音樂播放和混音

一、 聲音是什麼?

實作過很多章節之後，本章開始來談些不一樣的主題，聲音就是一個，很多時後，美妙的聲音能感動人心，能讓 EduCake 發出動人的聲音想必是件有趣的事情。

首先，得先了解，聲音是一種需要透過介質(空氣、水等等)傳遞的波動，在不同介質裡面還會有不同的速度，傳入耳朵，震動耳膜，然後我們才有辦法感知聲音的來源和內容。一般來說，人類耳朵可以聽到聲音的頻率約是在 20Hz~ 20000Hz(赫茲) 之間，超過這個範圍就稱為超音波，而低於這一範圍的稱為次聲波；而狗就可以聽到 40～50000Hz 的聲音，所以有時候明明沒聽到聲音狗還是大叫就是這樣，因為他們聽到人類聽不到的聲音了，用超音波模組簡單的測試就可以發現這個狀況。其中 Hz 是一種描述聲音頻率的單位，如下圖 1

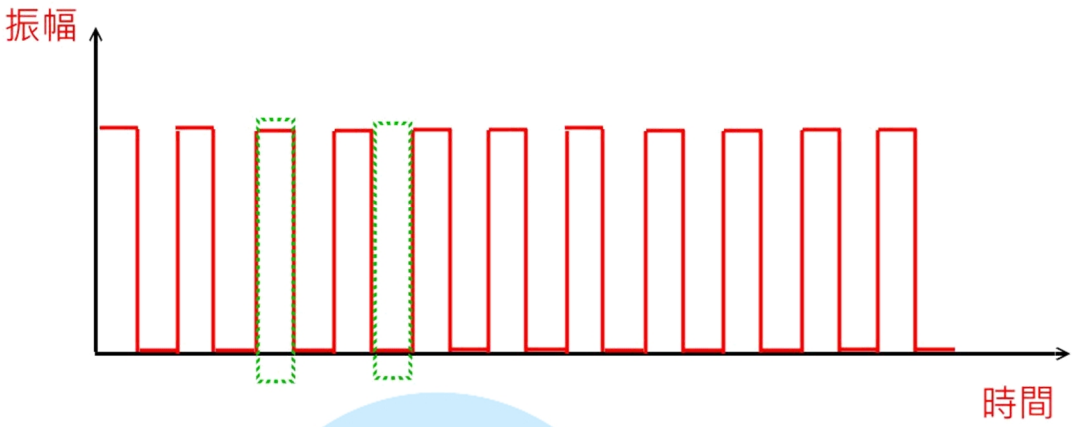

圖 1. 一個簡單的聲音波動圖

圖中的聲音只是一種簡單的高低的方波型構成，有點像是之前談過的PWM，這個圖的波只有兩種，高態和低態，不管哪一種，都算是一次震動，只是差別震動幅度高低而已，頻率的定義就是: 一秒鐘裡面發生多少次這樣的震動，EX: 300Hz 代表一秒鐘有三百次震動次數，頻率越高，代表震動的次數越多，我們的耳朵聽到的聲音就會越尖銳，相對的頻率越低聽到的聲音就會越低沉。而震動的幅度代表了聲音的強度，震動的幅度越大，我們就會聽到越大的聲音。這就好像我們用手拍水面，輕輕拍，水波很低緩；但用力拍，水就會波動劇烈是一樣的。

再來就是音色(聲音的特色)，如圖 2，同樣的音高和同樣的聲音強度的情況下，音色會讓一段聲音明顯被分辨出是不同樂器或是不同的人聲，這主要是因為發出聲音的材質不同，雖然發出一樣頻率的聲音，但該材質的緻密度、剛性、彈性等等的物理性質不同使得發出的聲音裡面會有額外的”特色雜音”混在其中，形成特色頻率。就好像唱歌，雖然同樣都唱出 Do、Re、Me、Fa、So…的音，但男生唱出來混有男生特有的低沉音在內，女聲唱出混有女生特有的高音在內，都會使得一段同樣旋律的聲音出來有明顯不同；各種大提琴、簫、吉

他、鋼琴等等也是因為這樣。這些特色音裡面的這些特色雜音，因為通常含蓋了中低高等音頻，甚至有些幾乎要超過人耳能聽到的範圍，所有的這些聲音若是以特定比例混合起來，雖然同時都發出一樣聲調，可以因為彼此音色的特性，使的整體的聲音變好聽，這也是為何樂團演奏總喜歡一堆各式樂器混和去搭配的原因，整體演奏的感覺好很多。

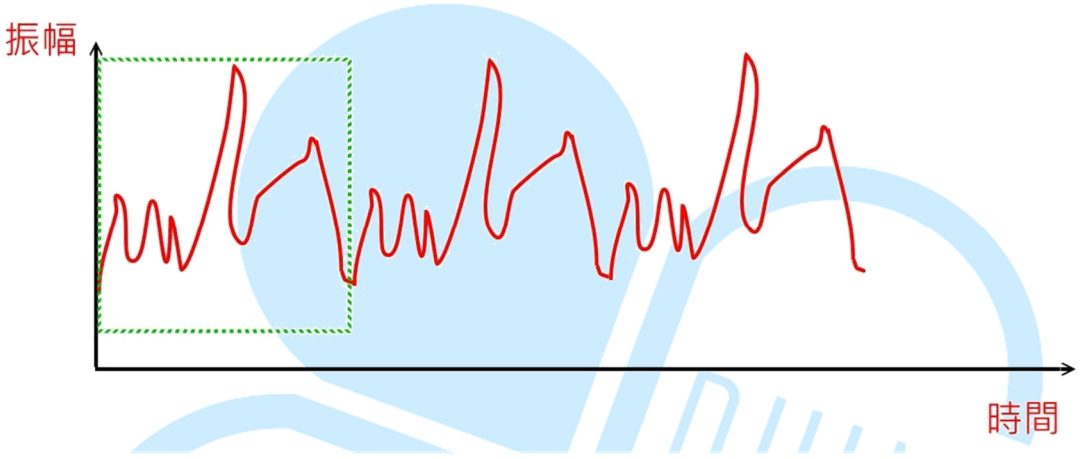

圖 2. 聲音的特色

有了頻率、振幅、音色，其實我們就可以把很多美妙的聲音重現出來。控制振幅，也就是音量大小(對控制來說就是改變電流大小或是 PWM)，可以模擬出聲音遠近的變化感覺，而變化的速度，還可以讓人耳有速度的感覺，就好像警車或是救護車，我們可以聽聲音就知道他們接近或遠離的速度是一樣的，不同的是，這裡面除了音量的變化還帶有都普勒效應的頻率變化。也可以利用左右兩個喇叭來配合，更能營造出立體的環場和左右方向感覺的聲音。甚至反推這個過程就可以去作聲音的辨識，甚至語音的辨識了，不過後面這些不是本章討論範圍，往後再談。

