Ch06 計步器 – 震動開關

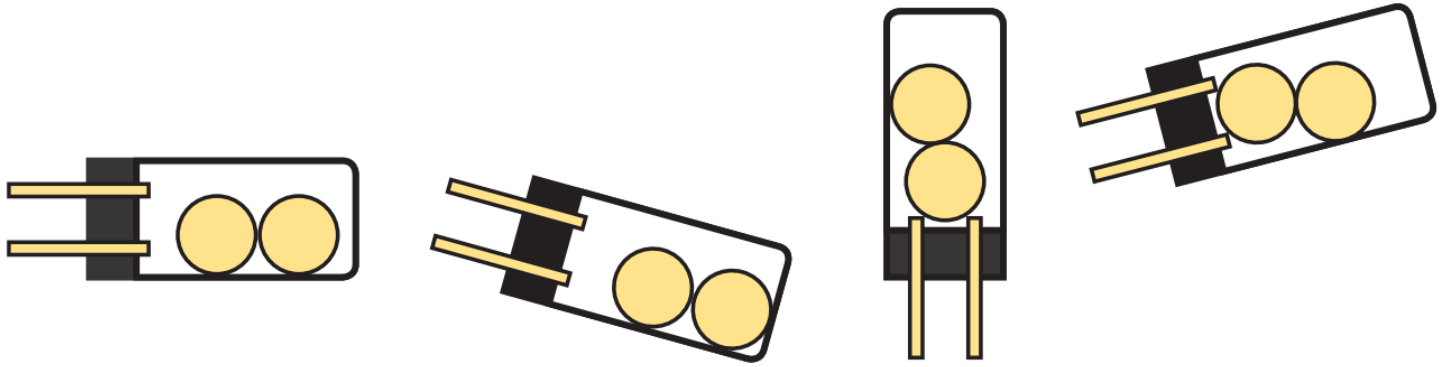
6-1 認識震動感測開關

- **震動開關**

SW-520D 是滾珠震動開關，在其金色圓筒內有 2 枚金屬滾珠，會隨著開關的傾斜角度移動：

6-1 認識震動感測開關

圓筒朝正上方或傾斜角度很小時，電路上形成短路。
圓筒傾斜或受到震動、搖晃時，針腳之間為斷路。

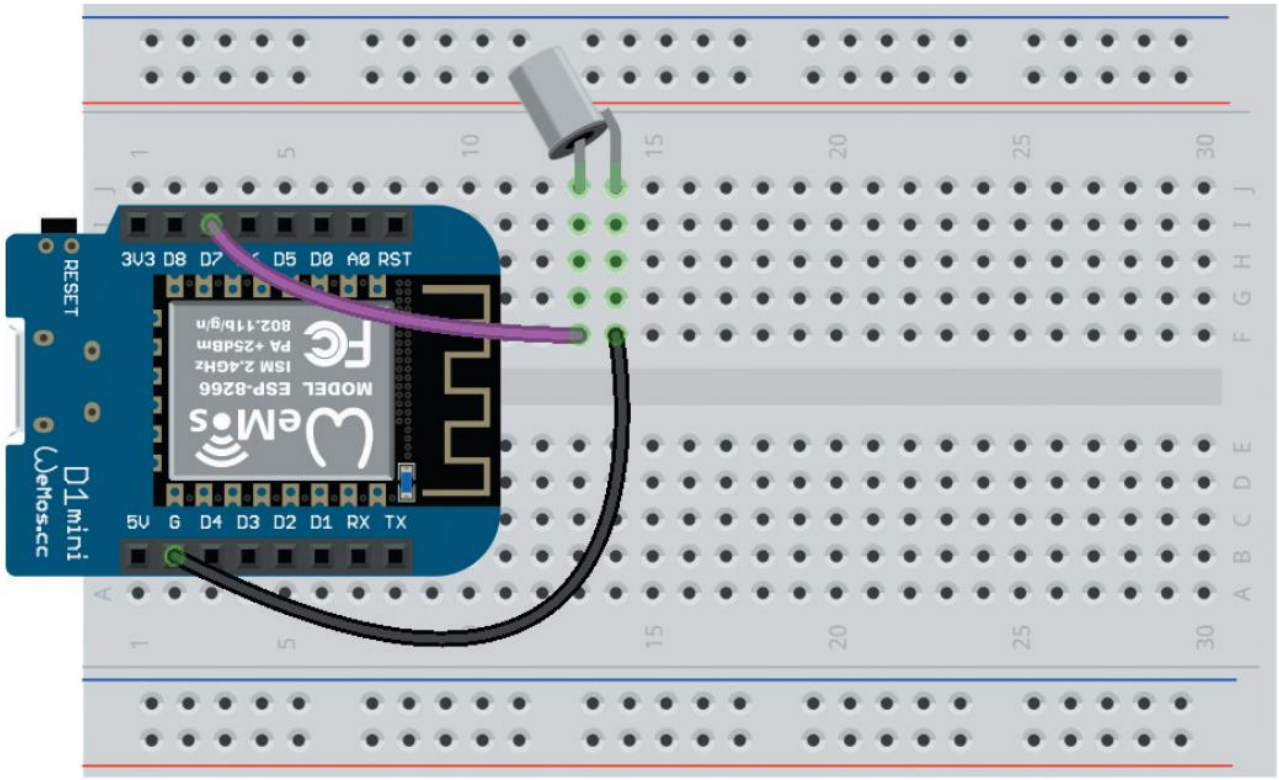


傾斜 – 關　　大幅傾斜 – 關　　無傾斜 – 開　　微微傾斜 – 開

6-1 認識震動感測開關

- **接線**

