Adaptive Hungry Hippos Game

– Build Guide –

Table of Contents

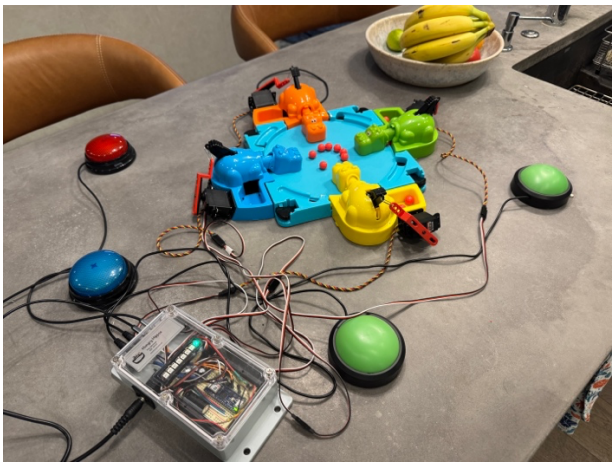
Summary

This project shows how I modified the classic *Hungry Hungry Hippos* game using an Arduino, adaptive switches, and servos so it can be played by individuals with physical limitations. I've shared everything I used in this build — parts list, wiring, code, and 3D-printed components — in case others want to learn from it, adapt it, or create something similar. While it's not a formal tutorial, I've tried to keep the documentation clear and beginner-friendly. Beyond this specific build, I hope this project sparks ideas for how accessible, DIY electronics can be used to adapt other games or create new forms of play. Everything, including the Arduino code and 3D-printed part designs is licensed under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0) and can be adapted for non-commercial use.

Background

A parent was looking to modify the *Hungry Hungry Hippos* game with adaptive switches to make it accessible for their child. The idea came from an earlier implementation by Bob Paradiso, who adapted the game several years ago. The basic setup replaces the hand-operated levers with four servos — one for each hippo — triggered by large, colorful adaptive switches. A low-cost microcontroller (like an Arduino) monitors the switches and activates the corresponding servo. When a button is pressed, the servo moves the hippo's lever, causing it to lunge forward, scoop up marbles, and retract.

I built a version of this modification by loosely following Bob's original design. Since the parent was just getting started with Arduino — in part to explore more options for their child — I documented the build so they could reference it later. But once I began writing up the parts list and wiring notes, I realized others might find value in my process too, especially if they're interested in modifying games for accessibility or just tinkering with microcontrollers and servos.



Of course, there's no such thing as a "typical" parent in this space. Every child has different needs, and every parent brings a different mix of time, tools, and experience. That's actually part of what makes Arduino such a good match — it's flexible, customizable, and supports creative problem-solving from wherever you're starting.

Everything, including this document, the Arduino code and 3D-printed part designs — is stored in this GitHub repository, and is licensed under the Creative Commons Attribution-NonCommercial 4.0 International (CC BY-NC 4.0). It's probably easiest to download the entire repository as a ZIP file using GitHub's built-in Download ZIP feature. Feel free to adapt anything here for your own non-commercial use! Just keep in mind that I'm not an expert — I'm a hobbyist sharing what

worked for me. If you find ways to improve it or spot a mistake, I'd love to hear about it. I make no warranties or claims about the fitness of this design or approach. You're ultimately responsible for verifying your own implementation.

Observations on the Original Design

The original adaptive Hungry Hungry Hippos project was created around 2018 by Bob Paradiso. The online blog post is still available here. While the core idea is well demonstrated, there's limited technical detail provided, and two of the related implementation links are no longer active. There are no formal tutorials or build instructions included with the original post, but several design elements are visible through the blog and accompanying images.

Here are a few observations:

Servo Idling Using Transistors: The servos in the original build produced an audible motor whine even while holding position. This is typical behavior for digital servos, which constantly correct their position. To mitigate this, a transistor was added for each servo, acting as a low-side switch to cut off power when the servo was not actively in use. This helped reduce noise and power draw while idle.

Power Supply Behavior: The blog doesn't specify the exact power supply used, but it notes that the supply would occasionally shut down when all the servos activated at once — likely due to high inrush current. To address this, a very large capacitor was added to buffer the startup surge and stabilize the voltage during servo activation.

Electronics Placement and Protection: All electronics were mounted on a custom PCB, which was hot-glued under the center of the game board. This helped conceal and protect the wiring and components. To accommodate the extra hardware below the board, the entire game was elevated on risers made from Duplo blocks — a simple and creative solution for gaining clearance.

My Design Changes

Breadboard Instead of PCB: My implementation uses a small breadboard rather than a custom PCB. While PCBs are often better for permanent setups, I didn't have a suitable board for mounting the servo headers, and I felt that a breadboard would be more approachable and adaptable for parents who are just beginning to explore Arduino-based projects.

Analog Servos Instead of Digital: I opted for analog servos (HS-645-MG) instead of digital ones. They're significantly cheaper, quieter, and draw less power — all helpful traits in this application. While analog servos are slightly slower and less precise, they're perfectly fine for hobby use. This change also allowed me to eliminate the four transistors used in the original design to cut power to the digital servos when idle.

Microcontroller: I used an Arduino Nano 33 IoT board that I had on hand. These days, it's considered a fairly low-end board, but just about any microcontroller would be sufficient for this project — including most Arduino and Arduino-compatible options. Many modern boards include Bluetooth or Wi-Fi, which aren't needed here, so a simpler, lower-spec board is ideal.

A critical requirement for me was a microcontroller that could be powered through a pin (VIN) — a feature supported by most Arduino-branded boards — allowing it to share the same power supply as the servos. I also wanted a small form factor to maximize available space on the breadboard.

Eliminated the Resistors: The original design included four 1K resistors between the adaptive switches and the microcontroller. It's not entirely clear whether they were intended as pull-up resistors or for current limiting, but I found that they weren't needed in my setup.

Most Arduino-compatible boards, including the Arduino Nano 33 IoT, support internal pull-up resistors, which eliminate the need for external ones in simple switch circuits. I used this built-in feature with a single line of code in the setup() function:

```
pinMode(switchPins[i], INPUT_PULLUP);
```

The adaptive switches are normally open and connect to ground when pressed, so they create a low-side connection. In this configuration, there's no significant current flowing into the GPIO pin — just a logic-level change — so a current-limiting resistor isn't necessary. That said, adding a small resistor (e.g., 220–1K ohms) is sometimes done as a precaution in longer cable runs or less predictable environments, but it's not required here.

6V / 10A Power Supply: The power requirements are driven mostly by the servos. Each HS-645MG servo has a stall current of 2.5A, so in a worst-case scenario where all four stall at once, that's up to 10A. In practice, not all servos will stall simultaneously or for long, but it's smart to plan for peak demand. I also added about 0.5A of headroom for the microcontroller and any additional components. While these servos can run between 4.8V and 6V, I chose 6V for snappier servo response. A 5A supply might have technically worked, but the next common step up from 6V/5A was 6V/10A, which gave me both extra performance and breathing room.

Capacitors for Extra Protection: Although the large power supply seems to handle all four servos without issue, I added one large capacitor across the main power lines and four smaller capacitors near each servo. This helps smooth out transient voltage drops caused by brief but high current spikes when the servos start moving.

16V Capacitors: The original design used a 5.4V capacitor, which seemed under-spec'd given that the system runs at 6V. Capacitors should be rated higher than the supply voltage to handle spikes, noise, and long-term wear. A good rule of thumb is 1.5x to 2x the supply voltage, so I used 16V-rated capacitors for better reliability and safety.

Enclosed Electronics: Rather than mounting exposed wires and components underneath the game, I enclosed the electronics in a case with a clear lid. This keeps them protected, accessible, and visible — and also gave me the opportunity to add status LEDs that can be easily seen during use.

The generic case required several custom parts to accommodate the components. All of these parts were 3D printed, and the design files are attached below.:

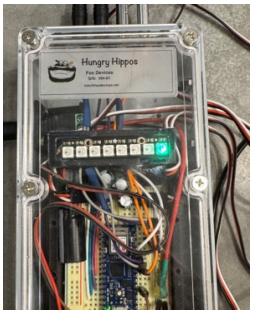
- A platform to hold the microcontroller
- A mounting panel for the four 3.5mm RCA jacks
- A tension plate for the servo cables

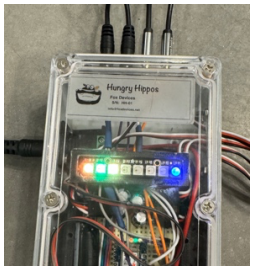
- A mounting clip and face plate for the power supply terminal
- A tray to hold the NeoPixel LED Stick to the case lid

Added LEDs: I added a NeoPixel strip with eight high-powered LEDs mounted underneath the clear case lid to provide visual status and add extra excitement to the gameplay.

One NeoPixel is always on to show the system is ready, and its color changes between blue and green depending on which of the two button modes is enabled (see below).

To make gameplay even more lively, four NeoPixels flash brightly in the colors of the hippos — orange, blue, yellow, and green — whenever a player presses their corresponding adaptive switch.





Added Button Modes: There are two modes that control how long the hippos remain extended when the servo activates:

• **Mode 1**: The hippo remains extended for a fixed duration of ½ second before automatically retracting. (This duration can be easily adjusted in the code.)

• **Mode 2**: The hippo remains extended as long as the player holds down their adaptive switch, retracting only when the button is released.