二、 發出聲音

知道聲音的特性以後，就可以利用 EduCake 的內建功能來實作出這些聲音，首先需要了解兩個指令 tone( )、notone( )。

tone( ) 這個指令會在指定的 digital pin 腳上產生一個指定頻率的方波(工作週期是 50% 的 PWM 信號)，用來模擬出不同頻率的聲音，可以指定持續發出聲音的時間或者一直持續直到呼叫 noTone()，這個 pin 腳上就可以接普通的小蜂鳴器(這種音質通常很難聽，只適合用來發出嗶嗶之類聲音)或者喇叭來發出一段聲音，後者聲音的品質明顯比較好。另外須注意同一時間只能產生一段聲音，如果已經有某個 pin 腳上在播放聲音，則想要同時在不同 pin 腳呼叫 tone() 是無效的，所以若想要在多個 pin 腳上播放不同的聲音，需要在呼叫前先使用 noTone()關閉此 pin 腳的聲音，再用 tone( )到別的 pin 腳去播放，這個動作對於製作像是左右聲道的喇叭輪流播放特別重要。tone( )的使用方法主要有以下兩種。

tone(pin, frequency)

tone(pin, frequency, duration)

pin: 指定要產生音頻的 pin 腳

frequency: 聲音頻率(Hz)，unsigned int，須注意人耳只能聽到 20~20000 左右的範圍

duration: 音頻的持續時間，單位為毫秒，unsigned long 型別 (此參數為非必需)

首先把電路先接好如圖 3:

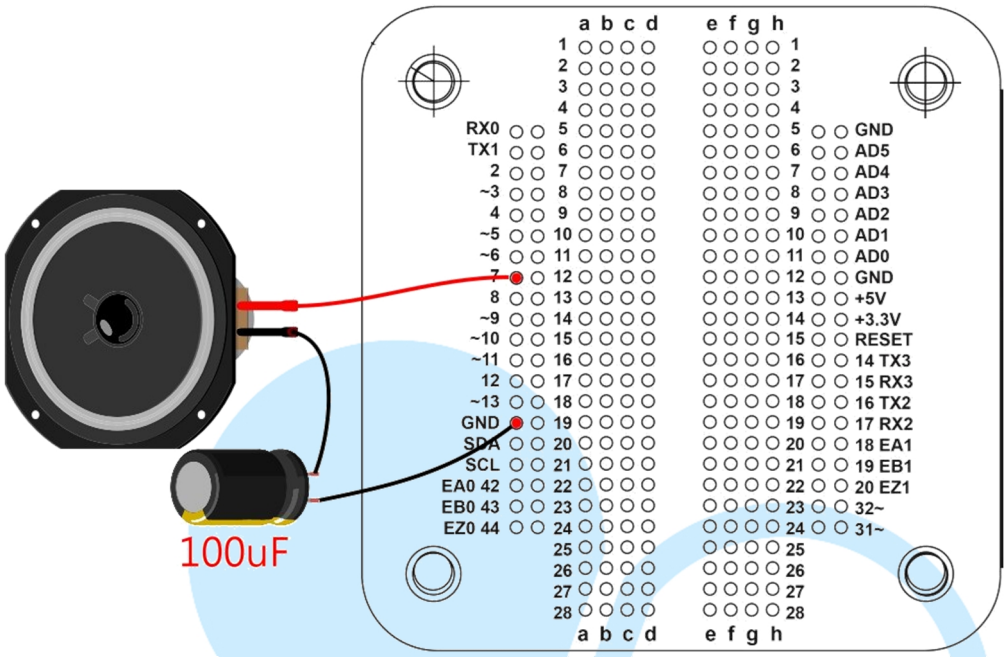

圖 3. 單喇叭接線圖

電路中那顆 100uF 的電解電容主要是濾波的效果，有些人則是直接只使用電阻，主要是怕喇叭的電阻過低，控制板的電流會太大，但會稍微降低音量，到底是搭配大小電容的串聯、併聯，或是加電阻上去調整，或是改變採用的喇叭尺寸的大小，或是用電晶體接電流放大電路去讓聲音更大聲更好控制等等，這部分就留給讀者自行去測試，各有不同的效果喔。若喇叭有接共振腔還會有不同效果，像有人會接大音箱或是小音箱，或是最近流行鸚鵡螺型狀的音箱都是為了比較好的音質效果而採用。而喇叭另外一條線接到 pin 7，到時候就可以使用 tone()指令來對 pin 7 作聲音的控制，這樣就可以實作出一組喇叭電路。簡易的測試程式碼如下

```
void setup()
{    }
void loop()
{
   tone(7, 500); //  令 pin7  不斷發出 500Hz 的聲音
   delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
   noTone(7); //  關閉 pin 7 的聲音
   delay(500);
}
```

