Magic Eight Ball Raspberry Pi Project

Project Overview:
The goal of this project was to build a functional Magic Eight Ball using a Raspberry Pi Zero W, a motion sensor, and a small OLED display. The device is housed in a 3D-printed Magic Eight Ball shell that can be shaken to reveal classic eight-ball answers.

3D Printing the Shell:

- **Printer Used:** Bambu Lab P1S
- **Software:** Bambu Studio
- **Model Source:** Pre-existing Magic Eight Ball model, modified in Bambu Studio
- **Modifications Made:**
    - Added space internally to fit an RPi Zero W, sensors, and wires
    - Designed a twist-lock mechanism to keep the two ball halves secured together
- **Printing Outcome:** Successfully printed two halves with secure twist mechanism

---

Internal Electronics Setup:

- **Initial Setup:**
    - Raspberry Pi Zero W
    - Make sure I2C is enabled on the Pi, then reboot.
    - Pi Sense HAT (for accelerometer)
    - 0.96" OLED Display
- **Initial Software:**
    - Used the sense-hat Python library to access the built-in accelerometer
    - Logic to detect a shake event and trigger a random Magic Eight Ball response
    - Displayed output on OLED using lib_oled96 and smbus2

Hardware Revisions:

- **Reason for Change:**
    - The Sense HAT was too large for the enclosure and overkill for just accelerometer functionality
- **New Sensor:**
    - Replaced Sense HAT with an ADXL345 3-Axis Accelerometer Module
    - Much smaller footprint

Software Changes:

- **Removed:** sense-hat library
- **Added:**
    - adafruit_adxl34x (for the ADXL345)
    - busio and board (initially tried for I2C access)
- **Problem Encountered:**
    - busio requires hardware I2C pins, which conflicted with PiSugar battery usage on the same GPIO pins

---

GPIO Conflict & Solution:

- **Problem:**
    - PiSugar S battery module uses GPIO SCL pin, causing conflicts with OLED and accelerometer
- **Investigation:**
    - Used i2cdetect to confirm address conflicts/missing devices
- **Solution:**
    - Switched to software-defined GPIO pins using dtoverlay in /boot/firmware/config.txt

- Reassigned ADXL345 sensor to software I2C pins (GPIO 27 for SCL, GPIO 17 for SDA)
- Installed Adafruit Blinka to use CircuitPython-style APIs
- Used bitbangio.I2C to define custom I2C pins
- OLED display remained on hardware I2C bus using smbus + lib_oled96

Final Behavior:

- On boot, the OLED displays a splash screen
- When the device detects ~2G movement (shake), it randomly displays a Magic Eight Ball response on the OLED
- Device is powered by PiSugar S 1200 mAh UPS battery, providing hand-held portability

---

Libraries Used:

- adafruit-circuitpython-adxl34x
- adafruit-circuitpython-bitbangio
- adafruit-blinka
- board
- time
- math
- random
- pillow
- lib_oled96
- smbus2

Next Steps:

- Create custom eight-ball answers
- Possibly add a buzzer for audio feedback

Conclusion:
A successful Raspberry Pi-based Magic Eight Ball built from scratch. The project demonstrates GPIO multiplexing, software I2C with bit-banging, integration of OLED displays, and real-world sensor triggering in a fun, interactive application. Very neat!

Addendum: Power Management & Switching:

- Soft Switching
    - A push-button switch is connected to the last two GPIO pins:
        - GPIO21
        - GND
    - Purpose: Acts as a soft shutdown trigger for the Raspberry Pi Zero W.
    - Behavior: A Python script using the gpiozero library listens for button presses. When the button is pressed, the Pi safely shuts down, cutting software-level power.
    - Python package used: from gpiozero import Button
- Hard Switching
    - A rocker switch is wired to the PiSugar S battery's PCB.
    - Connected pads on the PCB:
        - OFF
        - CTRL
    - Function: Physically cuts power from the PiSugar battery after the soft shutdown process, ensuring no power draw remains; acts like an extended switch circuit.
    - Important note: The main ON switch on the PiSugar battery must be in the ON position for this extended switch circuit to function.

    - Reference: Configuration follows instructions provided in the PiSugar GitHub Wiki.

- The button and rocker switch are mounted directly through holes in the outer shell of the Magic Eight Ball, making the power controls easily accessible.

- Issues and Resolutions

    - During testing, I encountered an issue related to the I2C serial protocol. Both the Magic Eight Ball script and the shutdown script attempted to access the OLED display on the same I2C bus, which occasionally resulted in I/O errors due to bus contention.

    - To resolve this, I modified the shutdown script to terminate the Magic Eight Ball process before initiating the system shutdown. This ensured that only one process accessed the I2C device at a time, preventing further I/O errors.

Addendum: External Charging Port Modification

- To enable charging the PiSugar battery without opening the Magic Eight Ball enclosure, I added an external USB-C charging port.

- I used the following components:

    - Cermant 10pcs Type C Breakout Board Serial Basic Breakout Female Connector Type 16P PCB Converter Board

    - EAHOSUCC 0.3M/1FT 22AWG USB Type C Male Pigtail, 5V 3A Single USB-C to 2-Pin Bare Wire Open End Cable Cord (Red & Black 2 Wire)

- I created a hole in the outer shell and designed a small flat standoff, which I glued in place in front of the hole. The USB-C connector is then screwed into the standoff, providing a secure and flush-mounted charging port.

- The USB-C cable is plugged into the PiSugar's USB-C connector, and the bare wire ends of the pigtail are soldered to the Cermant breakout board to complete the connection. This setup allows the PiSugar to be conveniently charged from the outside without needing to open the enclosure.