使用 RFID 製作問答遊戲

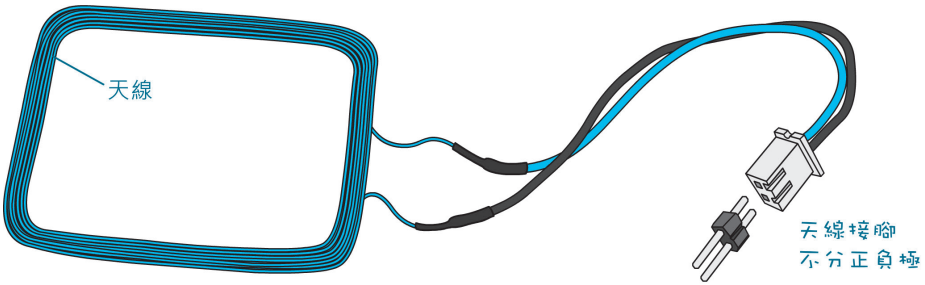
本附錄為本書第二版第 18 章，與本書第 16 章內容呼應，說明 125kHz 的 RFID 模組，方便讀者參考。

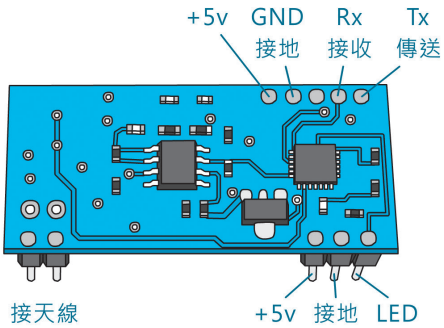
H-1 RFID 模組介紹與標籤讀取實驗

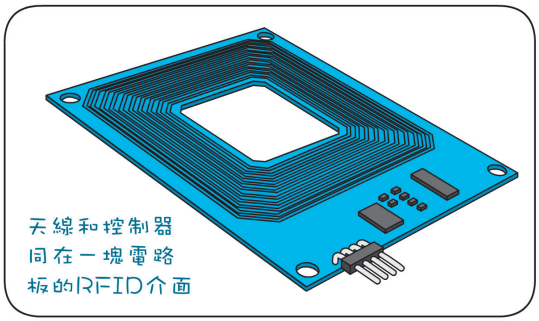
選購 RFID 模組時，需要留意底下幾項規格：

- **輸入電壓**：有些採用 5V，有些則是 3.3V，建議選用 5V。
- **標籤頻率**：通常是 125KHz 或 13.56MHz。採用任何頻率都行，但採購標籤時，記得要買相同頻率的款式。
- **資料介面**：讀卡機模組和微電腦連結的介面，有序列埠（TTL 電位格式，也稱作 UART 介面）、I^2C、SPI、USB 和藍牙無線等類型，本文選用序列埠類型。

底下是筆者購買的 RFID 模組外觀，標籤頻率為 125KHz、外接線圈、使用 5V 供電，消耗電流小於 50mA，採**鮑率 9600bps** 的 **TTL 序列埠**介面。有些讀卡機模組的天線直接刻蝕在印刷電路板上。