若要改兩組喇叭模擬左右聲道，那電路也要改一下，這次使用電阻來調整音量，電路如圖 4

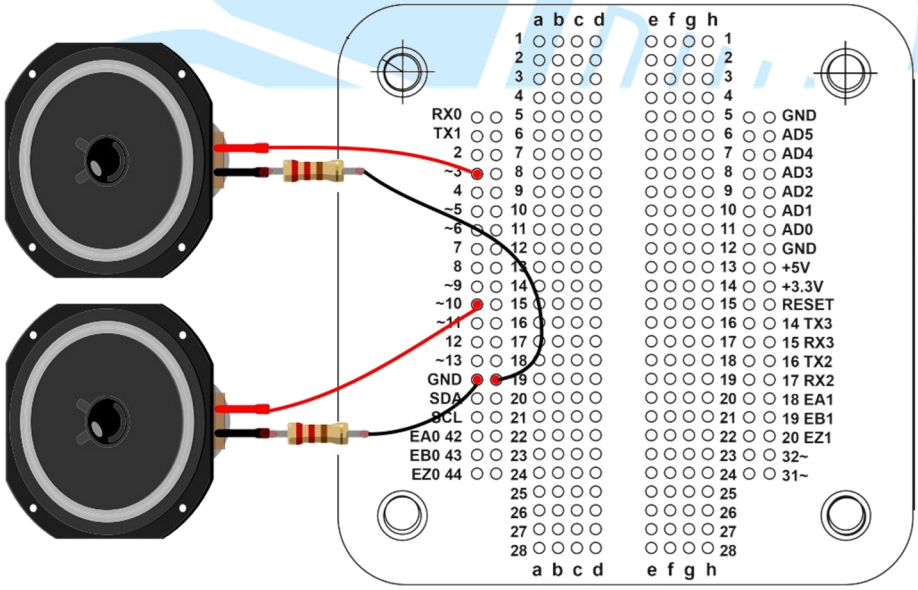

圖 4. 雙喇叭接線圖

搭配的程式碼就可以寫成這樣

```
int left_sound=10;
int right_sound=3;
void setup()
{    }
void loop()
{
  tone(left_sound, 500); //  令左邊喇叭不斷發出 500Hz 的聲音
  delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
  noTone(left_sound); //  關閉左邊喇叭
  tone(right_sound, 500); //  接著立刻打開右邊喇叭不斷發出 500Hz
的聲音
  delay(500); //  這個 delay 等於讓上面指令持續發出 0.5 秒聲音
  noTone(right_sound); //  關閉右邊喇叭
}
```

其中那個電阻變成可變電阻的話，就可以調整音量大小，若是用之前的章節談過的 serial 通訊，那就可以變成使用電腦的 UI + 滑鼠拖拉式的來改變音量了，應用的方式是很多的，就看讀者怎麼搭配前面談過的東西。

三、 音符和節拍

接下來想要播放音樂的話，必須先了解音符和節拍這兩件事情。節拍簡單講就是速度快慢，通常講一分鐘 180 拍，就是指一分鐘可以播放 180 個音符，所以很明顯節拍數越大，代表播放速度越快，相對就越慢，就可以利用控制節拍來控制播放速度，參照之前講的 tone 和 notone 中間使用的那個 delay()指令，裡面的數字其實就可以用來控制節拍和播放速度。

音符比較麻煩，一般以鋼琴鍵白色的部分(黑色的按鍵先不管)簡單來分就是低音(Do、Re、Me、Fa、So、La、Se)、中音、高音，各七個音共 21 音。頻率約是如下:

低音部分: 262、294、330、349、392、440、494(單位 Hz)

中音部分: 523、587、659、698、784、880、988

高音部分: 1046、1175、1318、1397、1568、1760、1976

這些音符的頻率是從維基百科查詢到的(http://zh.wikipedia.org/wiki/音符)，音符的相關知識很多，但這裡就不使用過多篇幅來研究，有興趣的讀者可以去更深入的研究看看。有了音符頻率表，再來就可以用程式來播出來看看，先來播看看低音的 Do~Se:

```
int frequency[]={
  262,294,330,349,392,440,494}; //  把低七音的頻率都先放到陣列裡面

void setup()
{
}
void loop()
{
  int a;
  for (a=0;a<7;a++)
  {
    tone(7, frequency[a]); //  從 pin7  發出聲音
    delay(500); //  每個音播放 0.5 秒
    noTone(7);
  }
}
```

這樣就可以把低音的 Do~Se 播出來聽聽看是什麼感覺了，用大喇叭，小喇叭或是迷你的蜂鳴器，聽到的聲音都不會一樣，讀者可以自行試試看。若是需要從低音到高音都一口氣播出來試試看，那就把裡面那個 frequency 的宣告改這樣

```
int frequency[]={
  262,294,330,349,392,440,494,
  523,587,659,698,784,880,988,
  1046,1175,1318,1397,1568,1760,1976};
```

然後後面那個 for 迴圈改這樣

```
for (a=0;a<7;a++)   改成   for (a=0;a<21;a++)
```

這樣就可以一次聽到所有的低中高音的標準音符頻率，有了頻率表，就可以開始來編輯歌譜讓 EduCake 唱歌了。

四、 迴圈由高至低、警車聲、弦波音

知道音符的播放後，我們還需要效果音，效果音其實還是各種頻率的聲音，只是可能連續、斷續、忽高忽低、起伏、亂數、甚至是以某種數學曲線計算去播放，可以作出各種不同的有趣效果，底下就一個一個來看。

前面有談過人耳能聽到的聲音介於 20~20000Hz 之間，這裡就寫個程式把這範圍的聲音播放出來聽聽看:

```
void setup()
{
}
void loop()
{
  int a;
  for(a=20;a<20000;a+=10) // 頻率 20~20000，每隔 10Hz 播出頻率
  {
    tone(7,a);
    delay(5);
  }
  noTone(7);
}
```

根據筆者測試，普通那種電腦用 6cm 小喇叭，大約到 12000Hz 以上耳朵就聽不太到了，要換比較小尺寸的喇叭才聽的到很尖銳很高的聲音。再來試試看嗶嗶的聲音:

```
void setup()
{
}
void loop()
{
    tone(7 ,900); //  調整裡面的 900 可以讓嗶嗶聲有不同感覺
    delay(200); //  控制嗶嗶聲的快慢
    noTone(7);
    delay(300);
}
```

可以發現嗶嗶聲其實是一開一關的聲音構成，裡面的頻率和 delay 的時間控制都可以玩出很多種不同的嗶嗶變化。 把這個聲音用迴圈包起來像上一個程式一樣令頻率變化，並加上加速度的感覺，可以製作類似高處掉下的滑音。

```
void setup()
{
}
void loop()
{
  int a;
  double v=0,s=3500; //  假設從 3500 米的高度掉下，初速為零，重力加速度 10
  while (s>50) //  掉到 50 米的高度就停
  {
    tone(7,s); //  每次直接把目前高度當作頻率輸出
    delay(30);
    noTone(7);
    s-=v; //  每次掉下 V 的距離
    v+=10; // V 每次都因為重力加速度增加 10
  }
}
```

