Sweep & Strike

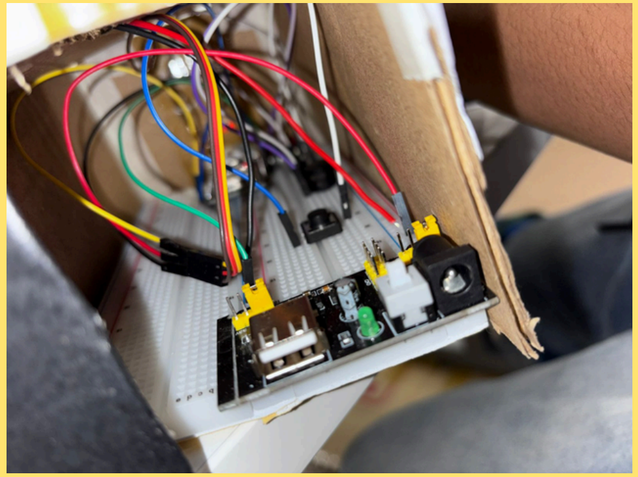
KRUSHNA JAIN

KHYATISPRIHA HAZARIKA

IDEA

“The idea behind our game is to simulate a radar-based detection system while testing human reflexes. The servo acts as a radar sweep, LEDs represent scanning and targets, and the player must react within a precise time window.

As the score increases, the system speeds up, increasing difficulty dynamically.”



It Started With a Real Radar Concept.....

How to Play - In 3 Simple Steps

1. Watch the Sweep

The servo arm moves like a radar, and a blue LED travels across the NeoPixel strip. One LED is randomly selected as the red target.

2. Wait for Yellow

When the moving blue light reaches the red target, it turns yellow — this is your hit window.