Any player can toggle between the two game modes by double-tapping their adaptive switch within ½ second (this timing can also be changed in the code). Mode 2 provides a little more direct control for the player, while Mode 1 may be easier for some children depending on their physical ability. An LED displays either green or blue to indicate which mode the game is currently in

Adaptive Parts List

Purchased Parts

Quant	Item	Typical Cost	Notes	Vendor
1	Hungry Hippo Game	$25	older version. See note below	ebay, Amazon, Etsy
1	Half-size breadboard (400 tie points)	$3	Larger sizes are fine too	Elegoo on Amazon
	Arduino Nano 33 IoT	$26	Microcontroller. See note below	Arduino
4	Servos - Hitec HS-645MG	4 x $35	Hitec is a good brand. There are several other good brands.	Amazon, Servo City
4	Servo mounting brackets	$17 for 4	Optional. Provides a sturdy mounting option. Zip ties may also work.	Amazon
4	12" servo wire extensions	$6	3-pin servo extension cable. Female to male. Optional but will provide more mounting/configuration options.	Amazon, Servo City
1	Large capacitor - 4700uF 16V 13X25mm Electrolytic Capacitor	$8	Optional - to help smoothen power delivery if all 4 servos are moving at the same time. See section on design changes.	Amazon
4	Small capacitors- 470uF 16V 8x12mm Electrolytic Capacitor	$6	Optional - to help smoothen power delivery if all 4 servos are moving at the same time. See section on design changes.	Amazon
4	AdaptiveSwitcheswith 3.5MM RCA plugs	4 x $50	There are several vendors with varying prices. See notes.	Amazon
4	3.5mm Mono RCA jacks (female)	$8 for 20	There were pretty cheap but adequate.	Amazon
1	Power supply 6v-10A	$17	See notes.	Amazon
1	Screw terminal for 6V power supply	$2	May be included with power supply	Adafruit, Amazon
1`	ABS Plastic Junction Box	$10	6.2” x 3.5” x 1.8”. See notes.	Amazon
1	A package of Dupont cables (6")	$10	Optional but highly recommended for breadboard work	Amazon
1	Adafruit NeoPixel Stick	$6	Option but useful for game status	Adafruit
1	5V Buck Converter	$2	Drop power to 5V for NeoPixels	Amazon
1	Assorted 24 awg Wire (solid core)	$10	For connecting things on the breadboard	Amazon
1	18 awg wire (stranded core)	$13	For connecting the power supply. Only need a few inches.	Amazon, Lowe's
1	Single Row Pin Header Strip 40Pin 2.54mm Pitch	$6	Optional. See notes.	

	Stainless steel screws and lock nuts	$10	For fastening 3D printed parts to enclosure. M3-0.5x12mm works well of the thickness of the enclosure listed.	Amazon
	An assortment of 3D printed parts	free	.STLs provided. To mount the electronics to the enclosure.	Custom fabricated as needed

STL Files for 3D Printed Parts

MCU tray	Servo arm extension	Switch jack plate	Power jack plate	Servo cable clamp	LED Holder
					

(These files are in the STLs folder in the adaptive-hungry-hippos GitHub repository)

Notes on Parts List

Cost: The project can get a bit pricey, mostly due to the servos and adaptive switches. The good news is that the switches — with their standard 3.5mm RCA jacks — can easily be unplugged and reused on other games. The last 6 or 7 items on the parts list are general-purpose supplies you might already have on hand, and they'll be useful for future projects too.

Power Supply: You may already have a compatible 6V power supply from another device — many use the same standard-sized barrel jack. But don't skimp on current capacity. Trying to make an underpowered supply work is usually more frustrating and time-consuming than it's worth, and in some cases, it could introduce safety concerns.

Game Version: Older versions of *Hungry Hungry Hippos* are much easier to modify. Look for ones that have a flat-sided, integrated marble tray, which makes mounting the servos a lot easier. These older versions are often available on Amazon, eBay, and even Etsy.

Adaptive Switches: Most adaptive switches will work, as long as they are *normally open* (NO) and use a 3.5mm jack. Prices can vary significantly, so it's worth shopping around. One site with a wide

selection is Adaptive Tech Solutions. Just be sure to check the wiring requirements, as some switches may differ from the ones used in this project.

Choosing a Power Supply: The power supply must at least handle the servos, since microcontrollers typically can't provide enough current. You can either power the microcontroller separately via USB or from the same supply. I chose to use a single 6V / 10A supply for both.

- The Arduino Nano 33 IoT can tolerate 5-18V on the VIN pin (anything higher risks damage), and the servos operate best between 4.8–6V.

- Because servos run faster at higher voltage, I went with 6V for snappier gameplay.

- Each servo has a stall current of about 2.5 A, so four servos would draw 10 A if they all stalled simultaneously. Add another 0.5 A for the microcontroller and NeoPixels, and you're right at the edge of a 10 A supply.

- A 6–8 A supply probably would've been fine, since the servos don't actually stall in this implementation and won't draw their full rated current. But 10 A supplies were easy to find on Amazon, and having a bit of headroom never hurts.

- If you're using a smaller supply and experience brownouts or glitches when multiple servos activate, adding capacitors may help stabilize voltage.

Enclosures: There are tons of options for project cases, but I like the kind listed in the parts list — they come with clear lids. That makes it easy to check if the device is even on without opening the case. They're also waterproof, easy to drill/cut, and come in a range of sizes. I like to keep a few sizes on hand to find the smallest one that still leaves enough space for a clean, serviceable install.

Power-to-Breadboard: One challenge with breadboard designs is connecting external power securely. In this project, I soldered stranded 18 AWG wires to a pair of pin headers, which can then be plugged directly into the breadboard for a solid connection.

3D Printed Parts: Some of the 3D printed components help polish the look of the project by hiding rough drill holes or tool marks. But they also serve functional roles: holding the breadboard in place, securing the power terminal block, and neatly anchoring the servo cables where they enter the enclosure.

Tools Needed

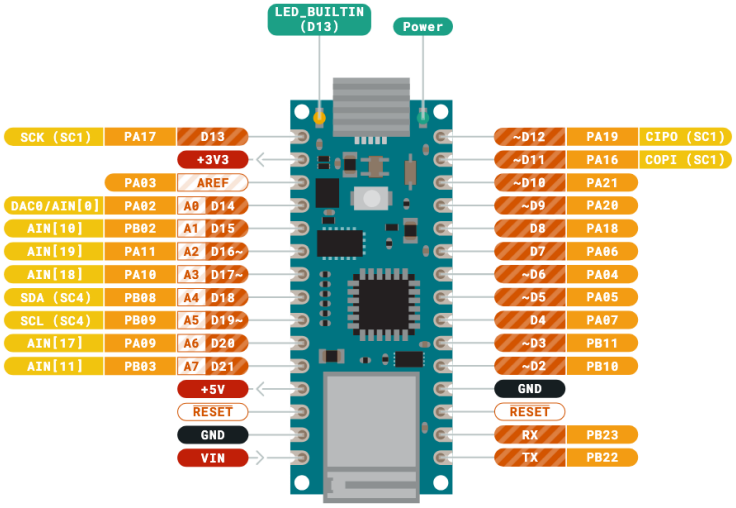
Computer – For writing code and uploading it to the microcontroller
USB cable – To connect the microcontroller to your computer
Arduino IDE (free) – Software to write and upload code
T**inkerCad** (free) – Software for viewing/modifying 3D print designs
3D printer – Or access to one through a friend, library, or maker space
Soldering iron – For attaching wires to headers and connectors
Small Phillips-head screwdriver – For servo and enclosure screws
Wire cutters/strippers – To prepare wiring for the breadboard and power terminals
Glue gun (optional) – For securing components, insulating solder points, or fixing mistakes
Dremel + drill bits – For mounting components and drilling case openings for switches, wires, etc.

Diagrams

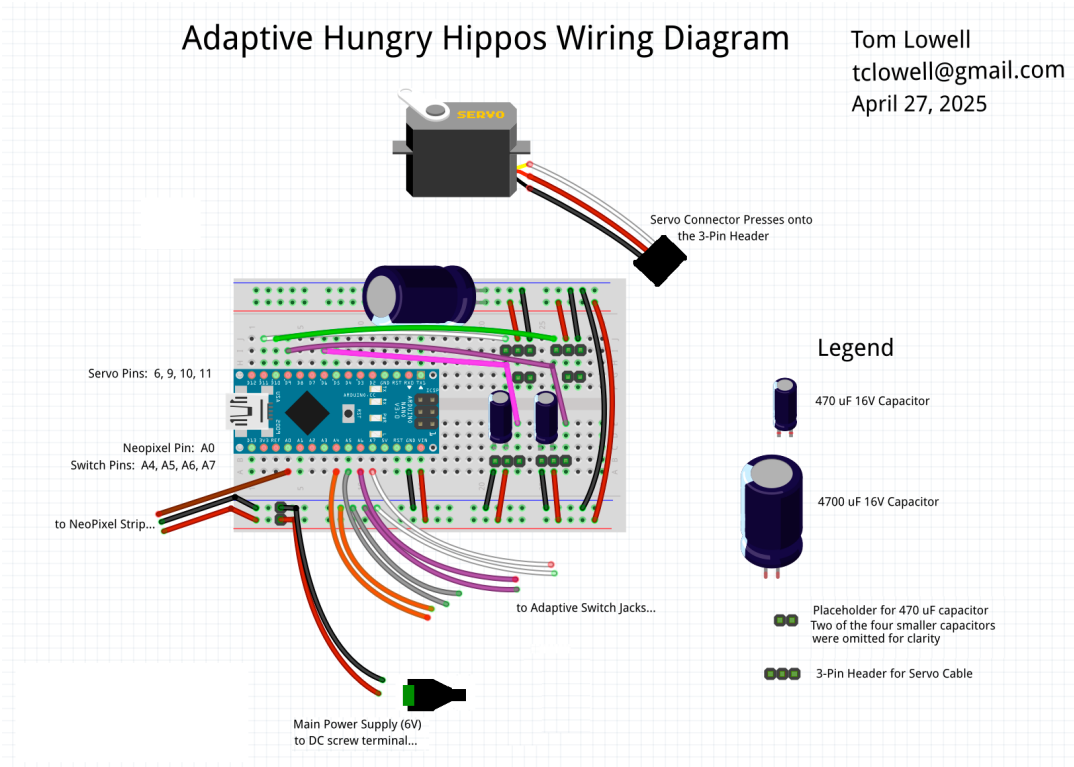
This writeup will reference the following diagrams.