測試結果可以聽出來這聲音還滿好玩的，而程式碼裡面的 tone(7,s); 這裡面的頻率如果替換成 tone(7,3500-s);，還會有不一樣的有趣效果，本來由高到低的滑音會變成由低到高，調整程式碼裡面的 V+=10 那個代表加速度的數字也會讓聲音變化率有特別的效果，當然，delay()裡面的數字也可以調整看看，都很有趣。這裡若加入使用三角函數的成分，還會有不一樣的效果:

```
void setup()
{
}
void loop()
{
   double a;
   for(a=0;a<100;a+=0.15)
   {
      tone(7, 1000+500*sin(a));
      //一圓周=2 pi =2x3.14=6.28 ，所以 100 約跑了 16 圈
      // sin(a) 會在+-1 之間變化 ，乘上 500 再加上一千以後，
      // 他會在 500~1500 之間作 sin 的弦波變化
      delay(50);
   }
   noTone(7);
}
```

感覺是不是比滑音又更有趣了?不同的弦波函數如 cos()、tan()等等，都會有不同表現喔。而生活中常聽到的警車救護車聲音，其實仔細分析也是這種忽高忽低的聲音構成，以下來試試看這個片段:

```
int spd =10; //  改變這個能直接影響循環快慢
void setup()
{
}
void loop()
{
   int a;
   for(a=420;a<1300;a+= spd)
   {
      tone(7,a);
      delay(20);
   }
   for(a=1300;a>500;a-= spd)
   {
      tone(7,a);
      delay(30);
   }
   noTone(7);
}
```

程式碼裡面的 delay 和迴圈裡面的數字調整一下，就可以模擬出各種警車或是救護車的聲音囉，不過，通常我們實際聽到的和這個有點不同，因為警車或是救護車除了高低起伏的聲音以外，還會搭配高速的接近或是遠離我們，這會有都普勒效應，頻率會有稍微的改變，所以若想模擬警車聲音接近的話，就得讓整體頻率慢慢變高；遠離則是慢慢變低，就會有類似效果。相關的實際公式就請參照高中物理課本，裡面會有詳細說明和公式，或是查詢這個維基網頁:

http://zh.wikipedia.org/wiki/都卜勒效應

以下這段程式碼用來亂數產生不可預期的聲音，很有電影裡面機器人講話的感覺

```
void setup()
{
  randomSeed(analogRead(0));
}

void loop()
{
  tone(7, random(100,2000));
  delay(100);
  noTone(7);
}
```

裡面的 random(100,2000) 會產生 100~1999 之間的亂數頻率，改變這兩個數值或是改變 delay()裡面的時間都會有很奇特的聲音出現。差不多所有的一般聲音都可以藉由以上這些小技巧去測試合成，也讓靜悄悄的單晶世界有更多樂趣。

五、 放一首歌，歌譜分析

上面談過各種音頻控制，這裡我們就實際來讓 EduCake 唱出一首歌來試試看。首先，我們先要有一個歌譜，這到 GOOGLE 上面到處抓都會有，只是歌曲幾乎都要注意版權問題，所以雖然可以抓到很多歌，卻不能直接拿來使用。找很久最後發現一些很久很久以前的童謠就比較沒有版權問題，可以直接使用來說明或是自己玩，但這若是要販賣恐怕還是不行的。

以下就以兩隻老虎這首經典童謠來說明相關的細節，如下圖 5。因為這也不是音樂課，我們不需要看懂整張圖裡面所有的細節，只要幾個重點看懂就好。第一點就是圖中的歌詞上面都會跟著數字，像是”兩隻老虎”上面跟著 1231，這些 1234567 數字對應了歌譜的 Do、Re、Me、Fa、So、La、Se，只要照著這些數字對應的音符去敲鋼琴或是樂器，就能談出兩隻老虎這條有趣的童謠。第二點就是升降 KEY 問題，前面談過 Do~Se 七個音，還有分低音、中音和高音，歌譜裡面也需要有這些標住才能展現歌詞的高揚或是低沉的變化，注意第二行和第三行的歌詞裡面就有這些標記，第二行的”一隻沒有耳朵”上面跟的 565431，其中的 5654 底下有底線，這就說要升階，中音要變高音才對；第三行的”真奇怪”那個”奇”字上面的 5，下方有一點，那個點的意思就是要降階，中音要變低音。第三點，第一行和第二行裡面的”跑的快”後面都有接著”-“，這是代表前一個音要多播放一拍，一般每個字是一拍，有這個符號就可以讓某個字兩拍，拖長音的意思，所以若是想要更長的音就兩個，再長就三個，依此類推。

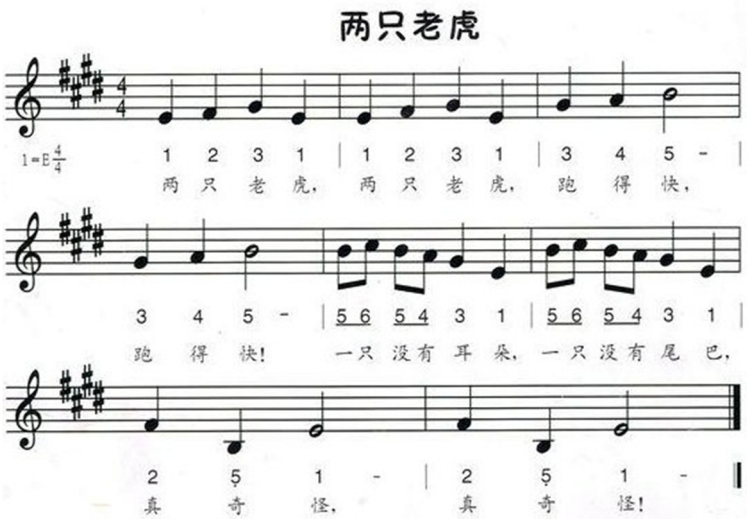

圖 5. 兩隻老虎五線譜

有了歌譜再來就要把歌譜想辦法轉換成真正的程式碼，由上面的歌譜和解說資訊，我們可以大概的整理出每個歌詞裡面的字都需要音頻(高低音)和節拍(長短)，並將他們宣告成程式可以使用的陣列如下

```
byte tigerTone[]= //記錄所有音符的頻率對應索引值
{7,8,9,7,7,8,9,7,9,10,11, 9,10,11,18,19,18,17,9,7,
18,19,18,17,9,7,8,4,7,8,4,7};

byte tigerBeat[]=// 記錄每一個音符需要播放多長拍數，程式裡面再看要定義每分鐘
幾拍，就可以控制撥放速度
{1,1,1,1,1,1,1,1,1,2,1,1,2, 1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
```