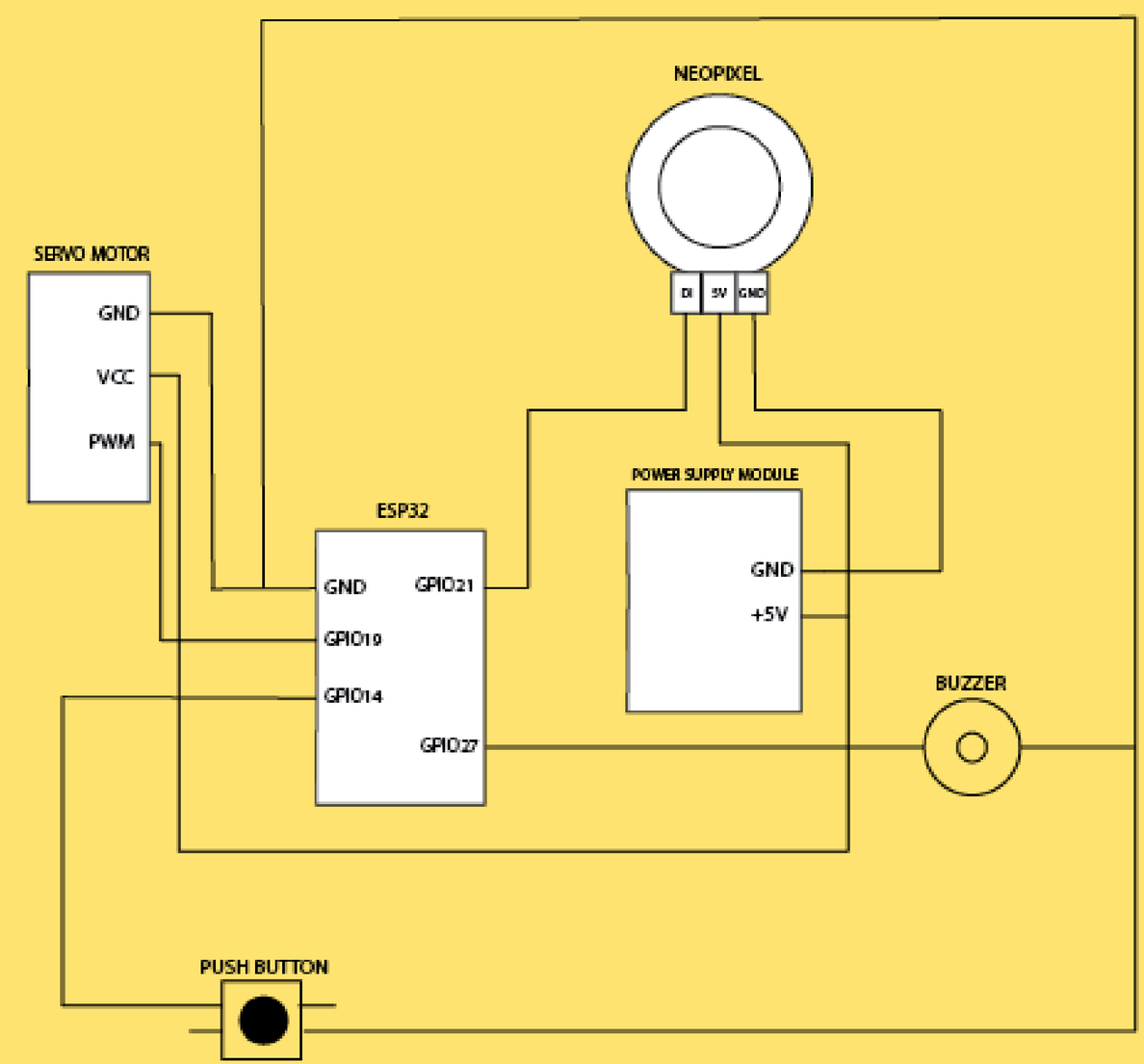
3. Press to Score

Press the button while it's yellow to score a HIT , if its a hit u hear buzzer sound.

Press at the wrong time or miss the yellow window - it counts as a MISS.

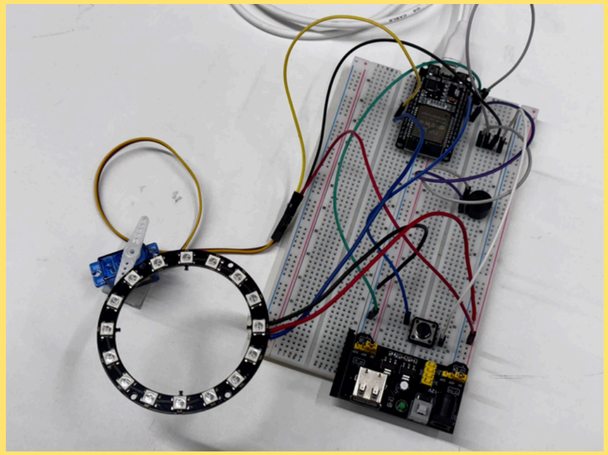
PLAY

CIRCUIT DIAGRAM



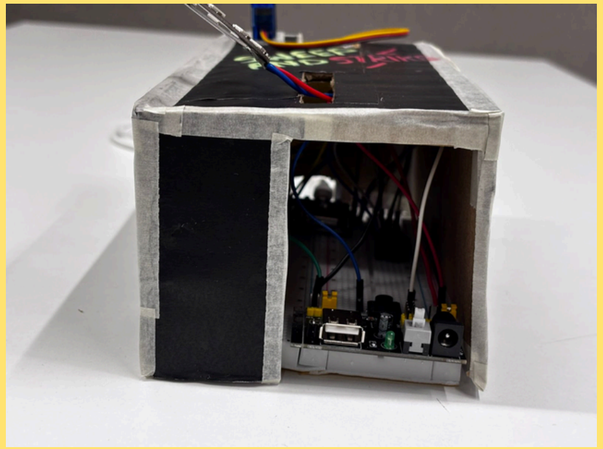
PHOTOS

1.