各家的 RFID 讀卡機模組的接腳定義不盡相同，請詳閱說明書。上圖的腳位依照廠商提供的說明書繪製，實際安裝時，右下角的電源和接地不用接。

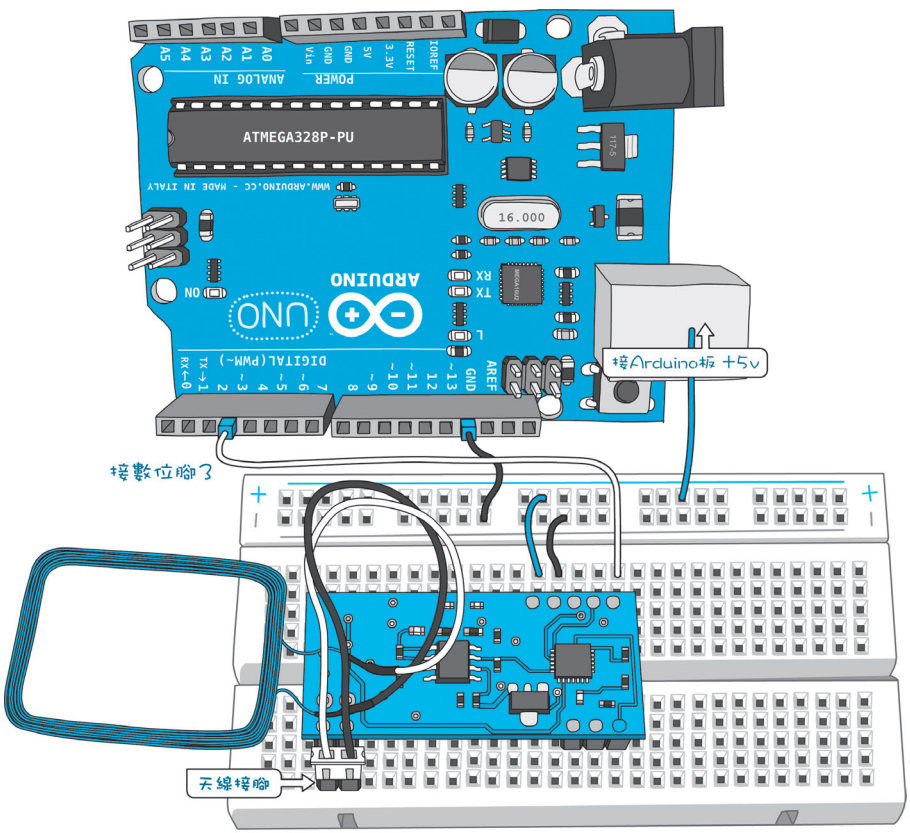
動手做 H-1 讀取 RFID 標籤

實驗說明：連接 RFID 模組與 Arduino，讀取兩個不同的 RFID 標籤，在**序列埠監控視窗**顯示標籤資料（內碼）。

實驗材料：

RFID 讀卡機（頻率規格需要和 RFID 標籤搭配）	1 個
RFID 標籤	2 個

實驗電路：讀卡機只需連結**電源**、**接地**和**序列傳送（TxD）接腳**（讀卡機僅傳出標籤資料，不寫入資料，因此不用連接序列輸入腳）：



實驗程式：為了避免影響 Arduino 預設序列埠的運作，本單元的程式同樣採用「軟體序列埠」程式庫來接收讀卡機的資料，範例程式如下（原始檔名：diyH_1_1.ino）：

```
#include <SoftwareSerial.h>
SoftwareSerial RFID(3, 4);      // 接收腳=3, 傳送腳=4

byte data;                      // 暫存標籤資料的變數

void setup() {
  Serial.begin(9600);
  RFID.begin(9600);
  Serial.println("RFID Ready!");
}

void loop() {
  if (RFID.available() > 0) { // 若讀取到序列資料...
    data = RFID.read();       // 儲存讀取到的資料
    Serial.println(data);     // 將資料顯示在序列埠監視窗
  }
}
```

實驗結果：編譯並執行以上程式後，開啟**序列埠監控視窗**，視窗首先會顯示 "RFID Ready!" 訊息。當你感應 RFID 標籤時，序列埠將持續顯示 RFID 的標籤碼。筆者購買的 125KHz 標籤，總是以數字 2 開頭，3 結尾，共 14 個數字：

```
2
48
54
48
48
55
53
54
52
57
53
56
50
3
```