把震動開關兩腳分別接上控制板的 D7 腳位與 G :



ex6-1

6-1 認識震動感測開關

透過控制板腳位來讀取電路的狀態：

```
from machine import Pin
import utime

sw520d = Pin(13, Pin.IN)   # 設為讀取模式

while True:
    print(sw520d.value()) # 讀取狀態
    utime.sleep_ms(100)
```

6-1 認識震動感測開關

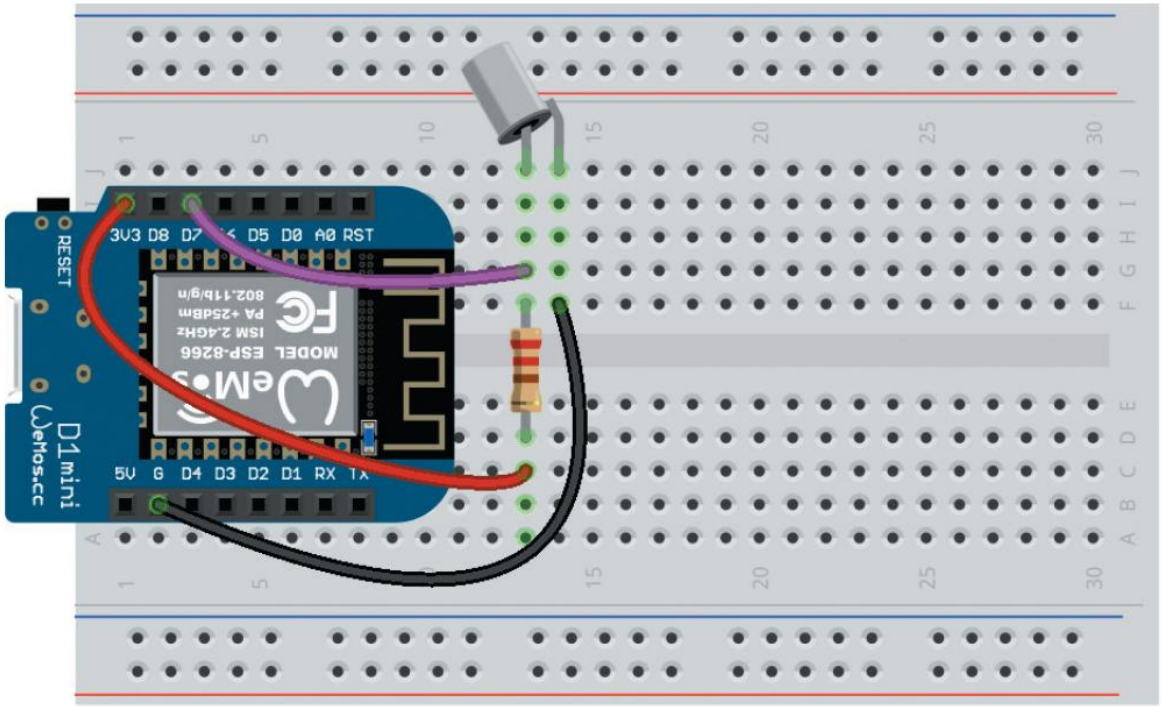
執行程式後，由於震動開關兩端正插在麵包板上，
D7 腳位會讀到接地線的低電位 (0) 。