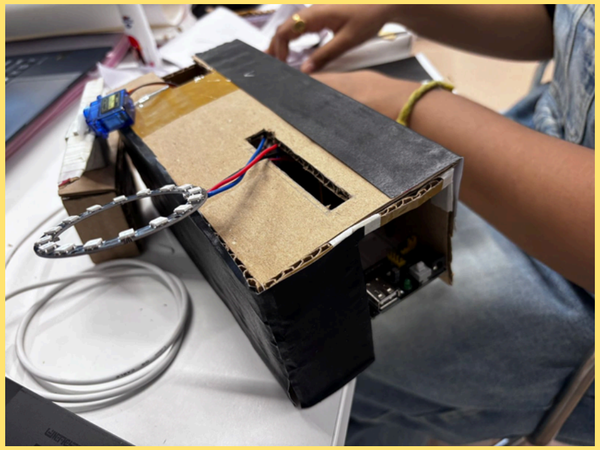
2.

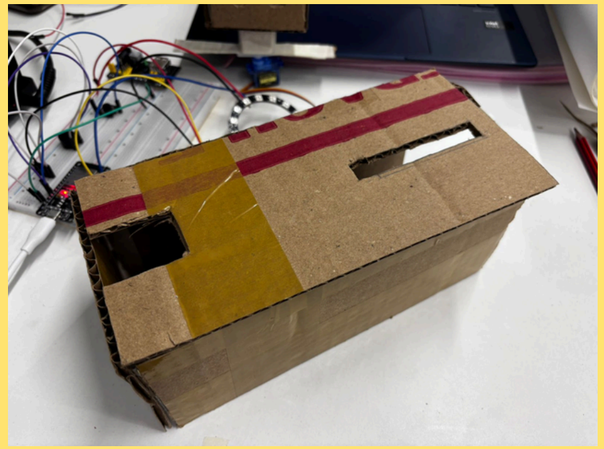
3.

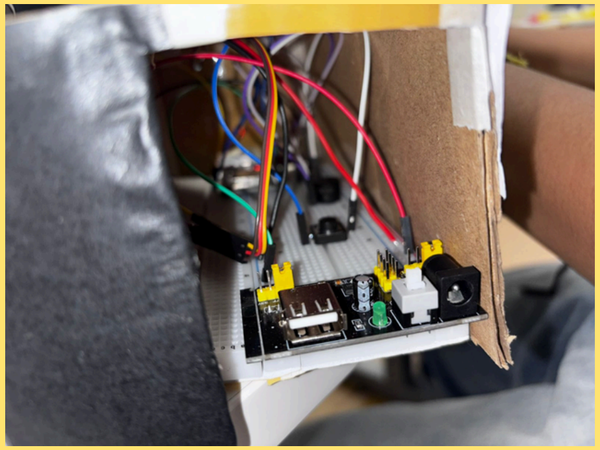


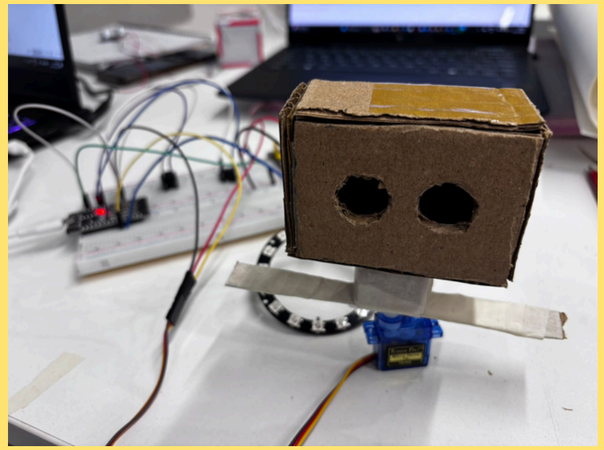
1. Circuit before adding the box.
2. Circuit after adding the box.
3. Inner view of the circuit in the box.