請掃描你手邊的 RFID 標籤（至少兩個），並記下這些標籤的編碼備用。

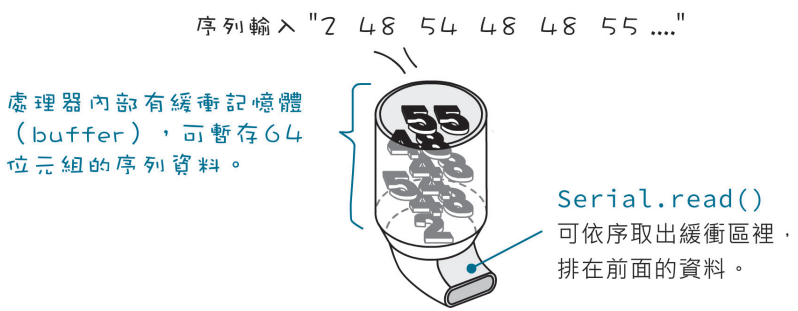
H-2 儲存與比對 RFID 編碼

典型的 RFID 應用，例如門禁卡，都是事**先在微電腦中儲存特定 RFID 卡片的編碼值**。當持卡人掃描門禁卡時，系統將讀取並且比對儲存值，如果有相符，就開門讓持卡人通過。

筆者購買的 RFID 標籤每次都會傳回 14 個數字，為了比對資料，每次掃描到的卡片編碼，都要先暫存在記憶體中。我們可以用一個執行 14 次讀取的迴圈，把讀入的數字分別存入陣列：

```
byte temp[14];  // 宣告將用來儲存 RFID 標籤編碼的陣列
byte i = 0;     // 陣列元素的索引，從 0 開始
...
// 如果有資料，而且元素索引值小於 14...
while (RFID.available() && i < 14){
    temp[i] = Serial.read();       // 讀取資料並存入陣列
    i++;
}
i = 0;  // 陣列索引值歸 0，以便再次記錄下一組 14 個數字
...
```

底下是另一種寫法。由於**傳入序列埠的資料會暫存在微處理器內部的緩衝區（buffer）**，因此，程式可以等到所有數字都傳入微處理器之後，再一起讀取。



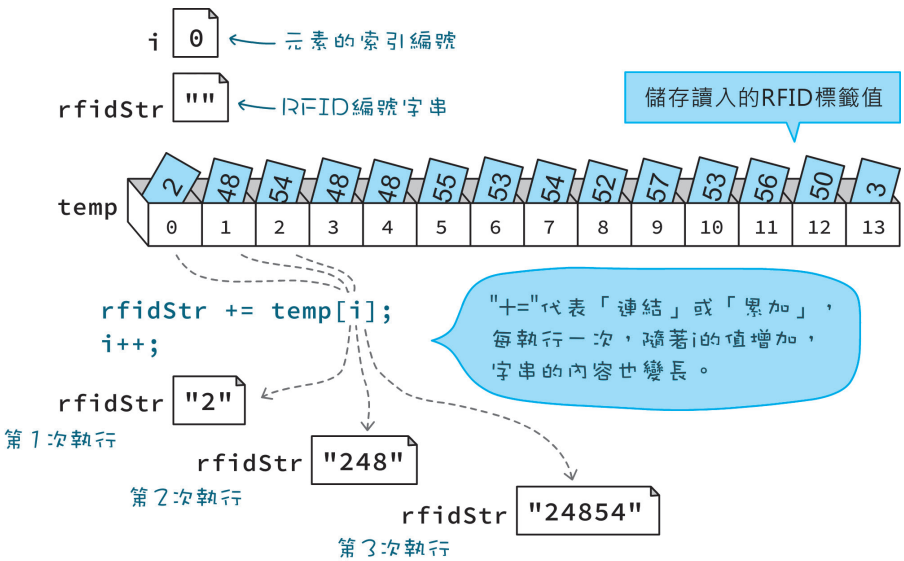
筆者的 RFID 讀卡機模組採用 9600bps 連線，因此每一秒鐘約可傳遞 1200 個字元 (9600÷8=1200)，所以當程式收到第一個字元後，等待 0.1 秒讓讀卡機傳入 14 個數字，已綽綽有餘。