但這時再把震動開關倒過來形成斷路、
D7腳位變沒有接任何電路的狀態，沒有讀取到訊號，
隨環境雜訊影響讀到不準確且浮動的數值，讀數依然是 0。
為解決上述問題，得使用上拉電阻。

6-1 認識震動感測開關

- **設定上拉電阻**

在電路中加入 3.3V 電源線，接一顆電阻後再接上開關的一腳，開關另一腳同樣連到接地線。



6-1 認識震動感測開關

電路的運作會如下 :

- ✓ 開關不通時，訊號線會讀到電源線的電壓 (高電位)。
- ✓ 開關接通時，電源線的電力流入接地線，
  訊號線讀到接地線電壓 (低電位)。

如此一來，程式就能讀到 0 或 1 兩種狀態。

6-1 認識震動感測開關

D1 mini 這類控制板都有內建上拉電阻，可用程式決定是否要啟用的上拉電阻：

```
sw520d = Pin(13, Pin.IN, Pin.PULL_UP)
```

Pin.PULL_UP 就是讓這個腳位啟用內建上拉電阻。

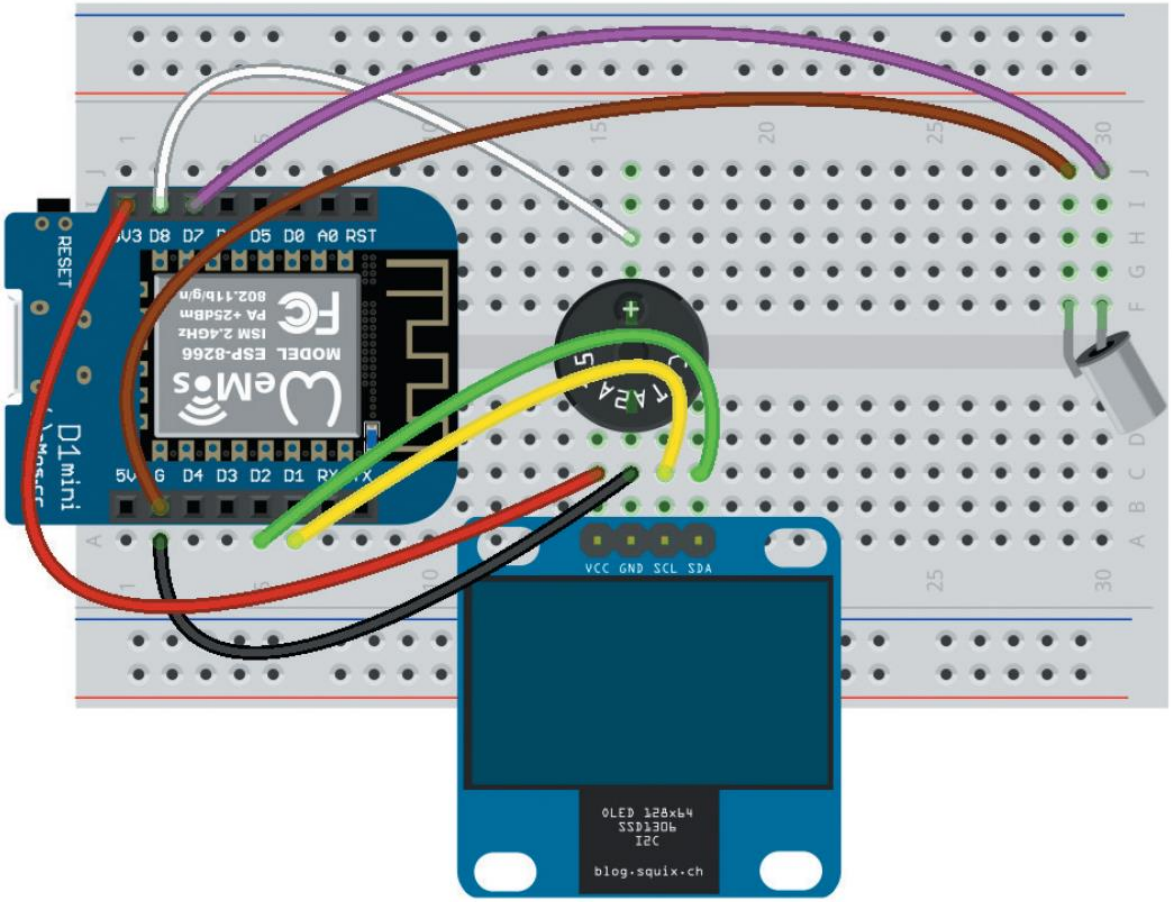
6-2 傾斜偵測

- **Lab17**

跌倒偵測器	
實驗目的	在偵測到震動開關傾斜時發出跌倒警報
材料	• D1 mini • OLED 模組 • 無源蜂鳴器 • 震動開關

6-2 傾斜偵測

- 接線圖


RESET
D1 mini
MODEL ESP-8266
ISM 2.4GHz
PA +25dBm
802.11b/g/n
VCC GND SCL SDA
0.96" 128x64
SSD1306
I2C
blog.squix.ch

6-2 傾斜偵測

- **設計原理**