BEHIND THE SCENES







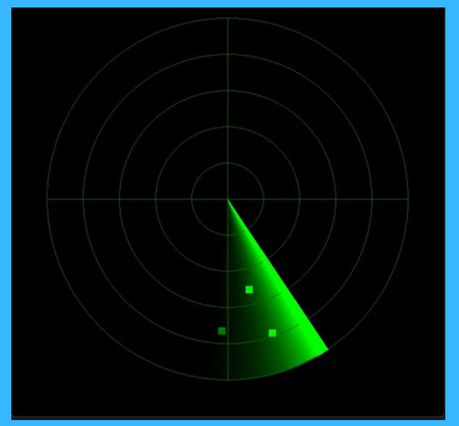


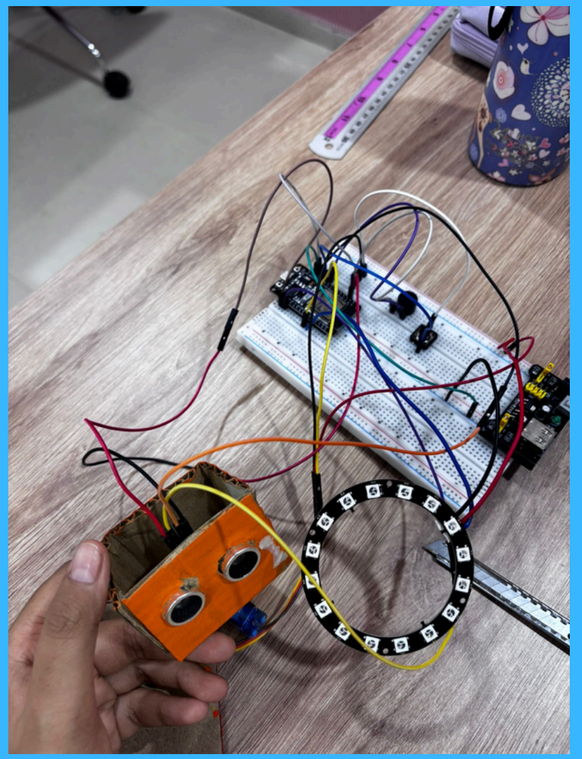
SWEEP
AND STRIKE

Original Technical Idea

- Servo motor rotates (like radar sweep)
- Ultrasonic sensor mounted on servo
- Sensor detects object distance
- Angle of servo mapped to NeoPixel LED
- If object detected at certain distance:
  - Corresponding LED lights up
  - System pauses
  - Player presses button to confirm detection
  - Loop starts again

It was meant to be a physical radar simulation.





The real challenge began when we integrated the ultrasonic sensor. At every servo step, the system had to send a pulse, wait for the echo, calculate distance, update LEDs, and check the button simultaneously. This overloaded the loop, making the sweep slow, inconsistent, and no longer smooth: turning what should have been fluid motion into a delayed process.

So we removed real-time ultrasonic detection.
And replaced it with:
Random target generation
Controlled timing window

Why These Specific Components?

🔹 **Servo Motor**

Mimics radar sweep

Teaches PWM control

Maps physical angle to logical index

🔹 **NeoPixel LEDs**

Individually addressable

Precise angle-to-light mapping

Clear visual feedback (blue, red, yellow)