因為前面已經使用完整的 frequency[]陣列記錄低中高音的頻率，共存放 21 個數值，這裡的 tigerTone 陣列裡面只要紀錄對應的號碼就可以順利播放，其中 0~6 對應低音的 Do~Se、7~13 對應中音的 Do~Se、14~20 對應高音的 Do~Se，組合起來的程式碼如下:

```
    const int speaker=13; //  喇叭改到 13 號腳位輸出，為了後面的整合作準備
    int frequency[]={   //  把低中高音的音頻先建立表格
      262,294,330,349,392,440,494,
      523,587,659,698,784,880,988,
      1046,1175,1318,1397,1568,1760,1976};
    byte tigerTone[]=
    {7,8,9,7,7,8,9,7,9,10,11, 9,10,11,18,19,18,17,9,7,
18,19,18,17,9,7,8,4,1,8,4,1};
    byte tigerBeat[]=
    {1,1,1,1,1,1,1,1,1,2,1,1,2, 1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
    const int playLen=sizeof(tigerTone); //  播放長度先計算出來，需注意這是實際占記憶體長度

    void setup()
    {
    }
    void loop()
    {
      int a;
      for(a=0;a< playLen -1;a++)
      {
       tone(speaker,frequency[tigerTone[a]]); //從預設的頻率表裡面索引目前要播放的
       delay(300* tigerBeat[a]); //  每個音符的延遲時間，延的越久這個音就播越久
       //  約是一秒播三個音符，每分鐘 180 個
       noTone(speaker); //關閉聲音，等待下一個音符，也可以最後在關
      }
      delay(3000);
    }
```

這樣作就能完整播放一首歌了，播完以後，最後那個 delay 指令會等待三秒鐘又會繼續在播一次。如果把這整個流程都包裝起來，就可以利用 SD 卡儲存大量的音樂，用按鈕來選擇要播哪一首，搭液晶螢幕甚至觸控面板，就是一台點歌器了，後面我們會有章節實際完整的來作這個有趣的東西。

六、 多首同播

前面講到點歌器的概念，這邊就來實際作看看簡單的版本。首先，得先有幾首歌，然後還要有對應的按鈕來選歌，或是很多首歌的話也可以利用 4*4 之類的數字鍵盤來選歌。要能選擇歌曲播放，首先得先要有多首歌譜，為了方便說明起見，以及配合後面的功能，我們安排四首歌:兩隻老虎、小星星、小蜜蜂、小毛驢這些經典童謠。先來建立每一首歌的頻率對應表格和播放節拍表

```
int frequency[]={   // 把低中高音的音頻先建立表格
  262,294,330,349,392,440,494,
  523,587,659,698,784,880,988,
  1046,1175,1318,1397,1568,1760,1976};

byte tigerTone[]={7,8,9,7,7,8,9,7,9,10,11, 9,10,11,18,19,18,17,9,7,
18,19,18,
17,9,7,8,4,1,8,4,1};// 兩隻老虎
byte tigerBeat[]={1,1,1,1,1,1,1,1,1,2,1,1,2,
1,1,1,1,1,1,1,1,1,1,1,1,1, 2,1,1,2};
int   tigerLen =sizeof(tigerTone) ;   // 順便把這首歌的長度計算出來

byte
beeTone[]={ 11,9,9,10,8,8,7,8,9,10,11,11,11,11,9,9,10,8,8,7,9,11,11,9,8,8
,8,8,8,9,
10,9,9,9,9,9,10,11,11,9,9,10,8,8,7,9,11,11,7};   // 小蜜蜂
byte
beeBeat[]={ 1,1,2,1,1,2,1,1,1,1,1,1,2,1,1,2,1,1,2,1,1,1,1,4,1,1,1,1,1,1,2,1,1,
1,1,1,1,2,1,1,2,1,1,2,1,1,1,1,4};
int beeLen=sizeof(beeTone) ;
```

```
        byte
starTone[]={7,7,11,11,12,12,11,10,10,9,9,8,8,7,11,11,10,10,9,9,8,11,11,1
0,
        10,9,9,8,7,7,11,11,12,12,11,10,10,9,9,8,8,7}; //小星星
        byte
starBeat[]={1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,
        1,1,2,1,1,1,1,1,1,2};
        int starLen=sizeof(starTone) ;

        byte
donTone[]={7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,12,12,9,9,9,9,8,8,8,
8,
        11,11,7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,10,12,9,9,9,9,9,8,8,8,
9,7};//小毛驢
        byte
donBeat[]={ 1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,
1,
        1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,4};
        int donLen=sizeof(donTone) ;
```