替震動開關建立名為 sw520d 的腳位讀取物件，
只要 sw520d.value() 傳回 1 代表開關傾倒、形成斷路，
反之則是正常狀態。

- **程式設計**

```
from machine import Pin, I2C, PWM
from ssd1306 import SSD1306_I2C
import utime

sw520d = Pin(13, Pin.IN, Pin.PULL_UP)
oled = SSD1306_I2C(128, 64, I2C(scl=Pin(5), sda=Pin(4)))
buzzer = PWM(Pin(15, Pin.OUT), freq=880, duty=0)
```

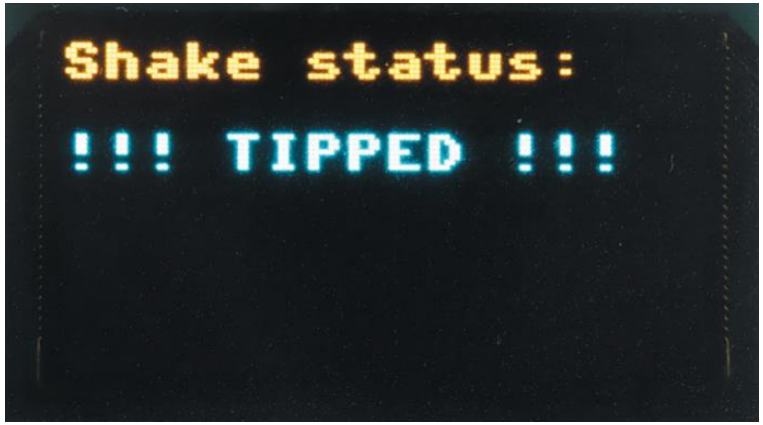
6-2 傾斜偵測

```
while True:

    oled.fill(0)
    oled.text("Shake status:", 0, 0)

    # 判斷震動開關狀態
    if sw520d.value() == 1:
        buzzer.duty(512)
        oled.text("!!! TIPPED !!!", 0, 16)
        print("!!! 感測器傾倒 !!!")

    else:
        buzzer.duty(0)
        oled.text("- Standby -", 0, 16)
        print("感測器狀態正常")

    oled.show()
    utime.sleep_ms(100)
```