🔹 **Push Button**

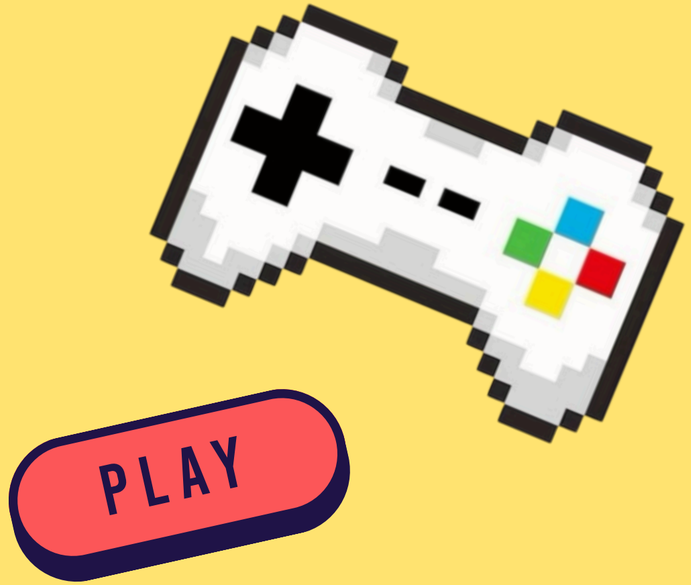
Simple user interaction

Reaction measurement

🔹 **Buzzer**

Immediate feedback loop

Improves user engagement



Coding Logic Used

1. Angle Mapping Logic
A list of servo duty values:

positions = [35,45,55,65,75,85,95,105,115]

Each index = one LED position

2. Forward & Backward Sweep

```
# Control sweep speed
time.sleep(delay)
```

```
# FORWARD SWEEP
# Move servo from left to right
for i in range(9):

    # Move servo to position corresponding to LED index
    servo.duty(positions[i])
```

for i in range(9):

for i in range(8,-1,-1):

3. Timing Window Logic

for k in range(12):
time.sleep(0.01)

This created a ~120ms reaction window.

```
# Set target LED to red
np[target] = (200,0,0)

# Set radar position to blue
np[i] = (0,0,200)

# Check if radar reached target
if i == target:

    # Yellow hit zone
    np[i] = (200,200,0)
    np.write()
```

```
# Increase game speed slightly
if delay > 0.05:    # Minimum delay limit
    delay -= 0.01
```

4. Delay to increases speed

5. When light reaches target turns yellow

CONTRIBUTION

KHYATI

```
from machine import Pin, PWM
import time
import neopixel
import random

# === Setup ===
servo = PWM(Pin(19), freq=50)
np = neopixel.NeoPixel(Pin(21), 9)
button = Pin(14, Pin.IN, Pin.PULL_UP)
buzzer = PWM(Pin(27))

positions = [35,45,55,65,75,85,95,105,115]

score = 0
miss = 0
delay = 0.11       # base sweep speed

press_count = 0
max_presses = 15

# ================ INITIALIZING ANIMATION (BLUE) ================
for loop in range(3):  # repeat 3 times
    # Forward loop
    for i in range(9):
        for j in range(9):
            np[j] = (0,0,0)
        np[i] = (0,0,200)  # blue
        np.write()
        time.sleep(0.08)
    # Backward loop
    for i in range(8,-1,-1):
        for j in range(9):
            np[j] = (0,0,0)
        np[i] = (0,0,200)
        np.write()
        time.sleep(0.08)

# Clear LEDs before game starts
for i in range(9):
    np[i] = (0,0,0)
np.write()
```

KRUSHNA

```
print("GAME START!")

# ================ GAME LOOP ================
while press_count < max_presses:

    target = random.randint(0,8)

    # -------- FORWARD SWEEP --------
    for i in range(9):

        if press_count >= max_presses:
            break

        servo.duty(positions[i])

        # Clear all LEDs
        for j in range(9):
            np[j] = (0,0,0)

        # Target red
        np[target] = (200,0,0)
        # Radar blue
        np[i] = (0,0,200)

        # Yellow hit zone
        if i == target:
            np[i] = (200,200,0)
        np.write()

        # Check button presses in hit window
        for k in range(9):    # ~90ms window
            if press_count >= max_presses:
                break

            if button.value() == 0:
                press_count += 1
```