Pinout Diagram for Microcontroller

Arduino Nano 33 IoT microcontroller:



Wiring Diagram



Microcontroller Setup

Before diving into soldering and wiring, it's smart to confirm that your Arduino development environment is working and that you're able to flash the microcontroller successfully. This includes installing the correct board definition, uploading a simple test sketch (like Blink), and checking the Serial Monitor output.

Doing this now helps avoid frustrating troubleshooting later on — especially when multiple things are connected and it's harder to isolate issues. Once this workflow is confirmed, you'll be in good shape to proceed with the physical build.

Steps:

1. **I**nstall the microcontroller (MCU) onto the breadboard with its USB connector positioned near the edge, as shown in the wiring diagram.
2. Open the Arduino IDE, and in the **Board Manager**, install the appropriate board definition for your device.
   → For the Nano 33 IoT, I installed support for the **Arduino SAM boards (32-bits ARM Cortex-M0+)**.
3. Connect the MCU to your computer using a compatible USB cable.
4. Flash the built-in **Blink** example sketch to confirm that your development environment is set up correctly and communicating with the board.
5. If the onboard LED is blinking as expected, open hippos.ino (located in the code folder in the adaptive-hungry-hippos GitHub repository) and upload it to the MCU.
6. Once the sketch is successfully uploaded, disconnect the USB cable from the microcontroller.

Build Instructions

These instructions cover a mix of coding, wiring, and physical assembly. I'm writing them from memory, so there may be a few small errors or omissions — think of this as a rough guide rather than step-by-step documentation.

While the goal is to make this approachable for beginners, some basic steps (like flashing the microcontroller) aren't covered in detail here, since there's already plenty of great info online for those topics.

The recommended workflow is:

1. Do all the soldering first
2. Build and test the system on your desk (breadboard stage)
3. Once it's working as expected, install the electronics into the enclosure and physically mount the servos to the game board.
4. Finally, complete the servo tuning and perform a final functional check.

This lets you troubleshoot and tweak before committing everything to the case or game.

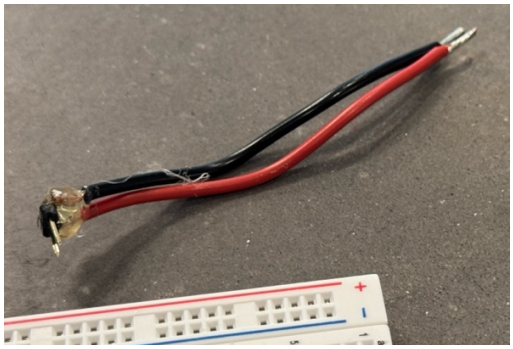
Soldering

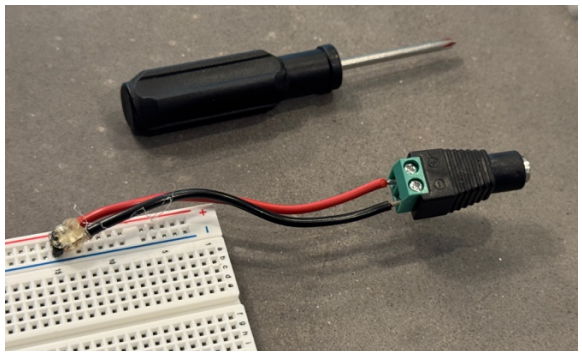
Three sub-assemblies need to be soldered.

Preparing Power Wires for Breadboard Connection: The goal here is to build a solid and reliable power connection to the breadboard.

Cut a pair of 6” wires (18 AWG stranded — one red, one black). Strip about 1/4” of insulation from each end. Clip off a pair of pin headers from a strip, keeping them together. Solder one red and one black wire end to the pin headers — make sure your connections are solid and clean.

On the other end of each wire, tin the exposed strands with a bit of solder to prevent fraying.





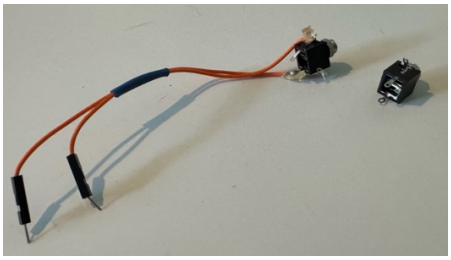
Insert these tinned ends into the screw terminal and tighten down. Set this assembly aside for now — it’ll plug into the breadboard later and connect to your external power supply.

Wiring the 3.5 mm RCA jacks for adaptive switches: In this step, you’ll solder wires to the four 3.5 mm RCA jacks so they can connect easily to the breadboard. You can use any 24 AWG wire, but I recommend using *Dupont wires* — the male ends plug directly into a breadboard and the wires are flexible, making them easy to manage inside the enclosure later.

Grab 8 Dupont wires. To reduce confusion, use matching color pairs (e.g., 2 blue, 2 yellow, etc.). Trim off one end of each wire (the end without the male pin), then strip about 1/2” of insulation and twist the strands neatly.

If your RCA jacks have 3 terminals (as in this build), you’ll only use two. One of them is a “trim” tab, used to detect when a plug is inserted — don’t use that one. Use the *signal* and *ground* terminals instead.

Thread the stripped wire ends through the correct tabs and twist the wire back on itself. Then solder each connection and ensure they’re secure.

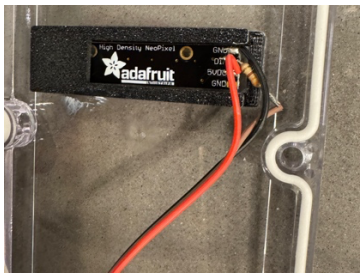


Since the terminals are small and the wires are thin, it’s easy to damage them during assembly. I recommend adding a dab of hot glue to each solder joint to protect it from strain or bending. Once all four jacks are wired, set them aside for now.

NeoPixel Leads:

The NeoPixel stick requires three wires: **Data In**, **5V**, and **GND** — all of which need special handling. Since the LEDs will be mounted on the lid of the enclosure, you'll want at least 8–10 inches of lead length so the lid can be removed and set aside without disturbing the wiring.

• The **Data In** wire should have a **300–500Ω resistor** soldered **right at the point** where it connects to the NeoPixel stick to protect against voltage spikes.



• Both the **5V** (V+) and **GND** wires must be routed through a **buck converter** to drop the voltage from your **6.5V power supply** down to a safe **5V** for the NeoPixels.

Wiring Steps for. Buck Converter:

1. Take one **red** and one **black** Dupont wire (8–12" long). Cut off the pin ends, leaving about **4"** of wire on each piece.

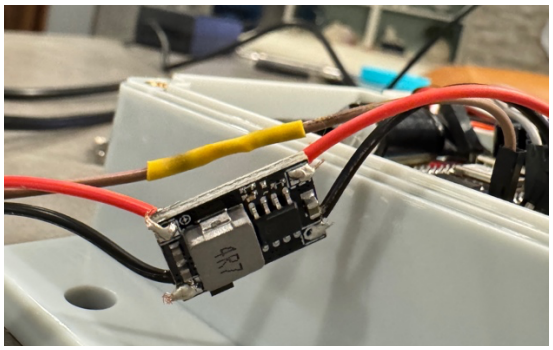
2. Strip about **¼"** of insulation from each cut end.

3. Solder the red wire to **Vin** and the black wire to **GND_in** on the buck converter (input side).

4. Strip both ends of the two remaining Dupont pieces.

5. Solder one wire each to **Vout** and **GND_out** on the buck converter (output side).

6. Finally, solder the other ends of these output wires to the **5V** and **GND** tabs on the NeoPixel stick.



Wiring/Breadboard Assembly

Power Distribution

1. Connect the two power rails on either side of the breadboard using two red and two black 24 AWG wires — one pair at each end. (Red to red, black to black.)

(See wiring diagram for reference.)

2. Using a short red wire (24 AWG), connect the breadboard channel with the BAT pin on the microcontroller to the positive power rail.