6-2 傾斜偵測

- **實測**

執行程式，刻意傾斜震動開關，OLED 模組及編輯器互動環境窗格便都會出現警告訊息，蜂鳴器也會響起：




```
感測器狀態正常
感測器狀態正常
!!!  感測器傾倒  !!!
!!!  感測器傾倒  !!!
!!!  感測器傾倒  !!!
```

6-3 震動偵測

- **Lab18**

健康計步器	
實驗目的	偵測震動開關的搖晃次數來計算步數
材料	• D1 mini • OLED 模組 • 無源蜂鳴器 • 震動開關

- **接線圖**

同 Lab 17 。

6-3 震動偵測

- **設計原理**

當震動開關晃動時，讀數會先變成 1，停止晃動時回到 0 。
為計算正確，程式會判斷變化間隔是否超過 50 毫秒。

- **程式設計**

```
from machine import Pin, I2C, PWM
from ssd1306 import SSD1306_I2C
import utime

sw520d = Pin(13, Pin.IN, Pin.PULL_UP)
oled = SSD1306_I2C(128, 64, I2C(scl=Pin(5), sda=Pin(4)))
buzzer = PWM(Pin(15, Pin.OUT), freq=392, duty=0)

oled.text("Step counter", 0, 8)
oled.show()
print("計步器已啟動")
```

ex6-3

6-3 震動偵測

```
count = 0

while True:

    if sw520d.value() == 1: # 震動開關晃動

        # 記錄震動開關受到晃動的時間起點
        start_time = utime.ticks_ms()

        # 等待震動開關停止晃動 (傳回 0)
        while sw520d.value() == 1:
            pass # 什麼事也不做, 但用 pass 這句來維持迴圈結構
```

ex6-3

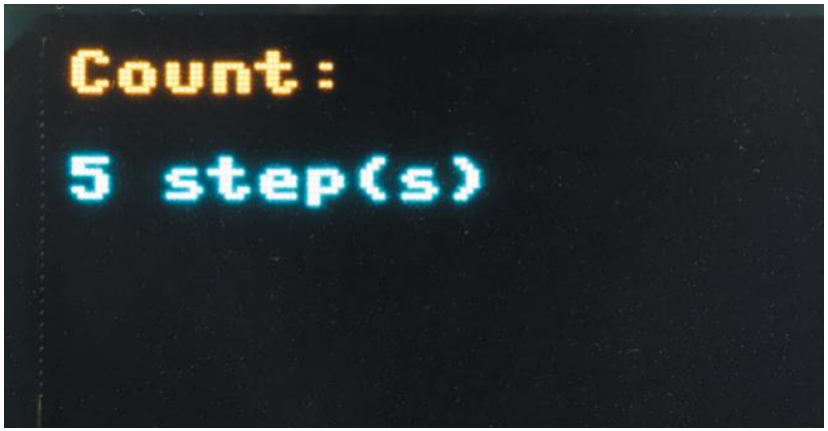
6-3 震動偵測

```
# 如果搖晃導致狀態改變的時間間隔超過 50 毫秒,
# 代表是正確人為搖晃, 步數加 1:
if utime.ticks_ms() - start_time > 50:

    count += 1
    print("步數: " + str(count))

    buzzer.duty(512)

    oled.fill(0)
    oled.text("Count:", 0, 0)
    oled.text(str(count) + " step(s)", 0, 16)
    oled.show()

    # OLED 本身讀寫有時間延遲, 故不再加入額外延遲
    buzzer.duty(0)
```

6-3 震動偵測

- **實測**

執行程式後，搖晃震動開關，每搖一次蜂鳴器就響一下，OLED 模組與編輯器互動窗格也會顯示步數：




```
計步器已啟動
步數: 1
步數: 2
步數: 3
步數: 4
步數: 5
```

