Lichttheremin

Calliope:


```
let fr = 0
basic.forever(function () {
fr = Math.map(input.lightLevel(), 0, 255, 131, 2093)
music.ringTone(fr)
basic.pause(100)
})
```

Raspberry Pico

Bauteile Theremin

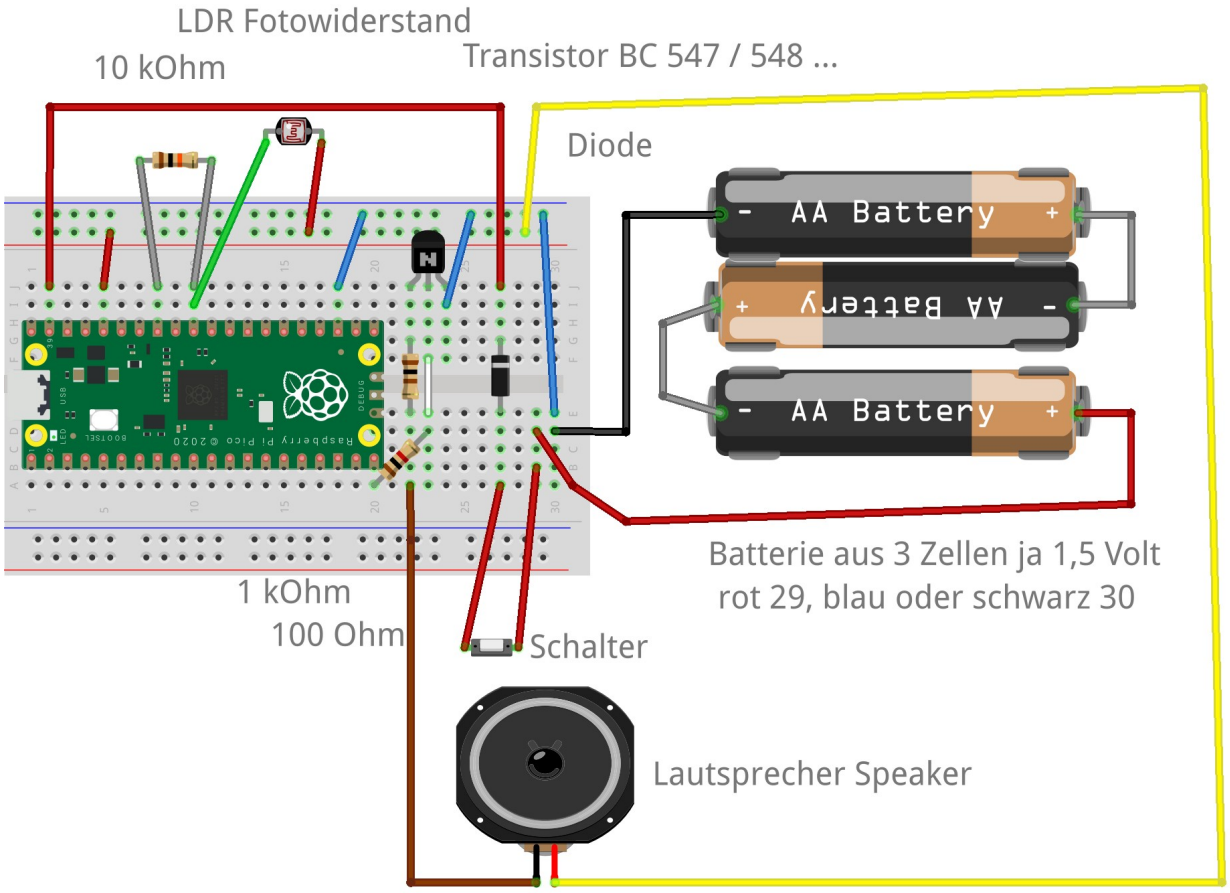
Amount	Part Type
3	Battery Holders
1	Schottky Diode
1	Raspberry Pi Pico
1	10kΩ Resistor
1	Photocell (LDR)
1	1kΩ Resistor
1	100Ω Resistor