筆者將讀取序列埠程式寫成**自訂函數 readTag()，此函數將傳回 0 或 1，代表是否有讀取到資料**（原始檔名：diyH_1_2.ino）：

```
#include <SoftwareSerial.h>
SoftwareSerial RFID(3, 4); // 接收腳=3, 傳送腳=4

const byte TAG_LEN = 14;    // 定義標籤資料的長度
byte temp[TAG_LEN];         // 存放讀入標籤的 14 個數字
String rfidStr = "";


// 負責讀入 RFID 編碼值的自訂函數，傳回值類型為「布林」
boolean readTag() {
  // 代表是否讀入資料的變數，預設值為 0，代表沒有
  boolean ok = 0;
  if (RFID.available()) { // 如果讀卡機傳入新的資料...
    delay(100);           // 等 0.1 秒，讓其餘數字都傳進來

    // 執行 14 次迴圈，讀取緩衝區裡的數字
    for (byte i=0; i<TAG_LEN; i++) {
      temp[i] = RFID.read();
    }
    RFID.flush();  // 清除緩衝區
    ok = 1;        // 讀取完畢後，設定成 1，代表有讀到新資料
  }
  return ok;       // 傳回 0 或 1
}

void setup() {
  Serial.begin(9600);
  RFID.begin(9600);
  Serial.println("RFID Ready!");
}

void loop() {
  // 呼叫自訂函數，如果讀取到標籤，此判斷條件式將「成立」
  if (readTag()) {
```

```
    rfidStr = "";
    // 把讀入的 14 個數字連結成字串
    for (byte i=0; i<TAG_LEN; i++) {
        rfidStr += temp[i];
    }
    Serial.println(rfidStr);  // 顯示標籤的編碼字串
  }
}
```

loop() 函數中，把讀入的 14 個數字連結成字串的迴圈敘述，說明如下：



編譯並上傳程式碼，再開啟**序列埠監控視窗**用 RFID 標籤測試，將能看到 Arduino 每次都會輸出類似底下的一長串 RFID 碼：

```
248544848555354525753565O3
```

動手做 H-2 使用 RFID 控制開關

實驗說明：根據上一節的說明，採用兩個 RFID 標籤，一個當做「開」，另一個當做「關」，來控制 Arduino 第 13 腳的 LED。本單元的實驗材料與電路，都和「動手做 H-1」相同。

實驗程式：請先在程式開頭加入兩個 RFID 標籤值，以及 LED 腳位的定義：

```
const byte ledPin = 13;     // LED 位於 13 腳
// 請將底下的數字改成你的 RFID 標籤數字
// 代表「開啟」的 RFID 編碼值
String tagON= "248504848535056495770527 13" ;
// 代表「關閉」的 RFID 編碼值
String tagOFF = "248544848555345257535 6503" ;
```

接著，將 LED 腳位設定為「輸出」：

```
void setup() {
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
  RFID.begin(9600);
  Serial.println("RFID Ready!");
}
```

最後，在主程式迴圈中，加入比對標籤值的敘述（原始檔名：diyH_2_1.ino）：

```
void loop() {
  if (readTag()) {
    rfidStr = "";
    for (byte i=0; i<TAG_LEN; i++) {
       rfidStr += temp[i];
    }
    Serial.println(rfidStr);

    if (rfidStr == tagON) { // 如果 RFID 碼等於「開啟」的編碼值
      digitalWrite(ledPin, HIGH); // 點亮 LED
    } else if (rfidStr == tagOFF) {
      // 否則，若 RFID 碼等於「關閉」編碼
      digitalWrite(ledPin, LOW);  // 關閉 LED
    }
  }
}
```

將 RFID 標籤資料存入「程式記憶體」區

上文的程式碼將 RFID 標籤資料儲存成字串 (String) 類型，這樣的程式寫法比較簡單，但由於這些資料將會佔用有限的**資料記憶體**空間，如果 RFID 標籤有很多組，建議把它們存放在**程式記憶體**區：

將陣列存入程式記憶體

```
#include <avr/pgmspace.h>
PROGMEM byte tags[2][14] = {
  {2, 48, 50, 48, 48, 53, 50, 56, 49, 57, 70, 52, 71, 3},
  {2, 48, 54, 48, 48, 55, 53, 54, 52, 57, 53, 56, 50, 3}
};
```