建立好這些以後就可以開始用程式來播放，也可以把這些資訊寫入檔案放到 SD 卡裡面，就可以利用 SD 卡的超大容量放置很多首歌來提供隨選播放了。

之前的程式只能播放一首歌，但現在我們有四首歌了，程式的編排也要針對多首歌的使用來重新思考。

首先我們想要用一些按鈕來對應每首歌，按 1 就播放兩隻老虎、按 2 就播放小蜜蜂、按 3 就播放小星星...依此類推。但因為每首歌長度並不相同、按鈕要按下就要重頭開始播放對應的歌，而且後面還需要能夠有類似暫停、修改音樂播放的內容或是升降音等等功能，所以需要把程式碼重新安排成類似這樣:

```
int play_no=-1; // 記錄現在是在播放哪一首，-1 代表沒有在播放
int play_pos=0; // 目前播放到歌曲的哪個位置
int is_play=0; // 是否有在播放

int bb[4]={2,3,4,5}; //四個按鍵的腳位定義在 digital 的 2~5
int bb_state[4]; // 記錄每個按鍵的狀態

void setup()
{
  int a;
  Serial.begin(9600);
  for(a=0;a<4;a++)
    pinMode(bb[a], INPUT_PULLUP);
}

void loop()
{
  int a;
  for(a=0;a<4;a++) // 先掃描這些按鍵是否有被按下
   {
   bb_state[a]=digitalRead(bb[a]);   //把按鍵狀態存到陣列中
   }

  // 接下來就判斷每個按鍵的狀態來作處理，這裡假設使用者一次只按一個按鍵
  // 一次只能播一首歌，當然也可以安排兩鍵一起按下，一次兩首歌混在一起播放
```

```
// 但這樣就聽不太出來到底在播什麼，所以先不要這樣作
 if (bb_state[0]==0) // 第一個按鍵被按下，要播第 1 首歌
 {
    Serial.println(beeLen); // 印出相關的簡單訊息到監視畫面
    play_no=1;    // 設定第 1 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("bee playing...");
 }
 else if (bb_state[1]==0) // 第二個按鍵被按下，要播第 2 首歌
 {
    Serial.println(starLen);
    play_no=2;    // 設定第 2 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("star playing...");
 }
 else if (bb_state[2]==0) // 第三個按鍵被按下，要播第 3 首歌
 {
    Serial.println(tigerLen);
    play_no=3;    // 設定第 3 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("tiger playing...");
 }
 else if (bb_state[3]==0) // 第四個按鍵被按下，要播第 4 首歌
 {
    Serial.println(donLen);
    play_no=4;    // 設定第 4 首歌要播
    play_pos=0;   // 位置歸零，從頭開始播放
    is_play=1;   // 1 代表可以播，0 代表不播放
    Serial.println("don playing...");
 }
```

```
    playSong(); //  呼叫歌曲播放函數
      delay(5);
  }

  // 這個函數處理播放一個單音的動作，每個音符會有一個頻率和想要播放的持續時間
  void play (byte toneNo, byte   beatNo)
  {
    tone(speaker,frequency[toneNo]);
    delay(300* beatNo);
    noTone(speaker);
  }

  // 這個函數專門處理播放的相關流程和問題
  void playSong()
  {
    if (is_play==1) //   1 代表現在可以播放歌曲，0 代表不播放
    {
      switch(play_no)   // 針對 play_no 的內容來決定要播放哪一首歌
      {
      case 1://beeTone
        if   (play_pos>=beeLen) // 判斷目前播放的位置是否已經到最後位置了
          {
            is_play=0; // 如果已經到歌曲最後位置，設定停止播放
            return ; //離開函數
          }
          //若還沒播完，就播放目前位置的音符
          play (beeTone[play_pos],beeBeat[play_pos]);
          play_pos++; //然後把播放的位置+1  ，指向下一個音符位置
          break;
      case 2://starTone
        if   (play_pos>=starLen)
```

```
            {
                is_play=0;
                return ;
            }
            play (starTone[play_pos],starBeat[play_pos]);
            play_pos++;
            break;
          case 3://tigerTone
            if   (play_pos>=tigerLen)
            {
                is_play=0;
                return ;
            }
            play (tigerTone[play_pos],tigerBeat[play_pos]);
            play_pos++;
            break;
          case 4://donTone
            if   (play_pos>=donLen)
            {
                is_play=0;
                return ;
            }
            play (donTone[play_pos],donBeat[play_pos]);
            play_pos++;
            break;
        }
      }
    }
```

這樣就可以按下 1~4 號按鈕，對應播放上面設定好的童謠，並且歌曲播放完畢會自動停止，播放過程中，還可以隨時按下其他的按鈕來切換播放不同的歌曲，是不是很有趣呢? 這裡還可以思考看看改用不同的方式，讓歌曲的種

類和選擇方式變很多樣化，像是使用紅外線遙控器來搖控，這需要結合紅外線接收感應器材行。也可以使用連續旋轉的旋轉編碼器，類似汽車音響那樣，用轉的來選擇歌曲或是頻道，也可以修改程式，多一些按鈕可以設定像是播放順序的設定；或是播完自動輪播；或是某首歌曲設定不斷重複播放...之類的進接使用者自訂功能，這都有助於讓音樂播放器更好玩更貼近使用者的需求。

七、 控制速度、混音、調音量、升降 KEY 等等

能夠播放多首歌曲以後，我們就可以開始來加入一些特效，讓整個播放器的功能能夠更豐富有趣。在一般 PUB 或是演唱會現場，常會看到有鍵盤手或是混音調音人員在按著很複雜的大按鈕盤，上面可能一堆按鈕或是一堆轉盤旋鈕之類的東西，可以讓人很輕易的在歌曲或是舞曲中加入各種特效聲音像是動物叫聲、物品撞擊聲、金屬音、搖滾節奏等等，不管是哪一種，其實原理都是一樣的，一首音樂就像上面講的是一堆音符構成，若我們把想要混入的特效音也編成音符，加入到正在播放中的音樂裡面，就可以結合音樂和特效音。加入的方法是，直接把音樂和特效音的頻率結合起來，可能是直接相加除以 2；或是兩者乘上一個權重以後相加，例如音樂占 7 成，特效音 3 成之類的，這樣還可以有個特效音權重設定的功能可以加入。但在我們這個範例裡面這種作法的效果都不會好，因為解析度太低了，前面的範例都一秒才播放三個音左右，等於取樣頻率 3Hz/sec，一般的音樂早就都 22KHz/sec 或以上了，這就好像照相機的解析度越高畫質越好一樣，取樣頻率越高也代表聲音的音質會越好，但這又牽涉到複雜的音訊處理和音樂取得問題，這邊先跳過，我們先從簡單的實作來處理。

首先是控制速度，這可以輕易的由 void play (byte toneNo, byte beatNo) 這個函數下手，裡面的 beatNo 就是控制音樂播放速度的關鍵，調整個這數字的大小就可以直接控制音樂的快慢，想像播放超快的兩隻老虎，或是超慢的小毛驢，其實很有趣的，這只要針對這個變數宣告一個 play_spd 來控制就可以了。

再來是混音，前面講過取樣頻率不高的時候效果不好，我們這裡就用不同方式，改作特效音，提供一個高音的彈跳聲音和一個低音的彈跳聲音，給使用者按下的時候直接插入播放，一樣可以有不同效果呈現。彈跳音的作法就如同前面講過的類似救護車的滑音或是亂數音，只是把迴圈的頻率壓在低頻可能是 200Hz 附近、高頻壓在 1200Hz 附近，也是有不錯效果。

調整音量的部分，其實就是輸出功率的控制，但因為這裡是直接由腳位輸出，功率很低，要調音量不容易作。若真的想處理就是要找一些音頻功率放大晶片來放大，搭可變電阻就可以簡單的處理音量。或是也可以採用現成的音頻放大模組，如圖 6