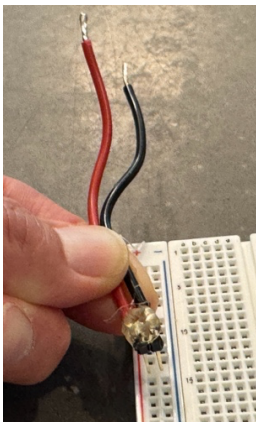
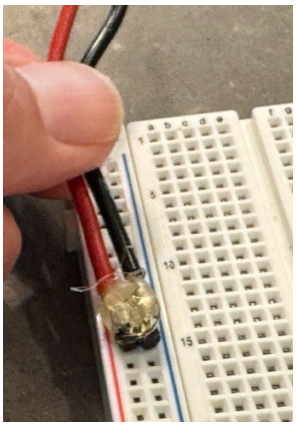
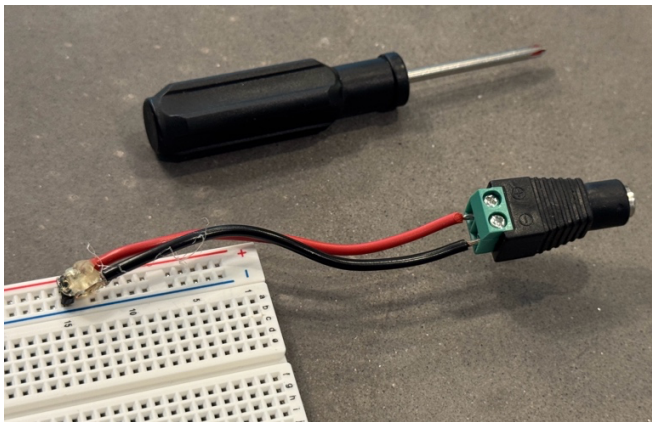
3. Using a short black wire (24 AWG), connect the GND pin channel to the negative power rail.

4. Insert the header pins from your screw terminal assembly (the one you previously soldered) into the breadboard. The corners farthest from the MCU are convenient and leave room for the enclosure later.

   → Be sure to insert the red wire into the positive rail and the black wire into the negative rail.

   → Make sure the connection feels firm and won't wiggle loose.

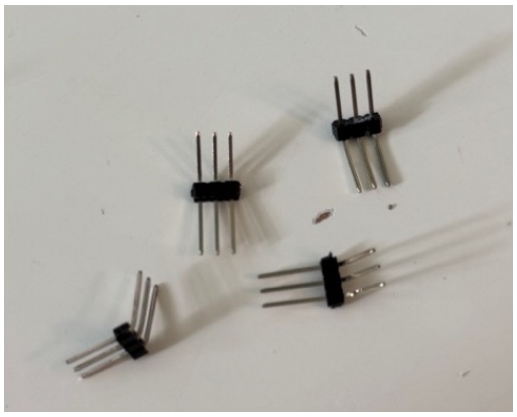
5. Connect the screw terminal to the RCA jack on your power supply cord — but do not plug in the power yet!

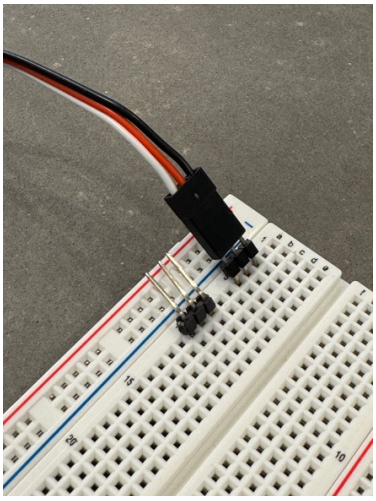




Servos

You may find it helpful to complete the following instructions using just one servo and one adaptive switch first. It's much easier to debug mistakes and loose connections in a simpler setup. Once that's working and you have a solid understanding of the system, you can add the remaining components. The adaptive switches were added last in this build because they're relatively easy to wire with Dupont connectors, and delaying their installation helped keep things tidy while setting up the servos.

1. Insert four sets of 3-pin headers into the breadboard at the opposite end from the MCU.





Note the wire orientation on each servo cable:

→ Black = Ground
→ Red = Power
→ Yellow = Signal

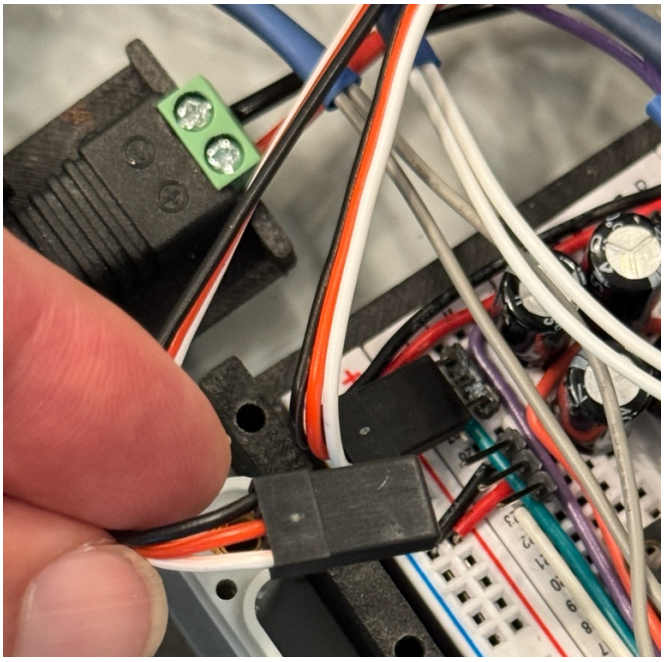
2. Run short red and black 24 AWG wires from the power rails to the appropriate rows for each pin header set (red to power, black to ground).

3. Run a longer signal wire (24 AWG) from each servo header's signal pin (where the yellow wire will go) back to the appropriate MCU pin.
   It helps in debugging to use different colors for each pin. In this build, we used the following colors and GPIO pins:

   • Purple → Pin 5
   • Pink → Pin 6
   • Green → Pin 9
   • White → Pin 10

This image is a little crowded, but it shows a 3-pin servo header inserted into the breadboard, with black, red, and white wires connected to the same breadboard rows. (The white wire in this image happens to correspond to servo pin 10 — your colors may vary.) The orientation matches the servo extension cable that will be slid onto the header pins.

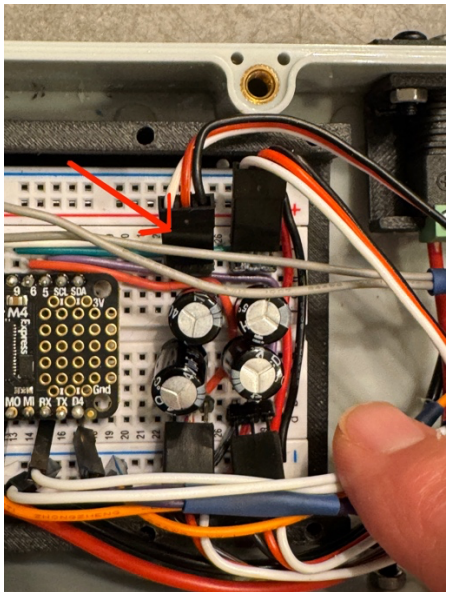
(refer to the Wiring Diagram)



4. Verify the pin mappings match what was specified in the Arduino sketch:

```
const int servoPins[NUM_HIPPOS] = {6, 9, 10, 11};    // Servo output pins
```

5. Plug a servo extension cable onto each 3-pin header paying attention to polarity. Make sure the black (ground) wire aligns with the ground rail on the breadboard. Then connect the other end of the servo extension cable to each servo's 3-wire connector.



At this stage, the servos are simply connected to verify functionality. Final adjustments to their range of motion will be done after they're mounted to the game.

Capacitors

This design includes one large capacitor across the breadboard's power rails and one smaller capacitor across the positive and negative lines near each servo. Their purpose is to smooth out voltage dips and spikes that can occur when servos draw sudden bursts of power. This is especially common with undersized or budget power supplies.

The power supply listed in the parts list should be sufficient, so these capacitors are optional. This design was tested successfully without them. However, if you're using different components (e.g., servos or power supply), or just want to build in some extra stability, including capacitors is a good practice.

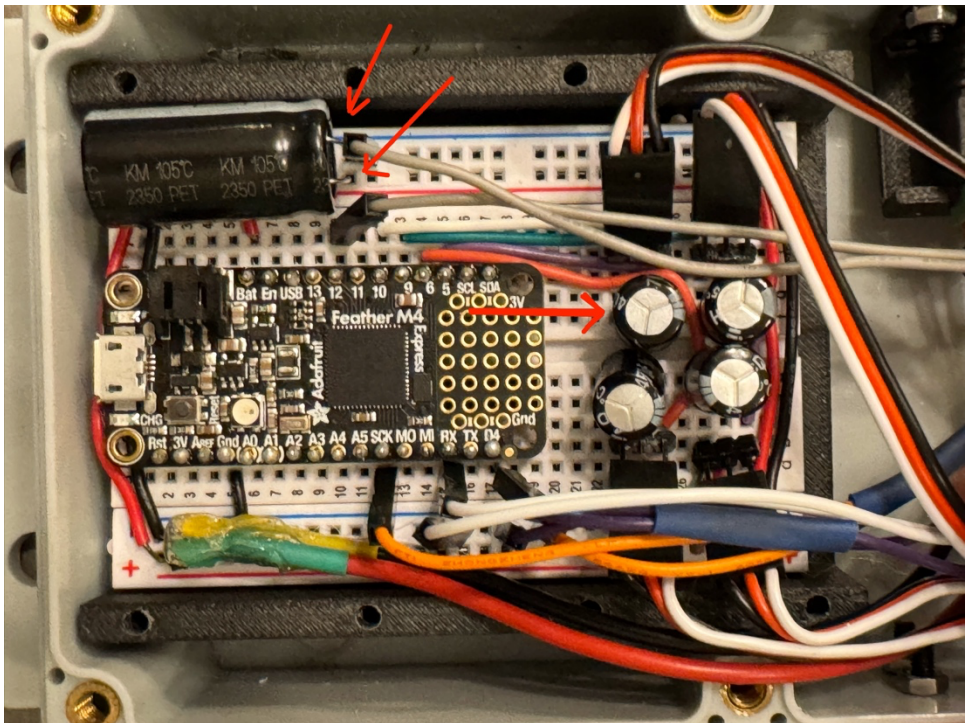
Steps:

1. Trim the capacitor leads down to a length that will insert cleanly into the breadboard, but leave enough for a good connection.
2. Insert the leads into the power rails:
   - Place one large capacitor across the main power rails of the breadboard.