宣告儲存兩組，各14個元素的陣列。

請將這些元素值，換成你自己的RFID標籤數字。

上面的 tags 陣列儲存了兩組標籤 (每一組各有 14 個元素)，假設我們要比對這兩組的標籤值是否相同，可以使用**比較兩個記憶體區塊 (陣列) 值的 memcmp() 函數**，語法如下：

此處正確的說法是「指向某記憶體區塊的變數」

memcmp (陣列1, 陣列2, 要比較的位元組數) ➡ 如果兩個陣列內容相同，則傳回0

代表memory comparison (記憶體比較)

以上指令用於比較位於主記憶體 (SRAM，或者說「資料記憶體」) 的資料，若要比較位於程式記憶體區域 (program memory) 的資料，請使用這個指令：

memcmp_P (陣列1, 陣列2, 要比較的位元組數) ➡ 如果兩個陣列內容相同，則傳回0

代表比較存在"Program"程式記憶體區的資料

陣列元素的編號從 0 開始，比對兩組標籤的範例程式片段如下，若兩組相同，就在**序列埠監控視窗**顯示 "YES!"：

比對存在程式記憶體裡的標籤資料

比較第0和第1組的14個位元組值

```
if (memcmp_P(tags[0], tags[1], 14) == 0) {
  Serial.println("YES!");
} else {
  Serial.println("NO!");
}
```

底下是另一個使用 memcmp() 函數比較陣列值的範例片段 (完整程式請參閱 diyH_2_2.ino 檔)：

```
byte tagA[3] = {1, 2, 3}; ←儲存了3個位元組元素的陣列
byte tagB[3] = {4, 5, 6};

if (memcmp(tagA, tagB, 3) == 0) {
  Serial.println("YES!");  ←比較兩個陣列裡的3個位元組
} else {
  Serial.println("NO!"); ←因兩者不相等，所以這一行將被執行。
}
```

我們可以把剛剛動手做 H-2 程式中的標籤值，改存放在「程式記憶體」區，程式碼首先要修改存放標籤碼的定義：

```
#include <avr/pgmspace.h>
int tag = -1;

PROGMEM byte tags[TAG_TOTAL][TAG_LEN] = {
  // 第 0 組標籤
  {2, 48, 50, 48, 48, 53, 50, 56, 49, 57, 70, 52, 71, 3},
  // 第 1 組標籤
  {2, 48, 54, 48, 48, 55, 53, 54, 52, 57, 53, 56, 50, 3}
};
```

上面程式裡的 tag 變數，用來存放 RFID 標籤編號，**-1 代表找不到相同的標籤碼**。主程式迴圈採用 memcmp_P() 函數比對資料：

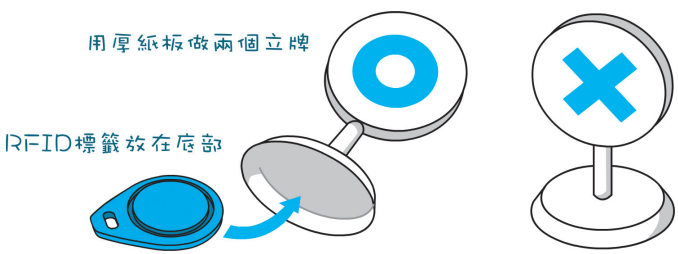
```
void loop() {
  if (readTag()) {          ←共有兩組標籤，此值為2。
    for (byte i=0; i<TAG_TOTAL; i++) {
                                  ←存放掃描到的標籤值
      if (memcmp_P(temp, tags[i], TAG_LEN) == 0 ) {
                                  每一組標籤有14個號碼
        Serial.println(i);
        tag = i;    //記錄標籤編號
        break;      //若找到相符的標籤，就跳出迴圈，不用再找了。
      } else {
        tag = -1;   //tag值為-1，代表沒有找到相同的標籤編碼。
      }
    }

    switch (tag) {
      case 0:       //如果是編號0的標籤，就點亮LED。
        digitalWrite(ledPin, HIGH);
        break;
      case 1:       //若是編號1的標籤，關閉LED。
        digitalWrite(ledPin, LOW);
        break;
    }
  }
}
```