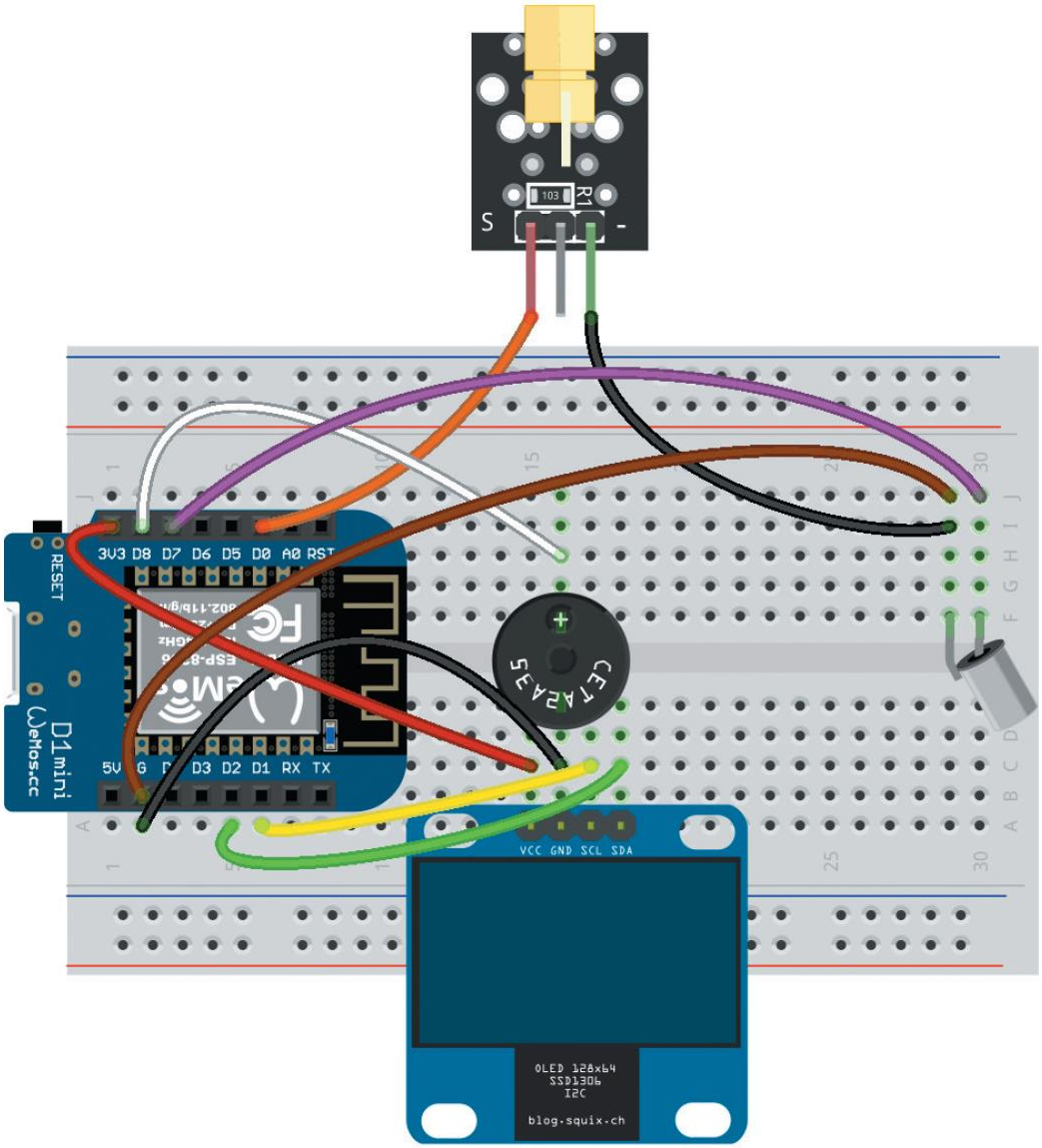
6-4 用感測器設計遊戲

- **Lab19**

反應考驗遊戲機	
實驗目的	在看到雷射光一點亮時就搖晃震動開關
材料	• D1 mini • OLED 模組 • 無源蜂鳴器 • 震動開關 • 雷射模組

6-4 用感測器設計遊戲

- 接線圖



6-4 用感測器設計遊戲

- **設計原理**

用控制板一個腳位點亮雷射模組，等震動開關傳回 1，計算從雷射點亮到開關晃動的時間間隔。

6-4 用感測器設計遊戲

- 程式設計

```
from machine import Pin, I2C, PWM
from ssd1306 import SSD1306_I2C
import utime, urandom

sw520d = Pin(13, Pin.IN, Pin.PULL_UP)
oled = SSD1306_I2C(128, 64, I2C(scl=Pin(5), sda=Pin(4)))
buzzer = PWM(Pin(15, Pin.OUT), freq=440, duty=0)
laser = Pin(16, Pin.OUT)

laser.off()

oled.fill(0)
oled.text("Reaction test", 0, 0)
oled.text("SHAKE to start", 0, 16)
oled.show()
print("反應大考驗 - 搖晃震動開關來啟動測試")

while True:

    if sw520d.value() == 1: # 啟動測試

        laser.off()
```