- Place one small capacitor near each servo header, bridging its power and ground rows.

⚠️ Note: Capacitors are polarized. The negative lead is usually marked with a white stripe on the body. Be sure to connect it to the ground rail.

(See black circles in the wiring diagram for suggested placement.)



Adaptive Switches

Using the four assemblies you soldered for the adaptive switches…

1. Connect the four 3.5 mm RCA jacks (with their soldered wire pairs) to the breadboard:

    - Plug one wire into the correct GPIO pin channel for each switch: Pins **A4, A5, A6, and A7** correspond to the code in hippos.ino.

    - Plug the second wire into the ground rail. It's helpful to place this ground wire near the associated GPIO pin to keep things tidy.

      (See wiring diagram.)

2. Verify the pin mappings match what was specified in the code:

```
const int adaptiveSwitchPins[NUM_HIPPOS] = {A4, A5, A6, A7};  // Adaptive switch input pins
```

3. Temporarily plug one adaptive switch into each RCA jack for testing.

NeoPixel LEDS (optional)

⚡ **Note on NeoPixel Power:**

NeoPixels are sensitive to voltage. If you're using a 6V power supply to drive your servos and also plan to run NeoPixels from the same supply, you'll need to step the voltage down to 5V.

While some builders use BAT diodes to drop the voltage, the safer and more reliable option is to use a **buck converter** to provide a clean 5V output. These are inexpensive, easy to find, and help avoid mysterious flickering, color shifting, or outright failure of the LED strip.

Using the NeoPixel assembly you built during the soldering steps:

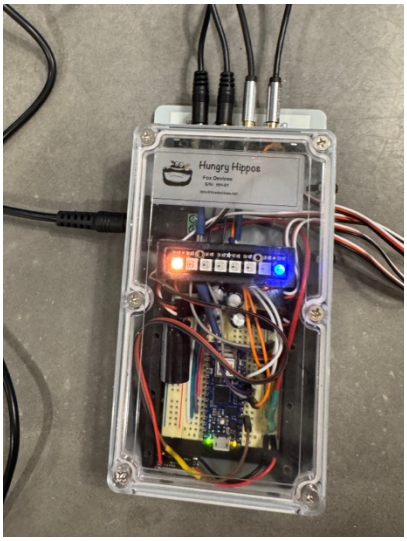
• Plug the **power** and **GND** Dupont pins into a convenient spot on the breadboard's power rails (be sure to observe correct polarity).

• Plug the **data wire's Dupont pin** into the breadboard row corresponding to the NEOPIXEL_PIN defined in your code. In this case, **pin A0**:

```
#define NEOPIXEL_PIN A0
```

Finally, glue the **3D-printed NeoPixel tray** to the underside of the case lid. Position it so it doesn't block the onboard LED or interfere with other wiring. Slide the NeoPixel Stick into the tray with the LEDs facing outward thru the case lid.

When reattaching the lid, tuck the wires neatly into the case, taking care not to pinch any of them.



Power Up and Bench Test!

1. **Double-check all connections before powering up.** Pay close attention to:
   - All **red wires** should connect to the positive power rails.
   - All **black wires** should connect to the negative power rails.
   - The **BAT** pin on the microcontroller should connect to the positive rail.
   - The **GND** pin on the microcontroller should connect to the negative rail.

2. Once everything looks correct, **plug the power supply into a wall outlet**.
3. The MCU's onboard LED should light up or blink (depending on your code). Verify this both when using only the power supply and when powering the microcontroller with both the power supply and a USB-C cable simultaneously.

4. Tap each adaptive switch and confirm that the corresponding servo responds**.**

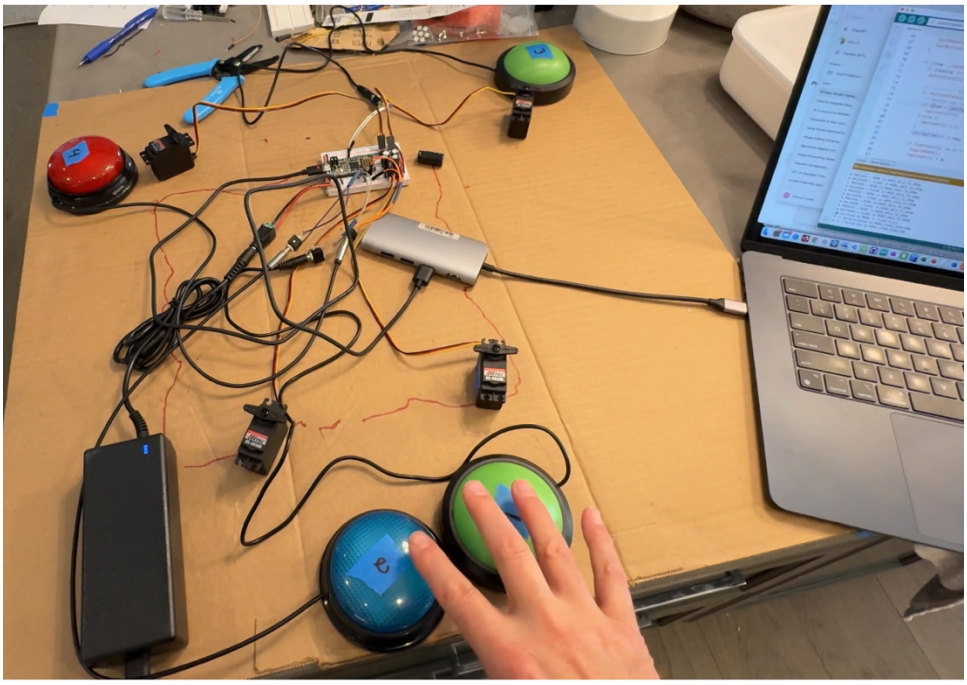
   - Make note of any that don't — this is your chance to catch wiring mistakes or loose connections before you mount anything permanently.

5. Use the Serial Monitor in the Arduino IDE to help debug.

   - The code includes some helpful Serial.println() statements that identify when a button is pressed and which servo is moving.
   - Open the Serial Monitor, press each switch, and confirm that both the servo action and print output match your expectations.

🛠️ *Tip:* Feel free to add your own Serial.println() statements anywhere in the code to trace behavior, especially if you're troubleshooting a specific component.

6. If the servos are responding, you're in great shape!
   - Their exact range of motion will be fine-tuned after the servos are physically mounted on the hippos.
   - This is just a first test to confirm the wiring and logic are working.



Troubleshooting

This isn’t meant to be a complete Arduino troubleshooting guide—there are already lots of great ones out there. This list just highlights common issues that came up during this specific build, in the hopes of helping you get unstuck faster.

1. No LED activity on the MCU

   Most likely a power issue: double-check that your power supply is properly connected and that the **BAT** and **GND** pins on the microcontroller are correctly wired to the power rails.

   Alternatively, the code might not have been successfully flashed. Try reflashing the board with a known working sketch, like Blink, to verify the connection and upload process.

2. Some servos aren’t responding (but others are)

   The most common cause is a wiring issue:

   - A loose or misaligned Dupont wire.
   - A pin connected to the wrong breadboard row.
   - A mismatch between the pin numbers in the servoPins[] array in hippos.ino and the actual physical connections.

   Try swapping a known working servo into the slot that isn’t responding. If it doesn't work there, it’s a wiring or code issue, not the servo itself.

3. Buttons not triggering servos

If a button press doesn’t cause any servo movement, it could be the switch wiring or code logic.

Add Serial.println("Button X was pressed") inside the appropriate logic block in loop() to verify that the microcontroller is seeing the button press. If you see this message in the Serial Monitor, the switch wiring is working and the issue lies further down in the servo logic or wiring.

4. Servos are vibrating or chattering

   This usually means they are trying to hold position but aren’t quite getting there—often due to inconsistent voltage or bad ground connections.

   Make sure all grounds are tied together (servos, MCU, and power supply).

   Even if you’re powering the microcontroller via a USB cable, the GND pin still needs to be connected to the negative rail on the breadboard so it shares a common ground with the servos.

   Double-check your supply voltage: analog servos like the HS-485HB are happiest near 6V.

5. Power brownouts or system resets

   If your servos all activate at once and the system cuts out or resets, your power supply may be underrated.
   - Check that your supply provides at least 6A at 6V (10A is better for headroom).
   - Adding bulk capacitors across the power rails and near each servo can help absorb current spikes.