迴圈程式裡的 **break 指令**，代表**中止迴圈**，也就是不再執行迴圈程式，直接跳到迴圈區塊以外，執行底下的程式。完整的程式碼請參閱書本範例檔 diyH_2_3.ino 檔。

動手做 H-3 使用 RFID 進行 Flash 問答遊戲

實驗說明：本章最後的範例為「是非問答題」，讀者可以根據此程式架構，自行擴充成選擇題、配對題或者其他形式的多媒體展示內容。

用戶依據題目，按下 O（對）或 ×（錯）鈕。若是要透過 Arduino 操作，則需要事先準備標示 O 或 × 的 RFID 標籤（標籤的編碼值，也需要預先設定好，請參閱下一節說明）。



自製包含 RFID 標籤的答題牌

採用 RFID 充當 Flash 問答題程式的輸入介面，本單元的實驗材料與電路，都和「動手做 H-1」相同。

實驗程式：Arduino 程式僅負責偵測並傳遞 RFID 資料，不需要預先儲存 RFID 標籤值。從 Arduino 傳遞給 Flash AS 程式的資料，必須是**以 Null 字元結尾的字串**，而 serproxy 序列埠通信軟體採用 57600 速率連線，因此，Arduino 程式的寫法如下（原始檔名：diyH_3.ino）：

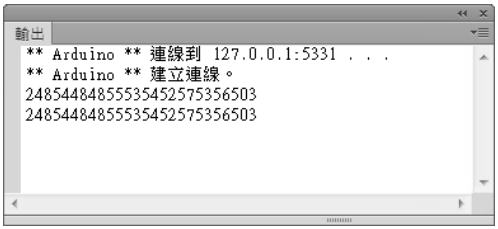
```
void setup() {
  Serial.begin(57600); // 使用 57600 鮑率和 serproxy 連線
  RFID.begin(9600);
}

void loop() {
  if (readTag()) {
    rfidStr = "";
    for (byte i=0; i<TAG_LEN; i++) {
        rfidStr += temp[i];      // 將收到的數字碼組成字串
    }
    Serial.print(rfidStr);       // 輸出 RFID 字串
    Serial.print( '\0');         // 緊接著輸出 Null 字元
  }
}
```

請先編譯並上傳此程式到 Arduino 板。

Flash AS 3.0 測試程式：底下是接收從 SerProxy 軟體傳來的 RFID 編碼的 ActionScript 3.0 程式，位於 readRFID.fla 範例檔的第一個關鍵影格。

```
import org.p43d.arduino.Arduino;
import org.p43d.arduino.ArduinoEvent;

// 請輸入 serProxy 設定的埠號
var a:Arduino = new Arduino(5331);
a.addEventListener(ArduinoEvent.ON_RECEIVE_DATA,
  receiveData);
// 每當 Arduino 有新資料傳入時，底下的事件函數將被自動執行
function receiveData(e:ArduinoEvent):void {
  var msg = e.data; // 讀取 Arduino 傳入的資料
  trace(msg);       // 在 Flash 的「輸出」面板呈現資料
}
a. connect();
```

請先把 Arduino 接上 USB，再執行 serproxy，然後回到 Flash 軟體並按下 Ctrl + Enter 鍵。當 Arduino 感應到 RFID 標籤時，Flash 將在**輸出**面板顯示類似右圖的 RFID 標籤碼：