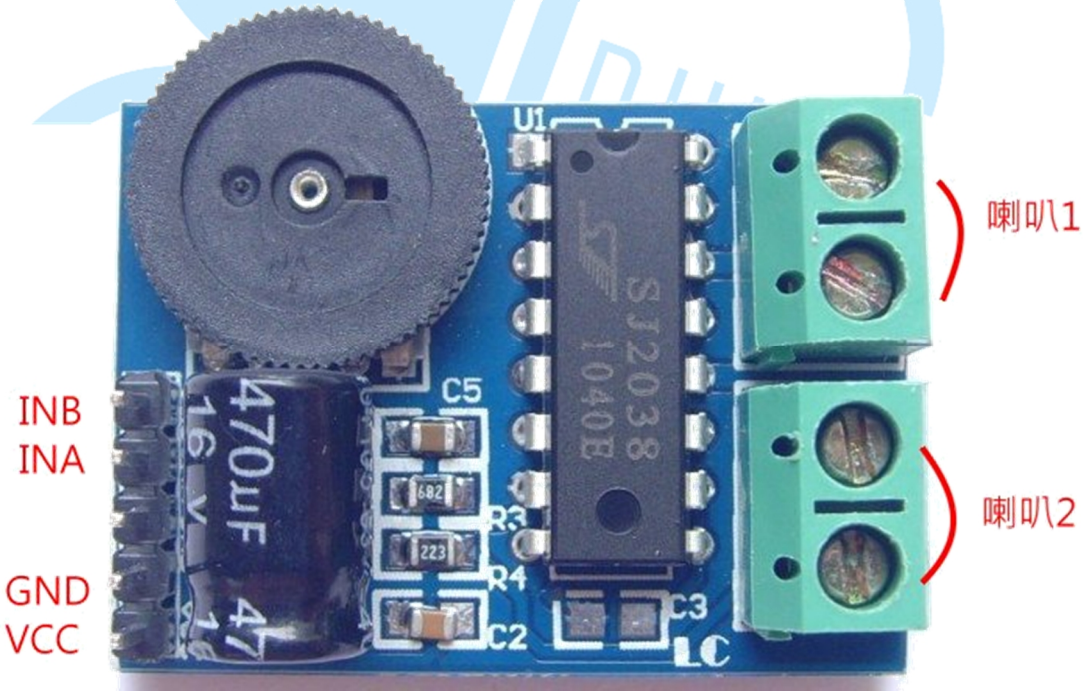

圖 6. 音頻放大模組圖

想要升降播放的 KEY，那就得要另外宣告變數來控制，這裡有兩種作法可以參考，一種是直接把頻率索引表裡面的數值調升或是調降，會有很明顯的變化，但有個問題是 0 這個索引值已經對應到頻率表裡面最低的一個頻率，沒有調降空間了，若是遇到這個頻率就調不了。同樣的 20 這個索引值已經對應到

最高的頻率，一樣沒有調高空間。所以另外一個作法是直接宣告一個變數 play_keys 來作處理，這變數可以預設為 0，代表頻率沒有變化，但讓這個變數可以在+-300~500 之間或是更大的範圍內受到按鈕調整，然後播放頻率的時候直接把這個變數加上去，就可以直接把接下來的每一個音符都作聲音頻率的調升/調降。完整電路圖如圖 7