Enclosure Assembly

We're not just building a sexy box to show off to all your friends. It's important to protect the components and also to anchor the wires that pass through the enclosure so they don't get pulled loose.

Installing the 3D-Printed Mounting Plates

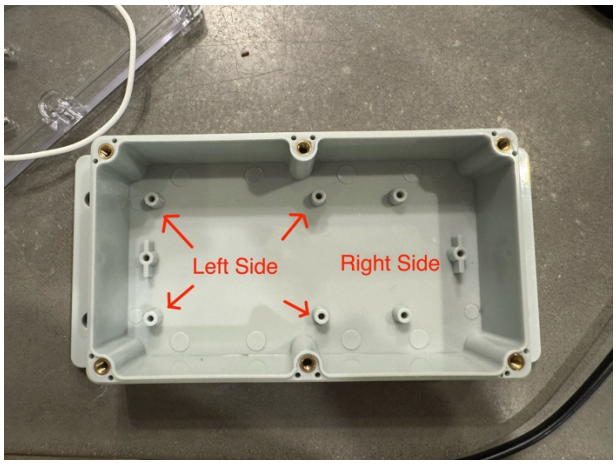
Follow these steps if you’re using the 3D-printed parts provided in this write-up:

The power jack plate and switch jack plate allow you to mount RCA-style jacks, making it easy to plug and unplug the power cable and adaptive switches. The power jack plate also helps ensure a sturdy power connection with the internal components. The two servo clamp plates serve two purposes:

1. They provide a clean slot for routing the servo extension cables into the case.
2. They create strain relief to prevent accidental yanking of the servo headers during use.

Note: These steps refer to the separate servo wire extension cables listed in the parts section — *not* the cables attached directly to the servos. Using extension cables allows the servos to be easily disconnected from the enclosure for storage or transport.

Mount the MCU tray into the left end of the case.
The recommended enclosure (see parts list) has asymmetric screw holes, so take care to align the correct ones with the tray.



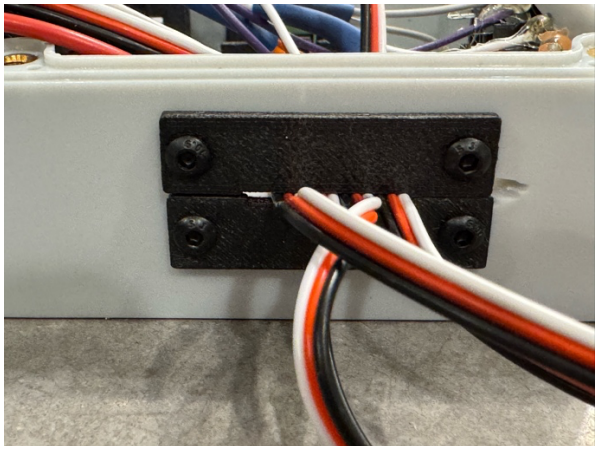
Plan the layout of the remaining plates at the right end of the case, away from the MCU.
This helps keep your wiring organized and minimizes cable crossing.

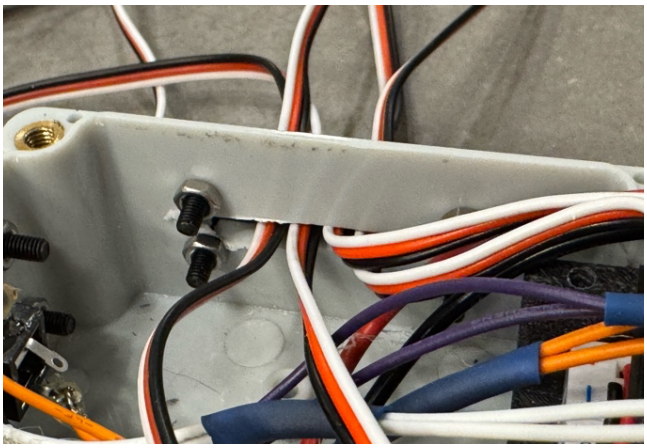
- The two servo cable clamp plates should be mounted on one of the side faces, nearer the MCU because the servo wires are thicker and less flexible.
- The adaptive switch jack plate (which uses thin Dupont wires) can be installed on the end face of the case.
- The power jack plate can go on the remaining side.

Mark the mounting positions for each plate with a Sharpie before drilling.

Drill holes and mount the plates using small screws or bolts that match the plate dimensions.
In addition, you'll need to cut a horizontal slot in the case to allow the servo extension cables to pass through — a Dremel works well for this. Make sure the slot is wide enough for the cable connectors to fit through.

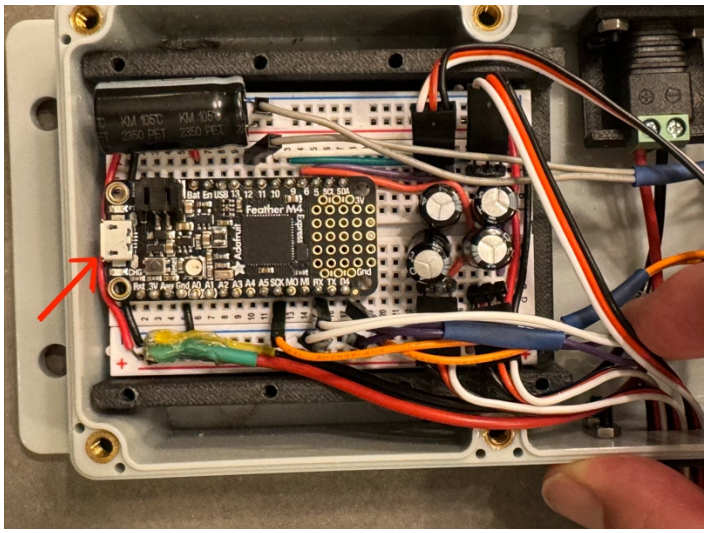
Pass the 4 servo extension cables through the slot and then install the two servo cable clamp plates with four appropriate (3M) nuts and bolts. Check the cables for snugness. If they're loose, try loosening the bolts and sliding the plates closer together before tightening.



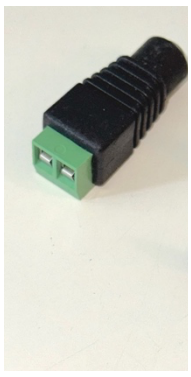


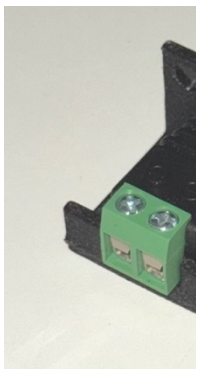
Installing the Components in the Enclosure

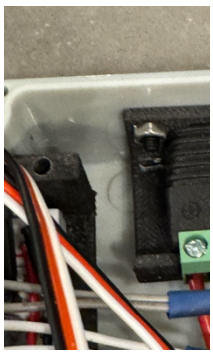
Place the breadboard and MCU into the MCU tray, with the USB port on the MCU facing the outer edge of the case.

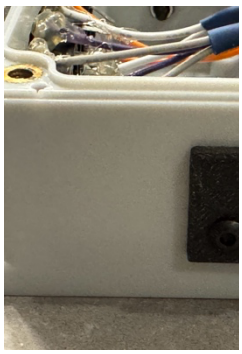


Insert the power supply screw terminal into its holder, positioning it so the jack protrudes through the hole in the case. If you haven't already done so, screw the tinned ends of the red and black power wires into the green terminal block, taking care to match the correct polarity. Then insert the header pins of the power wires back into the appropriate positions on the breadboard's power rails. Make sure all connections are snug and secure.