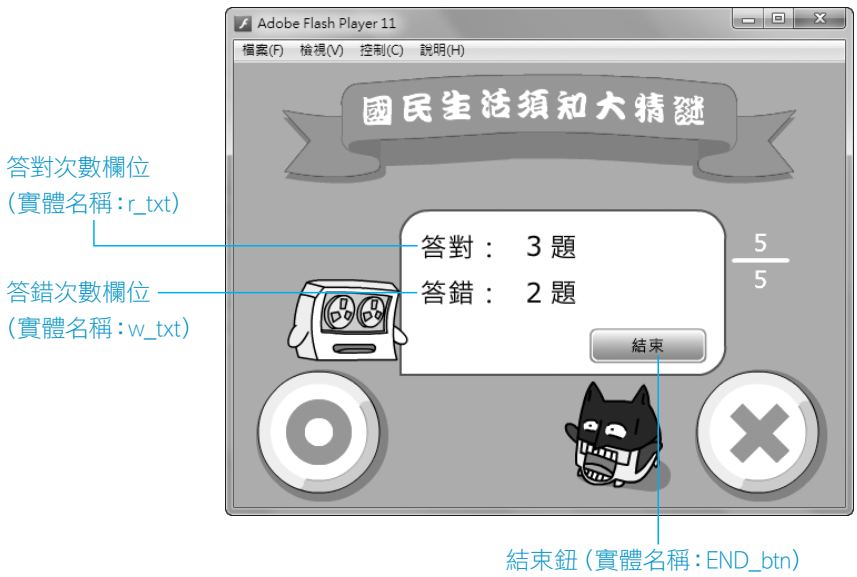
Flash 問答題的 AS 3.0 程式：本問答遊戲分成兩個場景，一開始首先顯示「操作方式選項」的場景畫面，使用者可以選擇「滑鼠」或 "Arduino+RFID" 操作（註：選擇 Arduino，還是能用滑鼠操作）。

選擇操作模式之後，畫面將切換到遊戲主畫面（場景 "main"）：

用戶將依照題目內容，按下 O（對）或×（錯）鈕作答。若是要透過 Arduino 操作，則需要事先準備標示 O 或×的 RFID 標籤（標籤的編碼值，也需要預先設定好，請參閱下一節說明）。

滑鼠及 Arduino+RFID 操作有一點不同：為了避免 Arduino 持續感應到 RFID 標籤，導致用戶尚未看清楚題目，系統就以為已經作答，並顯示如下的回應畫面，在 RFID 操作模式下，**當問題或回應畫面出現時，系統會延遲 1.5 秒（1500ms），才接收 RFID 標籤值**（此延遲時間可調整）。

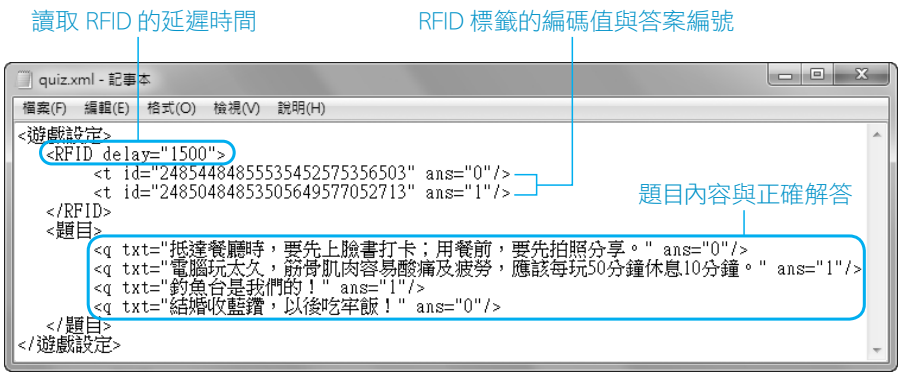
回答最後一題之後，**回應**畫面將切換到顯示答對與答錯的題數。若按下**結束**鈕，畫面將回到第一個場景。



遊戲過程中，出題角色 (drQ) 和玩家角色 (bian) 各自會切換不同的狀態畫面。

設定 RFID 標籤資料與「是非題」題目：本程式的 RFID 標籤編碼以及問題內容，都寫在一個稱為 "quiz.xml" 的 XML 檔案中。**XML 是一種用「標籤」定義資料內容的純文字檔**，它與 HTML 最大的不同是，所有標籤指令都能由我們自行定義，不像 HTML 有既定的標籤 (如：<html>, <body>, <p>...)。