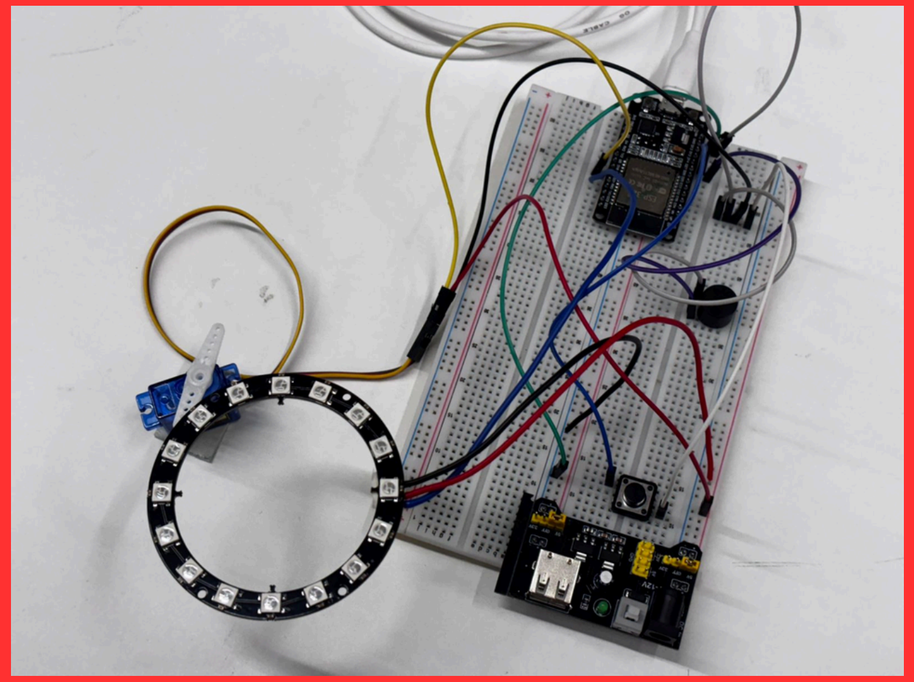
```
    # Check button presses in hit window
    for k in range(9):   # ~90ms window
        if press_count >= max_presses:
            break

        if button.value() == 0:
            press_count += 1

            if i == target:
                score += 1
                print("HIT! Score:", score)
            else:
                miss += 1
                print("MISS! Total misses:", miss)

            # Buzzer beep
            buzzer.freq(1000)
            buzzer.duty(512)
            time.sleep(0.06)
            buzzer.duty(0)

            if delay > 0.045:
                delay -= 0.007

            time.sleep(0.02)  # small debounce
            break

        time.sleep(0.01)

    time.sleep(delay)
```

KRUSHNA

```
# -------- BACKWARD SWEEP --------
for i in range(8,-1,-1):

    if press_count >= max_presses:
        break

    servo.duty(positions[i])

    # Clear all LEDs
    for j in range(9):
        np[j] = (0,0,0)

    # Target red
    np[target] = (200,0,0)
    # Radar blue
    np[i] = (0,0,200)

    # Yellow hit zone
    if i == target:
        np[i] = (200,200,0)
    np.write()

    # Check button presses in hit window
    for k in range(9):
        if press_count >= max_presses:
            break

        if button.value() == 0:
            press_count += 1

            if i == target:
                score += 1
                print("HIT! Score:", score)
            else:
                miss += 1
                print("MISS! Total misses:", miss)

            buzzer.freq(1000)
            buzzer.duty(512)
            time.sleep(0.06)
            buzzer.duty(0)

            if delay > 0.045:
                delay -= 0.007

            time.sleep(0.02)
            break

        time.sleep(0.01)

    time.sleep(delay)
```