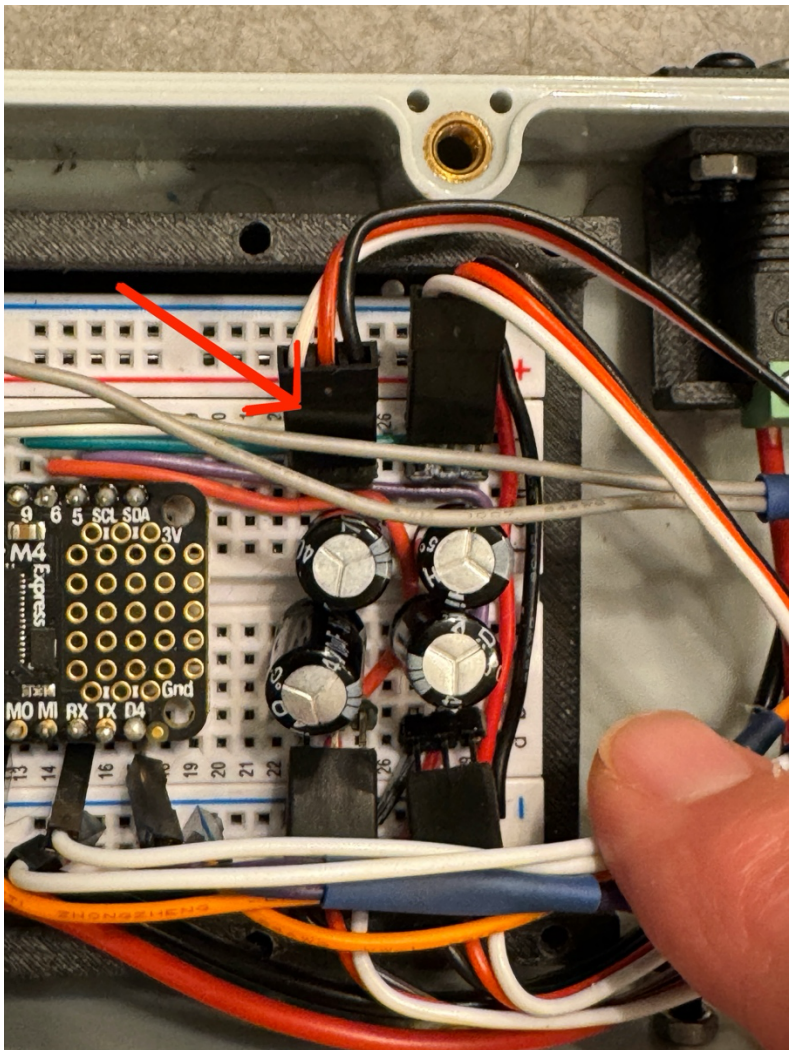






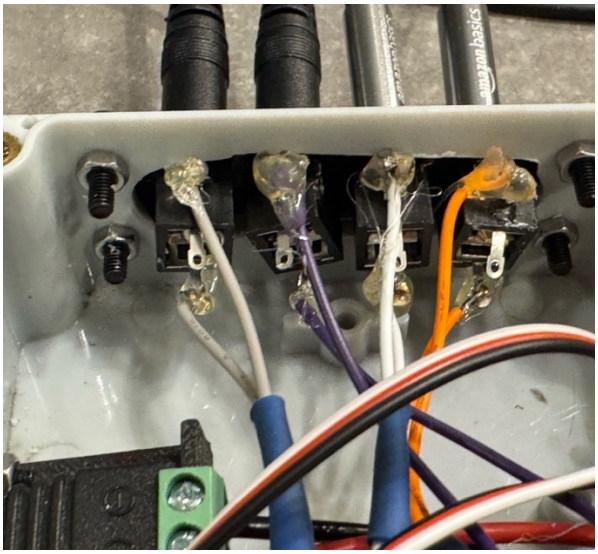


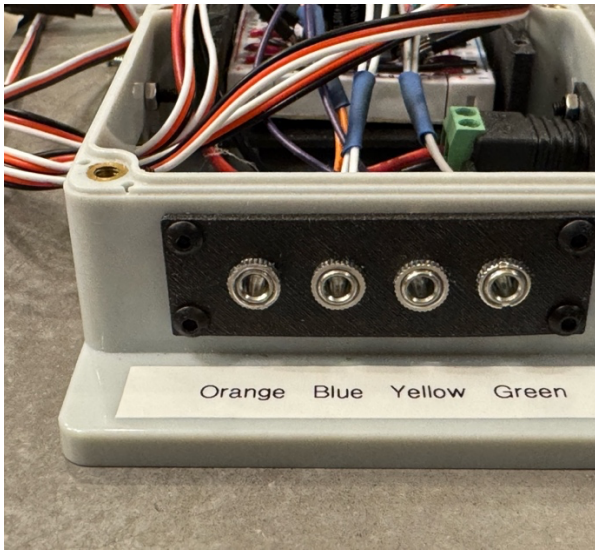
Connect the servo extension cables to the servo header pins on the breadboard, taking care to match the polarity.

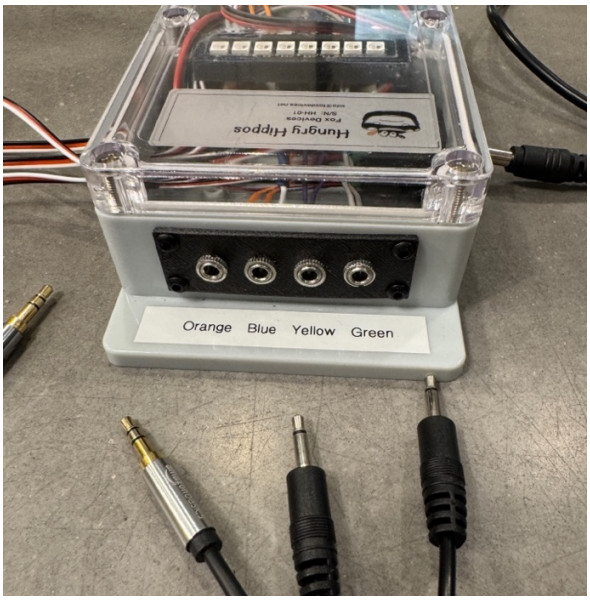


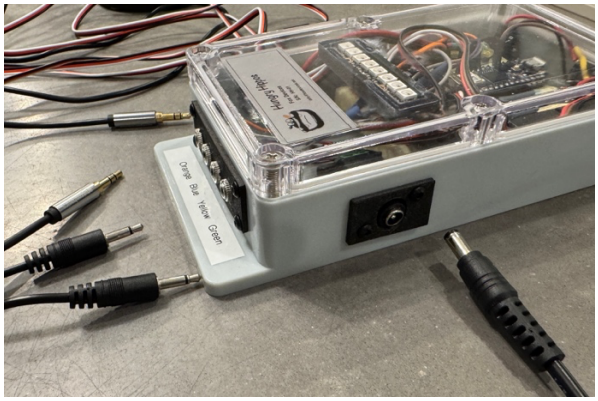
Mount the RCA jacks into the case:

- Remove the threaded ring from each jack.
- From the inside of the case, insert each 3.5mm RCA jack through a hole in the switch jack plate.
- From the outside, reattach the threaded ring to hold the jack securely in place.
- Finally, plug one adaptive switch into each RCA jack.









This is a good time to revisit the testing steps in the “Power Up and Bench Test” section to confirm everything is still working correctly before finalizing the servo adjustments.

Mounting the Servos to the Hippos

Mount the servo bracket onto the flat face of the marble tray using two M3 screws and nuts. The exact position isn’t critical, but the servo shaft should be oriented to the left.

Install the servo into the bracket using the screws that came with it, or M3 screws and nuts if preferred.

Next, slide the double-sided servo horn (included with the servo kit) onto the servo shaft. Press it on firmly, but don’t screw it in yet.

Mount the red 3D-printed hippo servo arm onto the servo horn and attach it using the main screw that came with the servo. The screw should pass through both the 3D-printed part and the servo horn, securing them to the servo shaft. In this build, the included servo screws were just long enough to fasten both parts together securely.

To lock the rotation between the printed arm and the horn, you can optionally use one or two small servo screws at the ends of the 3D-printed part.

Drill a small hole in the end of the hippo's black lever, but don't connect it to the servo arm yet. That step will be completed after verifying the servo's rotation angles while the code is running.

Firmware.

If you haven't already installed the firmware, you'll find it in the hippos/hippos.ino file located inside the code folder of the adaptive-hungry-hippos GitHub repository. Follow the steps described in the *Programming the Microcontroller* section to upload it before continuing.

The firmware runs automatically as soon as the game is powered on — no action is required to start. The switches will be active, and the servos will be ready. The onboard LED lights up whenever any switch is pressed.

The game supports two button modes:

- **Mode 1**: Pressing a button causes the hippo to extend for a fixed duration (0.5 seconds), regardless of how long the button is held. The hippo then retracts automatically.
- **Mode 2**: The hippo remains extended for as long as the button is held, retracting when released.

You can switch between modes by double-pressing any button quickly (within 0.5 seconds).

For more detail, refer to the comments in the Arduino code (hippos.ino).

The firmware also includes a tool to help you fine-tune how far each servo rotates, so you can make sure they're moving through the best possible range.

Tuning the Servos Through the Firmware

Now that the servos are mounted, turn on the game, press one of the adaptive switches, and observe the servo arm's range of motion. It should move from roughly the 10:00 position when closed to about the 7:00 position when open (with the hippo head fully extended). If that's not happening, it's worth adjusting the code now — before physically connecting the servos to the hippos. If the travel is way off, the servos may fight against the hippo levers, causing chattering, stress, or even damage.

The range of motion is controlled by two lines in the code.

```
int SERVO_OPEN_POS  = 0;   // Servo position when hippo head extends out. Change this as necessary.
int SERVO_CLOSED_POS = 75; // Servo position when hippo head returns. Change this as necessary.
```

The game's firmware includes a built-in feature that lets you try new servo open and closed positions without needing to re-upload the code each time. This is handy during setup, especially if you need to tweak how far a hippo moves.

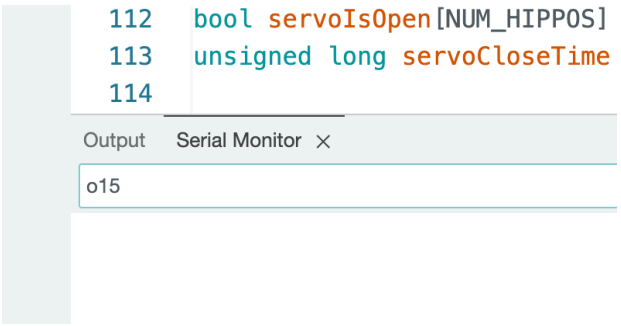
How It Works:

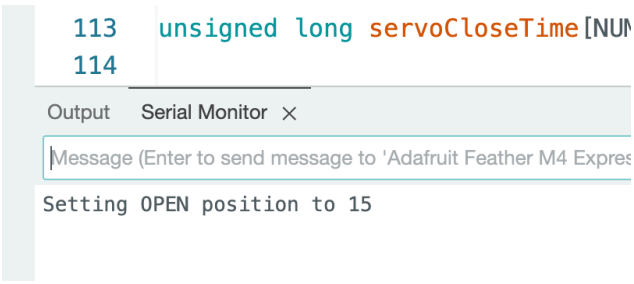
You can send a short command through the Serial Monitor in the Arduino IDE to temporarily update either the open or closed angle of all four servos.