1 Pushbutton

1 SPEAKER

1 TRANSISTOR_NPN_2

Aufbauplan



Pythonprogramm

```
"""
Licht-Theremin für Raspberry Pi Pico
Verwendet einen LDR (Lichtsensor) um klassische Theremin-Töne zu erzeugen

Hardware-Verbindungen:
- LDR: Ein Bein an 3.3V
- LDR: Anderes Bein an GPIO26 (ADC0) UND über 10kΩ Widerstand an GND
- Lautsprecher/Buzzer: Positiv an GPIO15, Negativ an GND
  (Für passiven Buzzer direkt, für Lautsprecher ggf. Verstärker nötig)
"""

from machine import Pin, ADC, PWM
import time
import math
```

```
# Konfiguration
LDR_PIN = 26        # ADC Pin für LDR
SPEAKER_PIN = 15    # PWM Pin für Lautsprecher
LED_PIN = 25        # Onboard LED

# Frequenzbereich (typisch für Theremin)
MIN_FREQ = 100      # Tiefste Frequenz in Hz
MAX_FREQ = 3000     # Höchste Frequenz in Hz

# Toleranzbereich (10% am hellen Ende = kein Ton)
SILENCE_THRESHOLD = 0.10  # 10% Toleranz wo kein Ton gespielt wird

# LDR-Kalibrierung (werden automatisch angepasst)
ldr_min = 65535
ldr_max = 0

# Hardware initialisieren
ldr = ADC(Pin(LDR_PIN))
speaker = PWM(Pin(SPEAKER_PIN))
led = Pin(LED_PIN, Pin.OUT)

def read_ldr():
    """Liest den LDR-Wert und gibt normalisierten Wert zurück (0.0 - 1.0)"""
    global ldr_min, ldr_max

    # Mehrfach-Messung für Stabilität
    value = 0
    samples = 5
    for _ in range(samples):
        value += ldr.read_u16()
        time.sleep_ms(1)
    value = value // samples

    # Auto-Kalibrierung
    if value < ldr_min:
        ldr_min = value
    if value > ldr_max:
        ldr_max = value

    # Normalisierung
    if ldr_max > ldr_min:
        normalized = (value - ldr_min) / (ldr_max - ldr_min)
    else:
        normalized = 0.5

    return normalized

def calculate_theremin_frequency(light_value):
    """
    Berechnet Theremin-Frequenz basierend auf Lichtwert
    Verwendet logarithmische Skalierung für musikalischeren Klang
    """
```

```
    # Logarithmische Interpolation (klingt musikalischer)
    log_min = math.log(MIN_FREQ)
    log_max = math.log(MAX_FREQ)

    log_freq = log_min + (log_max - log_min) * light_value
    frequency = math.exp(log_freq)

    return int(frequency)

def apply_vibrato(base_freq, time_ms, depth=0.02, rate=5):
    """
    Fügt Vibrato hinzu für authentischeren Theremin-Sound
    depth: Vibrato-Tiefe (0.02 = 2% Frequenzvariation)
    rate: Vibrato-Geschwindigkeit in Hz
    """
    vibrato = math.sin(2 * math.pi * rate * time_ms / 1000)
    freq_variation = base_freq * depth * vibrato
    return int(base_freq + freq_variation)

def play_theremin_tone(frequency, light_value):
    """Spielt einen Theremin-Ton mit der gegebenen Frequenz"""
    # 10% Toleranz am hellen Ende - kein Ton wenn > 90%
    if light_value > (1.0 - SILENCE_THRESHOLD):
        speaker.duty_u16(0)
        return

    if frequency < 20:  # Zu tief für menschliches Ohr
        speaker.duty_u16(0)
        return

    speaker.freq(frequency)
    # 50% Duty Cycle für klareren Ton
    speaker.duty_u16(32768)

def calibrate_ldr():
    """Kalibriert den LDR durch Messung der Extremwerte - erst hell, dann dunkel"""
    print("Kalibrierung startet...")
    print("\n>>> Phase 1: HELL-Messung <<<")
    print("Bitte LDR HELL beleuchten...")
    print("Messung läuft...\n")

    # Hell-Messung OHNE blinkende LED
    light_values = []
    for i in range(30):  # 3 Sekunden Messung
        light_values.append(ldr.read_u16())
        time.sleep_ms(100)

    light_value = sum(light_values) // len(light_values)
    print(f"✓ Hell-Wert gemessen: {light_value}")

    time.sleep(1)
```

```
    print("\n>>> Phase 2: DUNKEL-Messung <<<")
    print("Jetzt LDR komplett ABDECKEN (dunkel machen)...")
    print("LED blinkt während der Messung...\n")

    # Dunkel-Messung MIT blinkender LED
    dark_values = []
    for i in range(30):  # 3 Sekunden Messung
        led.toggle()
        dark_values.append(ldr.read_u16())
        time.sleep_ms(100)

    led.off()
    dark_value = sum(dark_values) // len(dark_values)
    print(f"✓ Dunkel-Wert gemessen: {dark_value}")

    global ldr_min, ldr_max
    ldr_min = min(dark_value, light_value)
    ldr_max = max(dark_value, light_value)

    print(f"\n✓ Kalibrierung abgeschlossen!")
    print(f"  Bereich: {ldr_min} - {ldr_max}")
    print(f"  Stummzone: {int((1-SILENCE_THRESHOLD)*100)}-100% (hell)")
    print("\nLicht-Theremin startet!\n")

def main():
    """Hauptprogramm"""
    print("=" * 40)
    print("  LICHT-THEREMIN für Raspberry Pi Pico")
    print("=" * 40)

    # Kalibrierung
    calibrate_ldr()

    start_time = time.ticks_ms()
    vibrato_enabled = True

    print("Bewege deine Hand über dem LDR um zu spielen!")
    print("Drücke Ctrl+C zum Beenden\n")

    try:
        while True:
            # Lichtwert lesen
            light_value = read_ldr()

            # Basis-Frequenz berechnen
            base_freq = calculate_theremin_frequency(light_value)

            # Vibrato hinzufügen für authentischen Theremin-Sound
            if vibrato_enabled:
                current_time = time.ticks_diff(time.ticks_ms(), start_time)
                frequency = apply_vibrato(base_freq, current_time)
            else:
```

```
            frequency = base_freq

        # Ton abspielen (mit Stummzone-Prüfung)
        play_theremin_tone(frequency, light_value)

        # Status ausgeben (nicht zu oft für bessere Performance)
        if time.ticks_ms() % 100 < 10:
            light_percent = int(light_value * 100)

            # Status-Anzeige mit Stummzonen-Indikator (stumm wenn > 90% = hell)
            if light_value > (1.0 - SILENCE_THRESHOLD):
                status = "STUMM"
                freq_display = "----"
            else:
                status = "SPIELT"
                freq_display = f"{frequency:4d}"

            print(f"Licht: {light_percent:3d}% | {status} | Freq: {freq_display} Hz | "
                  f"[{'' * (light_percent // 5)}{' ' * (20 - light_percent // 5)}]",
                  end='\r')

        # Kleine Verzögerung für Stabilität
        time.sleep_ms(5)

    except KeyboardInterrupt:
        print("\n\nLicht-Theremin gestoppt.")
        speaker.duty_u16(0)  # Ton ausschalten
        speaker.deinit()
        led.off()  # LED ausschalten
        print("Auf Wiedersehen!")

# Programm starten
if __name__ == "__main__":
    main()
```