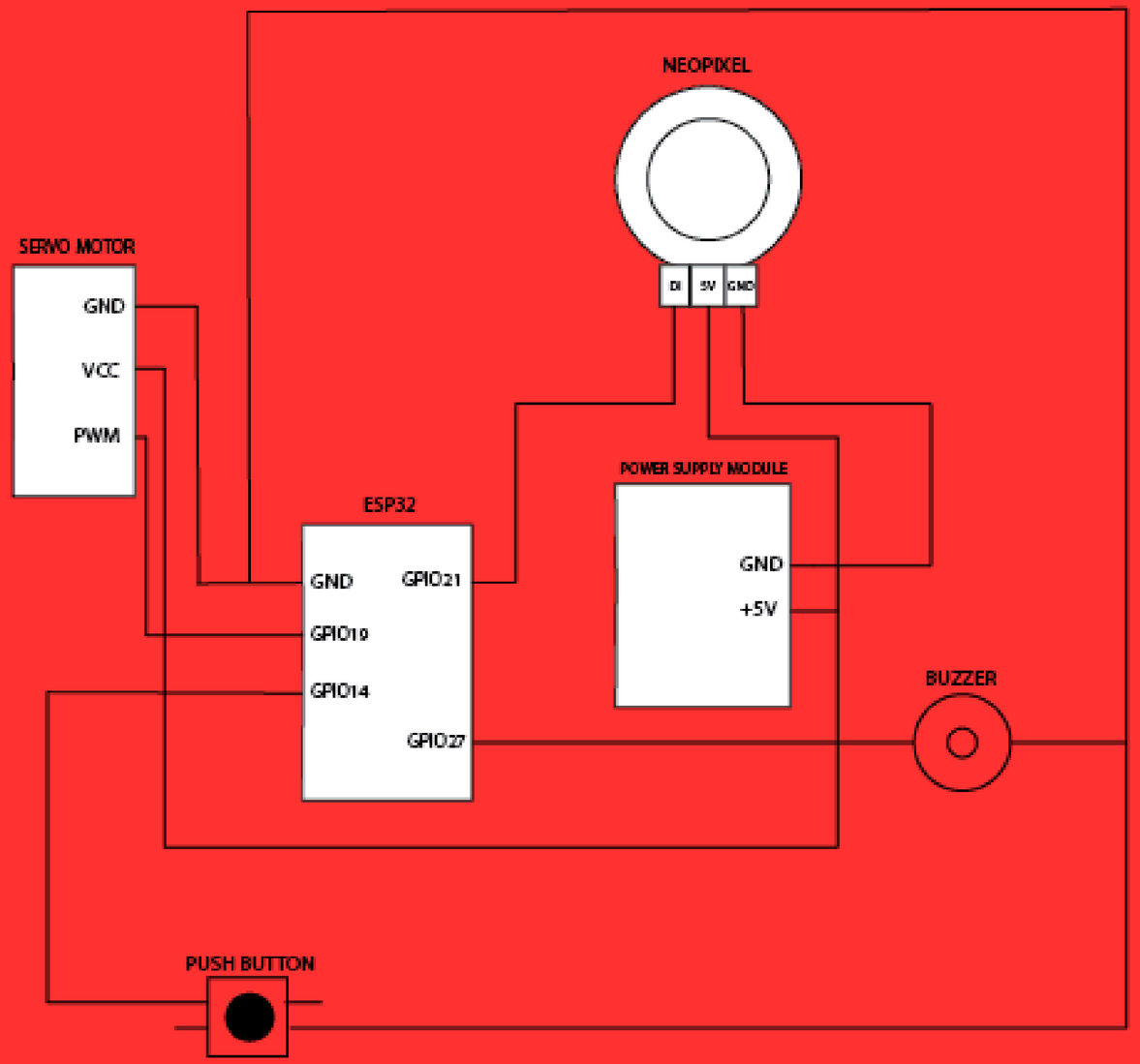
KHYATI

```
# ================ GAME OVER ================
print("GAME OVER")
print("Final Score (HITS):", score)
print("Total Misses:", miss)
print("Total Button Presses:", press_count)

# ================ END ANIMATION (RED) ================
for loop in range(3):  # repeat 3 times
    # Forward loop
    for i in range(9):
        for j in range(9):
            np[j] = (0,0,0)
        np[i] = (200,0,0)  # red
        np.write()
        time.sleep(0.08)
    # Backward loop
    for i in range(8,-1,-1):
        for j in range(9):
            np[j] = (0,0,0)
        np[i] = (200,0,0)
        np.write()
        time.sleep(0.08)

# Turn everything off
for i in range(9):
    np[i] = (0,0,0)
np.write()
servo.duty(0)
buzzer.duty(0)
```