Command Format:
- Oxx — Set the **Open** position
- Cxx — Set the **Closed** position

Where xx is a two-digit angle (from 00 to 99).

For example:
- O15 sets the open position to 15 degrees.
- C75 sets the closed position to 75 degrees.





Try it out:

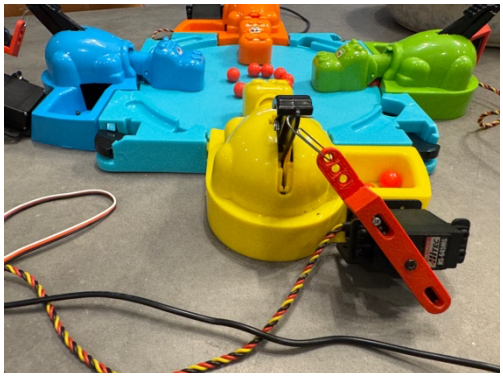
1. Connect the game box to your computer using a USB cable.
2. Open the Arduino IDE.
3. Select the correct board and port (Tools > Board and Tools > Port).
4. Use the Serial Monitor to send commands that adjust the open and close positions. Try different values until the servo travels roughly between the 10:00 (closed) and 7:00 (open) positions.
5. Once you're happy with the movement, update the SERVO_OPEN_POS and SERVO_CLOSED_POS values in hippos.ino to match your new settings.

Connecting the Servo Arm to the Hippo Lever

Once you've made a rough pass at tuning the servo open and closed positions (see *Tuning the Servos Through the Firmware* above), it's time to connect each servo arm to its corresponding hippo.

This build used a medium-size (#1) paper clip bent into shape to form the linkage between the red 3D-printed servo arm and the hole drilled into the black plastic hippo lever. It worked well, but thin wire or even string can also be used depending on what you have on hand.

Make sure the connection is secure but not rigidly tight — the hippo lever needs to swing freely without binding.



Final Tuning After Connecting Servos

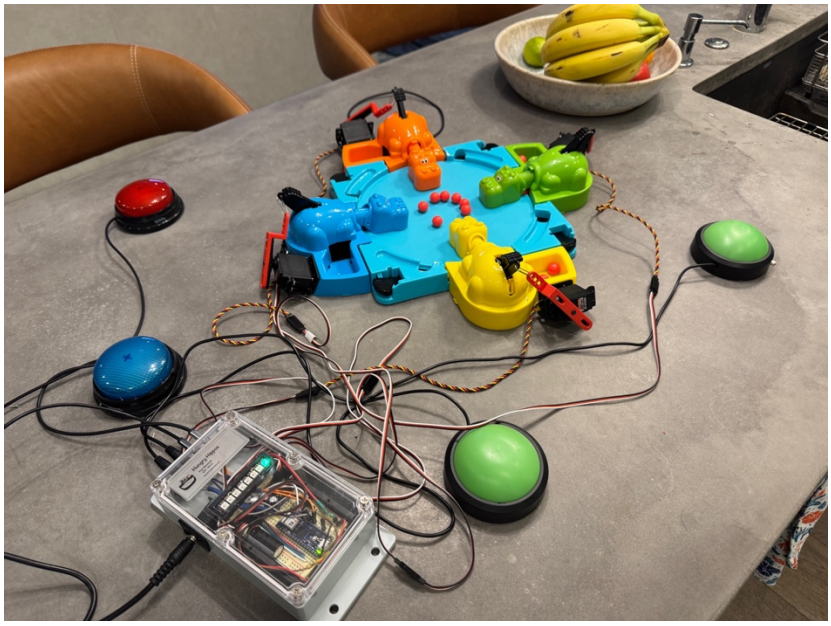
Once the servos are physically connected to the hippos:

- Run another round of servo tuning using the Serial Monitor.
- Test the action of each hippo to ensure it **snaps forward cleanly, grabs a marble, and pulls it back reliably**.
- Adjust the open and close angles if needed to fine-tune this motion.

⚠️ **Note:** By default, all four servos use the same open and close angle values. This simplifies setup and works well if the servo linkages are mechanically similar. However, if your linkages differ or one hippo behaves differently, you may want to tune each servo individually.

Optional: Individual Servo Tuning: The firmware can be modified to support separate open and closed positions for each servo. A short note in the Arduino code explains how to make this change by replacing the single SERVO_OPEN_POS and SERVO_CLOSED_POS values with arrays — one for each hippo. This gives you finer control and can help maximize gameplay consistency. See the comment near the variable declarations in the code for guidance.

That's it!



Reusing This Design for Other Games

The approach used to adapt *Hungry Hungry Hippos* can be applied to many other games that require pushing, pulling, tapping, or pressing actions. This project isn't just about modifying one toy — it's about showing how accessible, open-source hardware like Arduino can be used to reimagine interactivity for kids with limited mobility.

What Can Be Reused:

Once you've wired up adaptive switches and controlled a servo, you've got the foundation to build dozens of other accessible toys. The same basic ideas can be reused across many different builds:

Adaptive switch support: Any toy that responds to a mechanical input can be adapted to work with a switch — and the same switch logic and wiring applies across builds.

Servo control framework: Whether you're moving a hippo's head, turning a crank, or lifting a lever, servos can be adapted to many mechanical motions. This same servo code structure (timed vs. held) is highly reusable.

Power and Enclosure Design: The power planning, breadboarding, and enclosures used here are broadly applicable and easy to adapt.

Extend the Concept:

- Add sound or light feedback
- Use multiple modes (like timed vs. held)
- Integrate with speech buttons or text-to-speech boards
- Use sensors instead of mechanical switches (camera, proximity, tilt, even eye-tracking someday!)

You can extend this further by adding sound effects, light feedback, or using other kinds of sensors instead of mechanical switches (camera, proximity, tilt, even eye-tracking). This is the real value of the Arduino ecosystem — approachable electronics that let anyone with an idea turn imagination into action.

Changes During the Build

This section describes a couple of major design changes that were required during the build.

Servo Selection

For the servos, I originally started with the **HS-485HB**, which looked good on paper. They almost worked — but weren't quite strong enough to complete the final bit of the needed range of motion. Despite trying to really dial in the rotational geometry — experimenting with different designs for the servo arm extensions and carefully tuning the exact degrees of rotation — I realized the HS-485HBs still wouldn't deliver the snappy gameplay I wanted.

That realization was a real "oh shit!" moment. After investing so much time already, it wasn't clear if I'd be able to find a servo strong enough to make the game work — and for a moment, it genuinely felt like the entire project might not be doable. Fortunately, I found the **HS-645MGs**, which did the trick. The gameplay isn't super fast, but it's fast enough for the hippos' heads to snap up and lift — which is critical for actually grabbing the marbles.

Power Supply Challenges and Microcontroller Change

A major goal of the project was to not skimp on power delivery. There are three components — servos, LEDs, and the microcontroller — that all have different power requirements. My goal was to use a single, more-than-adequate power supply to ensure stable operation and to keep things simple with only one item to plug in. This requirement meant that the microcontroller needed to be powered through the breadboard, not via a USB cable.

Originally, I used a more modern Arduino-compatible board: the Adafruit Feather M4 Express, assuming it could be powered through the breadboard's BAT pin. But late in the project, I realized I had misunderstood the Adafruit board's power management. The BAT pin is designed for charging

or powering the board only if a battery is connected — not for standalone external power. Without a battery, the board would only boot when a USB cable was plugged in — an unacceptable requirement for this build. (Up until that point, I had done all testing over USB, so the problem wasn't obvious until near the end.) This was a gut punch moment: I thought I was finished, but it now looked like a core piece of the design might not work. Fortunately, I was able to switch to the Arduino Nano 33 IoT, which, like most Arduino-brand boards, provides a VIN pin connected to an onboard voltage regulator. This allows direct connection to external power supplies (typically 5–12V, sometimes up to 5–18V). The swap was technically simple, but the change was far-reaching: it required rewiring the breadboard, updating the Arduino code for different pinouts, re-creating the wiring diagram, revising this build guide, and repeating all the testing.

Summary: Choosing the Right Microcontroller for Power Needs

Adafruit boards are designed with strong battery support in mind. They typically offer features like:
• Automatic switching between USB and battery
• Overcharge protection
• Smart battery charging circuits

Arduino boards, by contrast, usually lack built-in battery management — but most include a **VIN pin** connected to an onboard voltage regulator, allowing them to safely accept external supply voltages (typically 5–12V).

Bottom Line:

- If you want to power your project by battery, consider Adafruit boards first.
- If you want to power your project by external supply (like this project with servos and LEDs), Arduino boards are often the better starting point.

Questions or Feedback?

I'm always curious to hear how others use or adapt this project. If you spot a mistake or come up with a clever improvement, feel free to reach out.

Tom Lowell
tclowell@gmail.com
github repository: https://github.com/tlowell/adaptive-hungry-hippos