底下是 quiz.xml 的內容，筆者定義了一個 RFID 標籤值的 "RFID"，與包含問題內容的「題目」標籤，每個問題都寫在自訂的 <q> 標籤中：

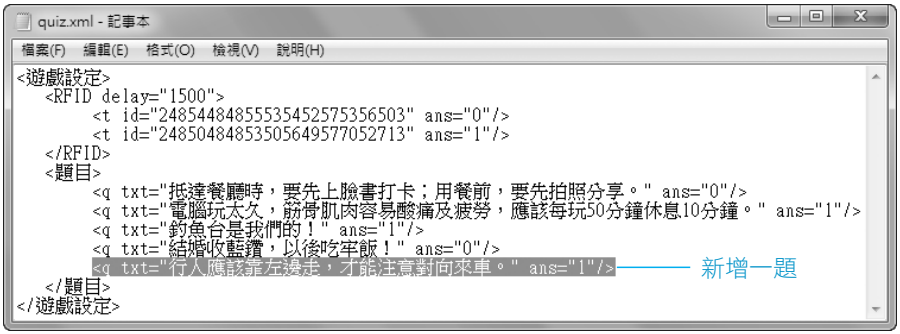


請使用「記事本」或其他文字編輯軟體開啟它，將 RFID 標籤的編碼改成你自己的 RFID 值，<t> 標籤裡的 ans 屬性值代表答案編號。

問題寫在 <q> 標籤裡的 txt 屬性，ans 屬性用於標示答案，0 代表「錯」、1 代表「對」。按照 XML 語法的規定，屬性值都要用雙引號或單引號刮起來。

XML 語法還規定，**每個標籤指令都要成對**，像 "<題目>" 與 "</題目>"，**若不成對，標籤指令必須用 "/>" 結束**，像本例中的 <t> 和 <q> 標籤。

若要增加新的題目，只要比照其他 <q> 標籤的格式，寫在 <題目> 元素之中即可，像這樣：



把遊戲的主要參數寫在 XML 檔的好處是，你不需要重新編譯、發佈程式，只要重新啟動遊戲，題目和其他參數就自動更新。

是非題遊戲的程式碼：遊戲影片檔（"問答遊戲.fla" 檔）的 **main 場景**的第一個關鍵影格程式，包含初始化程式的敘述，主要是載入遊戲 XML 設定檔：

```
// 設定題目的 XML 檔名與路徑，以及遊戲模式
init("quiz.xml", gameMode);
```

init() 函數的程式碼，實際寫在 Main.as 檔。此問答遊戲的程式碼由下列 .as 檔組成：

- tw\com\swf 路徑裡的 **Main.as** 類別程式，是問答遊戲的主程式。
- tw\com\swf**model** 路徑裡的 **Data.as** 類別程式，負責載入題目 XML 檔，並抽取出其中的 RFID 標籤碼與題目內容。
- tw\com\swf**model** 路徑裡的 **RFID.as** 類別程式，負責與 serproxy 序列程式連線，並且比對用戶的 RFID 標籤值。
- tw\com\swf**ui** 路徑裡的 **Response.as** 類別程式，是主畫面**回應**面板的主程式，負責記錄用戶答對和答錯的次數。當用戶按下其中的**下一題**或**結束**鈕時，它將發出 CLICK_BUTTON 自訂事件。
- tw\com\swf**events** 路徑裡的 **RFIDEvent.as** 自訂事件類別程式，用於傳遞從 Arduino 傳來的 RFID 標籤所代表的答案編號。

除了上文提到的介面，Flash 問答遊戲的其他主畫面元素，及其實體名稱如下：

當題目 (quizDialog) 和回應 (resp) 面板展開時，它們內部的時間軸將發出事件（事件名稱分別是 QUIZ_READY 和 POP），告知 Main.as 主程式，開始啟動 1.5 秒的延遲計時，再接收 RFID 標籤值。

若雙按舞台上的「回應」畫面，將能看到如下的時間軸架構：

其中的 actions 圖層的第 6 格關鍵影格，包含底下發佈 POP 自訂訊息的敘述：

```
dispatchEvent(new Event(POP));
stop();
```

其餘絕大部分的程式碼，都放在外部.as 檔，請讀者自行參閱裡面的註解說明。