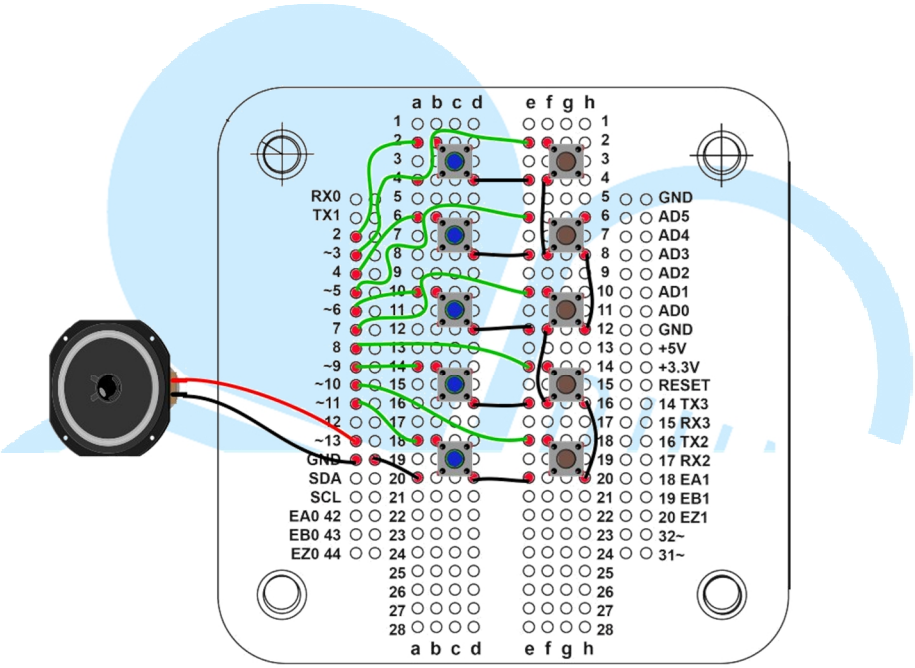

圖 7. 完整線路圖

完整程式碼如下

```
int speaker_pin=13;

int frequency[]={  // 把低中高音的音頻先建立表格
  262,294,330,349,392,440,494,
  523,587,659,698,784,880,988,
  1046,1175,1318,1397,1568,1760,1976};

byte tigerTone[]={7,8,9,7,7,8,9,7,9,10,11, 9,10,11,18,19,18,17,9,7,
18,19,18,
17,9,7,8,4,1,8,4,1};// 兩隻老虎
byte tigerBeat[]={1,1,1,1,1,1,1,1,1,1,2,1,1,2, 1,1,1,1,1,1,1,1,1,1,1,1,1,
2,1,1,2};
int  tigerLen =sizeof(tigerTone) ;  // 順便把這首歌的長度計算出來

byte beeTone[]={
11,9,9,10,8,8,7,8,9,10,11,11,11,11,9,9,10,8,8,7,9,11,11,9,8,8,8,8,8,9,
10,9,9,9,9,9,10,11,11,9,9,10,8,8,7,9,11,11,7};  // 小蜜蜂
```

```
        byte
beeBeat[]={ 1,1,2,1,1,2,1,1,1,1,1,1,2,1,1,2,1,1,2,1,1,1,1,4,1,1,1,1,1,1,2,1,1,
        1,1,1,1,2,1,1,2,1,1,2,1,1,1,1,4};
        int beeLen=sizeof(beeTone) ;

        byte
starTone[]={7,7,11,11,12,12,11,10,10,9,9,8,8,7,11,11,10,10,9,9,8,11,11,10,
        10,9,9,8,7,7,11,11,12,12,11,10,10,9,9,8,8,7}; //小星星
        byte
starBeat[]={1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,
        1,1,2,1,1,1,1,1,1,2};
        int starLen=sizeof(starTone) ;

        byte
donTone[]={7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,12,12,9,9,9,9,8,8,8,8,
        11,11,7,7,7,9,11,11,11,11,12,12,12,13,11,10,10,10,12,9,9,9,9,9,8,8,8,9,
7};//小毛驢
        byte
donBeat[]={ 1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,
        1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1,1,1,1,4};
        int donLen=sizeof(donTone) ;

        int play_spd =0; //  速度控制
        int play_no=-1; //  記錄現在是在播放哪一首，-1 代表沒有在播放
        int play_pos=0; //  目前播放到歌曲的哪個位置
        int is_play=0; //  是否有在播放
        int play_keys=0; //  音頻升降控制-300~+600

        int bb[10]={
           2,3,4,5,6,7,8,9,10,11}; //  十顆按鈕對應到:1~4 號歌曲、升降、速度和
2 種特效音
        int bb_state[10]; //對應十顆按鈕的按下狀態
```

```
    void setup()
    {
      int a;
      Serial.begin(9600);
      for(a=0;a<10;a++)
        pinMode(bb[a], INPUT_PULLUP); //利用內部的上拉電阻來設定
按鈕
    }

    void loop()
    {
      int a;

      for(a=0;a<10;a++) //  先掃描全部的按鈕狀態
          bb_state[a]=digitalRead(bb[a]);

      for(a=0;a<10;a++) //  輸出所有按鈕狀態到監控視窗
      {
        Serial.print( bb_state[a]);
        Serial.print(", ");
      }
      Serial.println("");

      if (bb_state[0]==0) //播第 1 首歌
      {
        Serial.println(beeLen);//  印出相關的簡單訊息到監視畫面
        play_spd =0; //重設速度控制變數
        play_no=1;   //  設定第 1 首歌要播
        play_pos=0;   //  位置歸零，從頭開始播放
        play_keys=0; //  把升降 KEY 先歸零
        is_play=1; // 1 代表可以播，0 代表不播放
        Serial.println("bee playing...");
      }
```

```
        else if (bb_state[1]==0) //播第 2 首歌
          {
             Serial.println(starLen);
             play_spd =0;
             play_no=2;
             play_pos=0;
             play_keys=0;
             is_play=1;
             Serial.println("star playing...");
          }
          else if (bb_state[2]==0) //播第 3 首歌
          {
             Serial.println(tigerLen);
             play_spd =0;
             play_no=3;
             play_pos=0;
             play_keys=0;
             is_play=1;
             Serial.println("tiger playing...");
          }
          else if (bb_state[3]==0) //播第 4 首歌
          {
             Serial.println(donLen);
             play_spd =0;
             play_no=4;
             play_pos=0;
             play_keys=0;
             is_play=1;
             Serial.println("don playing...");
          }
          else if    (bb_state[4]==0) //  升 KEY
          {
             if ( play_keys<600)
```

```
        play_keys+=50;
            else
               play_keys=600; //最高到 600，當然可以設定更高，但過高會
太尖銳聽不太到
          }
          else if   (bb_state[5]==0) //  降 KEY
          {
            if (play_keys<-300)
               play_keys=-300;
            else
               play_keys-=50;
          }
          else if   (bb_state[6]==0)//  速度降低
          {
            play_spd +=100;
            if (play_spd >1000)
               play_spd =1000;
          }
          else if   (bb_state[7]==0)//  速度提高
          {
            play_spd -=50;
            if (play_spd <-200)
               play_spd =-200;
          }
          else if   (bb_state[8]==0)//  特效音 1 ，時間約 0.6 秒
          {
            for(a=0;a<5;a++)   //  產生低音的碰碰聲
            {
               tone(speaker, 150);
               delay(50);
               tone(speaker, 250);
               delay(50);
            }
```

```
    }
      else if   (bb_state[9]==0)//  特效音 2  ，時間約 0.6 秒
      {
        for(a=1000;a<1600;a+=20)   //製造高音的滑音效果
        {
          tone(speaker, a);
          delay(20);
        }
      }

      playSong();//  呼叫歌曲播放函數
      delay(2);
  }

  //  這個函數處理播放一個單音的動作，每個音符會有一個頻率和想要播放的持續時間
  void play (byte toneNo, byte   beatNo)
  {
    int pk=play_keys+frequency[toneNo]; //  先把頻率和升降 key 結合
    tone(speaker, pk);
    // delay 裡面把速度控制變數加入，即可改變播放速率
    delay(300* beatNo+ play_spd);
    noTone(speaker);
  }

  //  這個函數專門處理播放的相關流程和問題
  void playSong()
  {
    if (is_play==1) //   1  代表現在可以播放歌曲，0 代表不播放
    {
      switch(play_no)   //  針對 play_no 的內容來決定要播放哪一首歌
      {
      case 1://beeTone
```

```
        if   (play_pos>=beeLen) //  判斷目前播放的位置是否已經到最後
位置了
              {
                is_play=0; //  如果已經到歌曲最後位置，設定停止播放
                return ; //離開函數
              }
              //若還沒播完，就播放目前位置的音符
              play (beeTone[play_pos],beeBeat[play_pos]);
              play_pos++; //然後把播放的位置+1  ，指向下一個音符位置
              break;
            case 2://starTone
              if   (play_pos>=starLen)
              {
                is_play=0;
                return ;
              }
              play (starTone[play_pos],starBeat[play_pos]);
              play_pos++;
              break;
            case 3://tigerTone
              if   (play_pos>=tigerLen)
              {
                is_play=0;
                return ;
              }
              play (tigerTone[play_pos],tigerBeat[play_pos]);
              play_pos++;
              break;
            case 4://donTone
              if   (play_pos>=donLen)
              {
                is_play=0;
                return ;
```

```
        }
            play (donTone[play_pos],donBeat[play_pos]);
            play_pos++;
            break;
          }
        }
      }
```