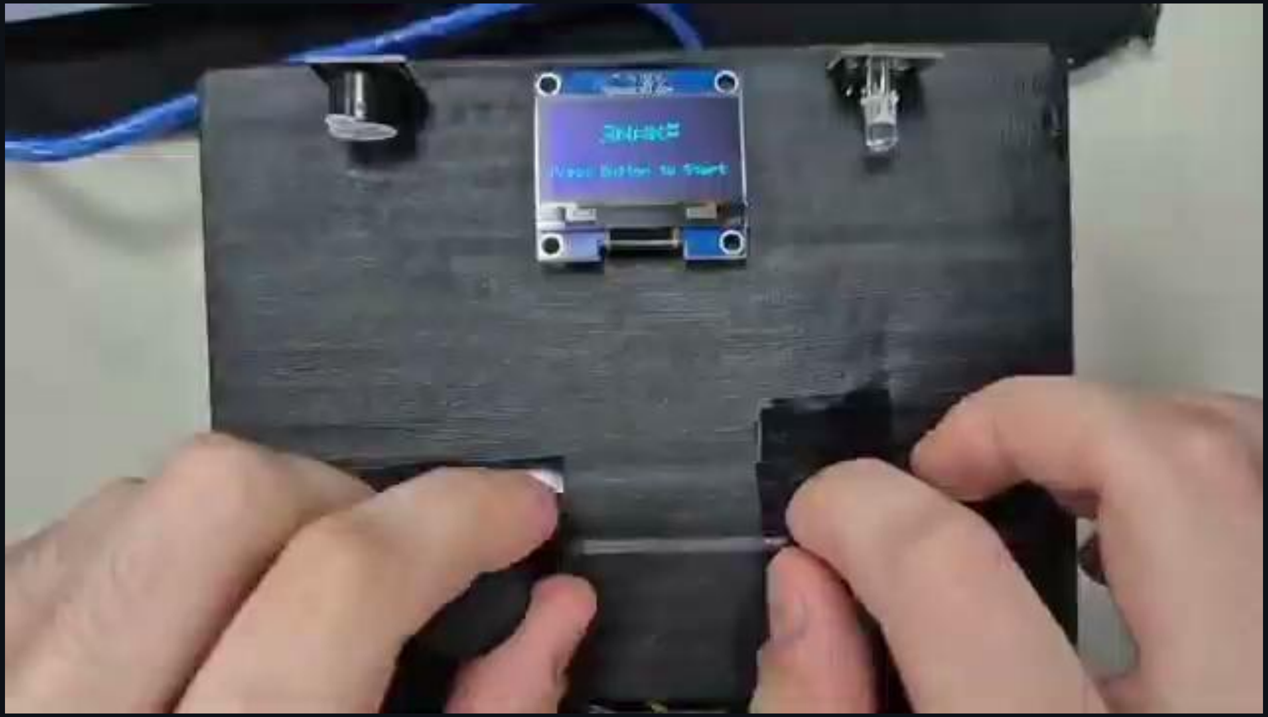




Survive

An avoidance endurance test.

- **Inputs:** Joystick for movement and button for invincibility skill.



Snake

The classic snake game.

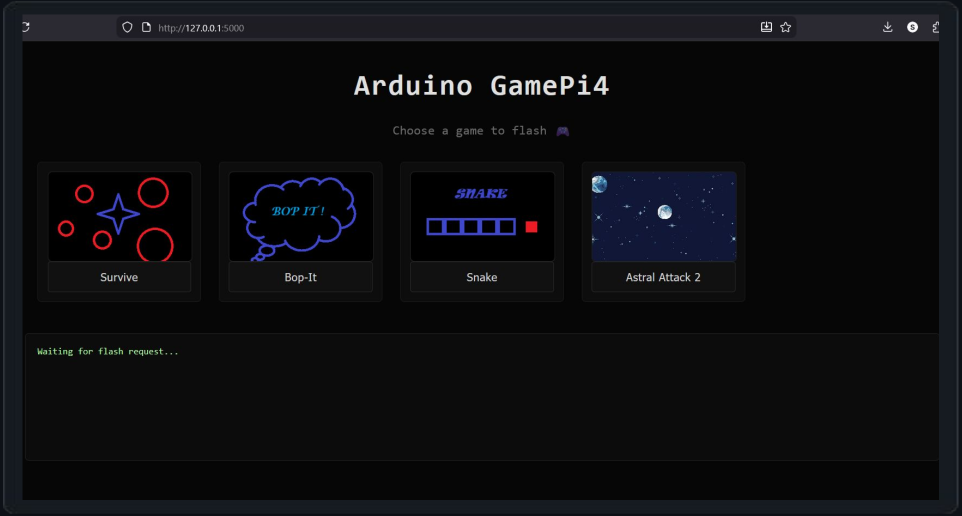
- **Inputs:** Joystick for movement.

🏁 Conclusion & Key Takeaways

1. We successfully separated the Game Execution (on Arduino) from the Game Management (on Raspberry Pi), creating a truly distributed console.

2. The Raspberry Pi acts as a network bridge. It receives instructions via Wi-Fi and automatically handles the physical USB flashing of the Arduino, removing the need for a PC with an IDE.

3. We overcame the Arduino's storage limits. By keeping the library on the Raspberry Pi, we can swap different games into the Arduino's memory on demand.



```
Waiting for flash request...
```

Q & A

Thank you for your attention.

marco.veron@uptp.edu.py | carlosa.gernhofer@uptp.edu.py | sebastian.nunez@uptp.edu.py | sebastian.perez@uptp.edu.py

PROJECT GITHUB:
https://github.com/Zeviant/Network-Based-Remote-Game-Deployment-System-on-Arduino-Using-a-Raspberry-Pi