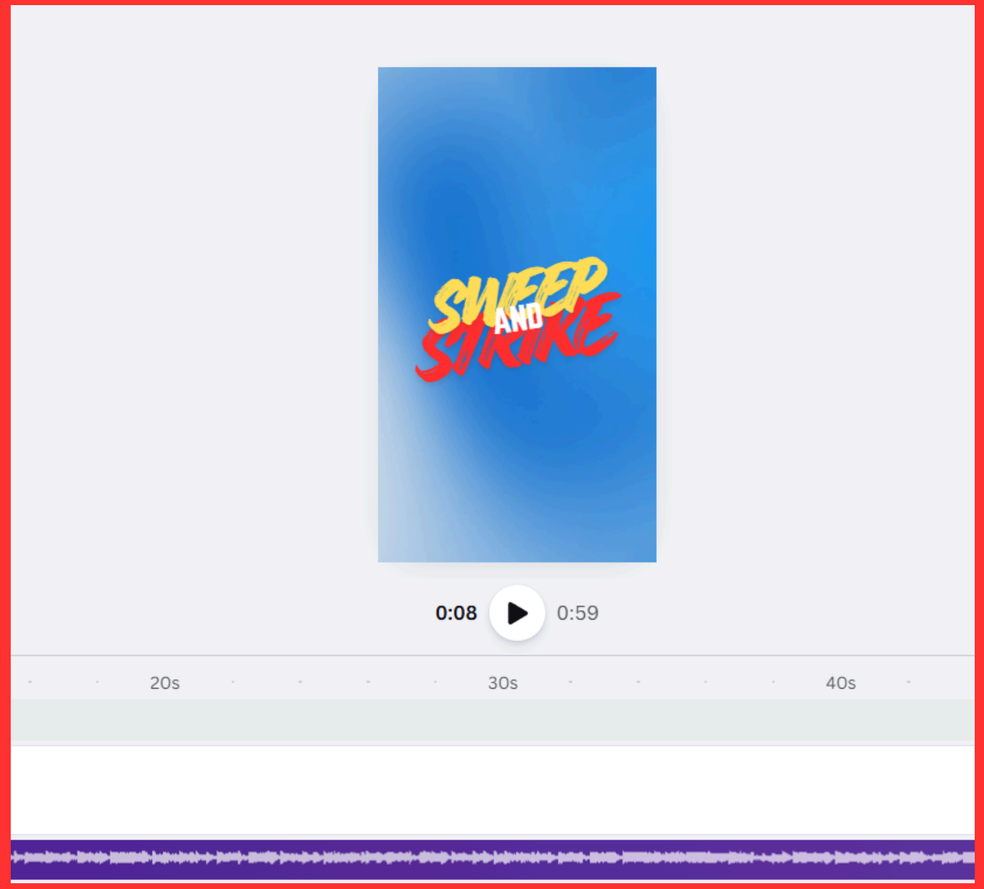


KRUSHNA & KHYATI

CIRCUIT DIAGRAM



VIDEO EDIT



KRUSHNA

AESTHETICS OF MODEL

KHYATI

thank you

KRUSHNA JAIN

KHYATISPRIHA HAZARIKA