Ondes sonores : Produire & Exploiter

Après avoir bien travaillé, maintenant que nous maîtrisons le principe des entrées et sorties de l'Arduino, nous allons nous détendre en écoutant un peu de musique…

Activité 1 : Faire clignoter une DEL et produire un "beep" synchrone

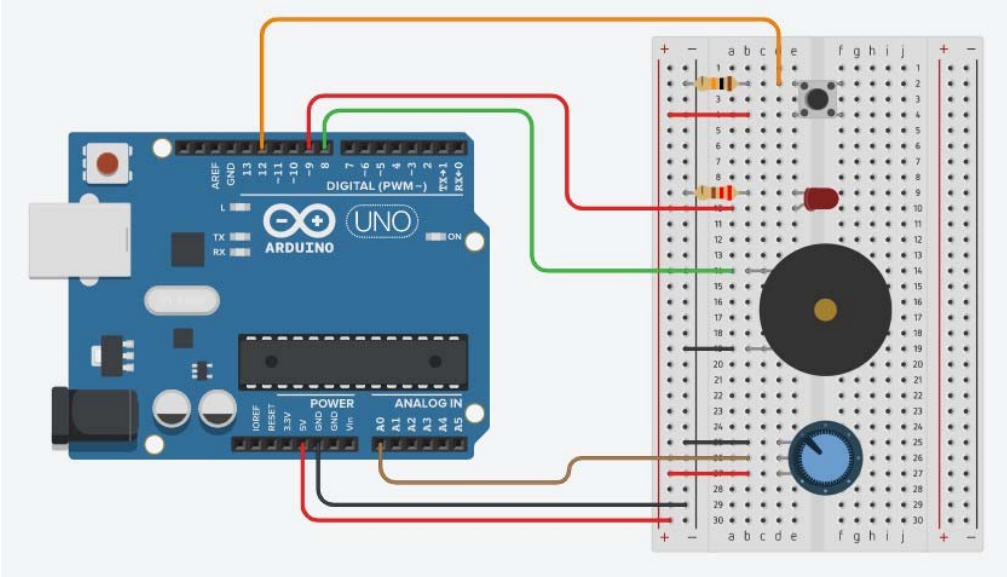
Dans cette activité, nous allons voir qu'il est possible d'émettre un son, caractérisé par sa fréquence en Hz, avec un Arduino et modifier le premier programme qui permet de faire clignoter une DEL, pour commander la production d'un signal sonore ("un beep"), émis par un buzzer ou un petit haut-parleur, synchrone avec le clignotement de la diode.

Comme une corde de guitare qui vibre et qui transmet sa vibration à l'air, pour produire un son avec un Arduino, il faut utiliser un matériel qui peut vibrer sur commande !

Pour cela on utilise un petit haut-parleur ou un buzzer (transducteur) piézo-électrique (communément appelé "piezo") connecté sur une des sorties de l'Arduino.

Il existe différents types de buzzer, les actifs qui nécessitent une alimentation et les passifs sans alimentation.

Nous allons donc ajouter un petit haut-parleur ou un buzzer connecté sur la broche 8 de l'Arduino à notre circuit d'apprentissage :



. Gestion du son en langage Arduino

. **La fonction tone()** :

Le signal électrique appliqué par l'Arduino sur une de ses sorties digitales ou analogiques, sur laquelle est connecté le piezo ou le haut-parleur et qui va créer l'onde sonore, est réalisé avec la fonction **tone()**.

Cette fonction génère une onde carrée (onde symétrique avec "duty cycle" (niveau haut/période) à 50%) à la fréquence spécifiée en Hertz (Hz) sur une broche. La durée peut être précisée, sinon l'impulsion continue jusqu'à l'appel de l'instruction **noTone()**.

Une seule note peut être produite à la fois. Si une note est déjà jouée sur une autre broche, l'appel de la fonction **tone()** n'aura aucun effet (tant qu'une instruction **noTone()** n'aura pas eu lieu).

Si la note est jouée sur la même broche, l'appel de la fonction **tone()** modifiera la fréquence jouée sur cette broche.

Enfin, l'utilisation de **tone()** rend impossible l'utilisation des broches **D3** et **D11** en PWM avec **analogWrite()**.

. Syntaxe :

tone(broche, fréquence)
tone(broche, fréquence, durée)

. Paramètres :

Broche : la broche sur laquelle la note est générée.
Fréquence : la fréquence de la note produite, en hertz (Hz)
Durée : la durée de la note en millisecondes (optionnel)

. **La fonction noTone()** :

La fonction **noTone()** stoppe la génération d'impulsion produite par l'instruction **tone()**. Elle n'a aucun effet si aucune impulsion n'a été générée.

. Syntaxe :

noTone(broche)

. Paramètres :

broche: la broche sur laquelle il faut stopper la note.