ex6-4

```
    # 倒數 3 秒, 用 reversed 讓 I 從 2 減到 0
    for i in reversed(range(3)):
        oled.fill(0)
        oled.text("Reaction test", 0, 0)
        oled.text("Ready in " + str(i + 1) + " secs", 0, 16)
        oled.show()
        print("倒數 " + str(i + 1) + " 秒...")
        buzzer.freq(440)
        buzzer.duty(512)
        utime.sleep(0.1)
        buzzer.duty(0)
        utime.sleep(0.9)

    buzzer.freq(880)
    buzzer.duty(512)
    oled.fill(0)
    oled.text("Started!", 0, 0)
    oled.text("SHAKE when laser", 0, 16)
    oled.text("turns on:", 0, 32)
    oled.show()
    print("測試開始! 在雷射光一點亮時搖晃震動開關")
    utime.sleep(0.3)
    buzzer.duty(0)
    utime.sleep(0.7)
```

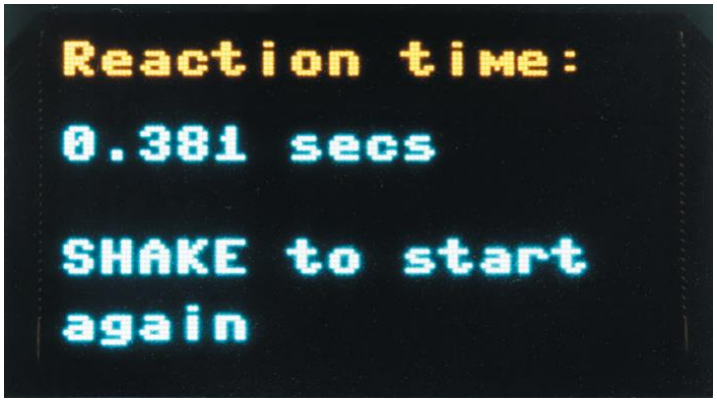
ex6-4

```
# 隨機等待 3 至 10 秒
utime.sleep(3 + urandom.getrandbits(3))

laser.on() # 打開雷射光, 開始計時
start_time = utime.ticks_ms()

while sw520d.value() == 0:
    pass

# 測到搖晃, 計算反應時間
reaction_time = (utime.ticks_ms() - start_time) / 1000

# 報告結果
oled.fill(0)
oled.text("Reaction time:", 0, 0)
oled.text(str(reaction_time) + " secs", 0, 16)
oled.text("SHAKE to start", 0, 40)
oled.text("again", 0, 54)
oled.show()
print("反應時間: " + str(reaction_time) + " 秒")
print("測試結束 - 搖晃震動開關以再次測試")
buzzer.freq(440)
buzzer.duty(512)
utime.sleep(0.1)
buzzer.freq(880)
utime.sleep(0.3)
buzzer.duty(0)
utime.sleep(1.6) # 結束時停頓片刻, 確定開關狀態保持靜止
```

6-4 用感測器設計遊戲

- **實測**

將雷射模組指著某平面，執行程式。
搖晃一次震動開關來啟動反應測試，
等倒數完畢。



測試開始後，
一看見雷射點亮就搖晃震動開關。

OLED 模組與編輯器互動環境窗格便會顯示反應時間：

```
反應大考驗 - 搖晃震動開關來啟動測試
倒數 3 秒...
倒數 2 秒...
倒數 1 秒...
測試開始! 在雷射光一點亮時搖晃震動開關
反應時間: 0.381 秒
測試結束 - 搖晃震動開關以再次測試
```