. Le programme

Le programme de l'activité (**"Activity1.ino" dans le dossier "Codes/Ondes Sonores"**) pourra être modifié pour voir l'influence des variables (durée d'allumage et d'extinction, fréquence en Hz de l'onde sonore) :

Activity1

```
// Déclaration des constantes et variables

const int PinLED = 9;
const int PinTone = 8;
const int FreqTone = 440;
const int TimeSleep1 = 500;
const int TimeSleep2 = 500;


// Initialisation des entrées et sorties

void setup()
{
  pinMode(PinLED, OUTPUT);
}


// Fonction principale en boucle

void loop()
{
  digitalWrite(PinLED, HIGH);
  tone(PinTone,FreqTone);
  delay(TimeSleep1);
  digitalWrite(PinLED, LOW);
  noTone(PinTone);
  delay(TimeSleep2);
{
```

<u>Déroulement du programme</u> :

- Déclaration des constantes et variables :

. **const int PinLed = 9** (constante nombre entier correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. **const int PinTone = 8** (constante nombre entier correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **const int FreqTone = 440** (constante nombre entier correspondant à la fréquence en Hz de l'onde sonore)

. **const int TimeSleep1 = 500** (constante nombre entier correspondant à la durée d'allumage de la Del en ms)

. **const int TimeSleep2 = 500** (constante nombre entier correspondant à la durée d'extinction de la Del en ms)

- Initialisation des entrées et sorties :

. La broche de la DEL est initialisée en sortie digitale. Des données seront donc envoyés depuis le microcontrôleur vers cette broche :

pinMode(PinLed, OUTPUT)

- Fonction principale en boucle :

. Niveau haut sur la broche de la Del : **digitalWrite(PinLed, HIGH)**

. Emission de l'onde sonore en continu : **tone(PinTone,FreqTone)**

. Attente pendant TimeSleep1 ms : **delay(TimeSleep1)**

. Niveau bas sur la broche de la Del : **digitalWrite(PinLed, Low)**

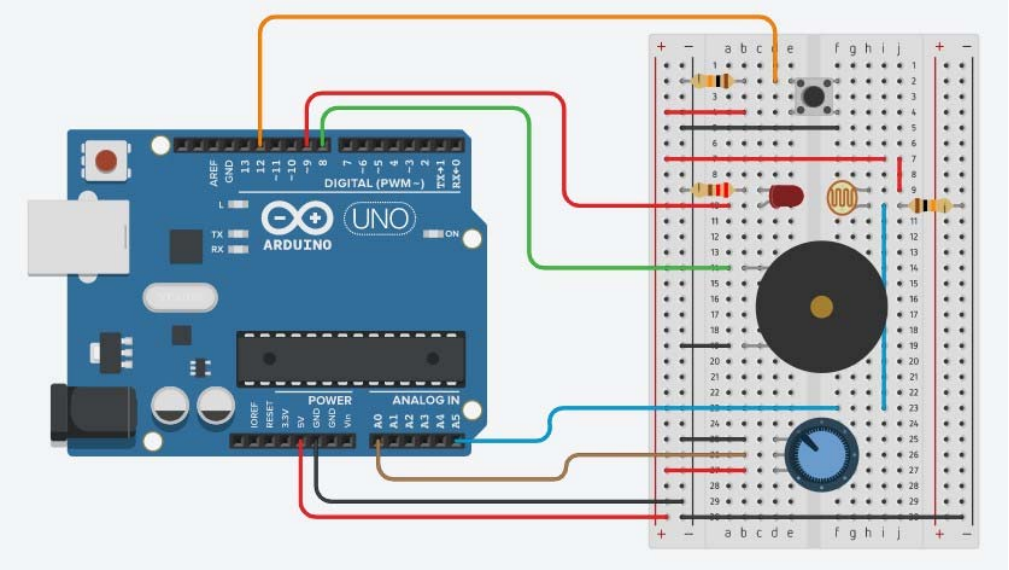
. Arrêt de l'émission sonore : **notone(PinTone)**

. Attente pendant TimeSleep2 ms : **delay(TimeSleep2)**

Activité 2 : Alarme sonore par détection de passage

Dans cette activité, le programme de production d'un "beep" de l'activité précédente va être utilisé comme alarme de détection de passage.

Pour cela, on va ajouter à notre montage une photorésistance qui sera éclairée par la DEL rouge. La sortie de la photorésistance est connectée à l'entrée analogique **A5** de l'Arduino :



La valeur de la broche **A5** est alors proportionnelle à l'intensité lumineuse reçue par la photorésistance.

En présence d'un obstacle entre la DEL et la photorésistance, la tension mesurée au niveau de la broche **A5** diminue et quand celle-ci est inférieure à un seuil (la sensibilité du capteur définie initialement), l'alarme sonore est déclenchée.

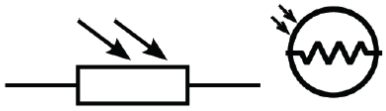
La photorésistance

Une photorésistance est un composant électronique dont la résistance en Ohm (Ω) varie en fonction de l'intensité lumineuse. Plus la luminosité est élevée, plus la résistance est basse. La photorésistance est un capteur résistif.

On peut donc l'utiliser comme capteur lumineux pour :

- Mesure de la lumière ambiante.
- Détecteur de lumière dans une pièce.
- Suiveur de lumière dans un robot.
- Détecteur de passage.
- ...

Ses symboles électroniques sont les suivants :

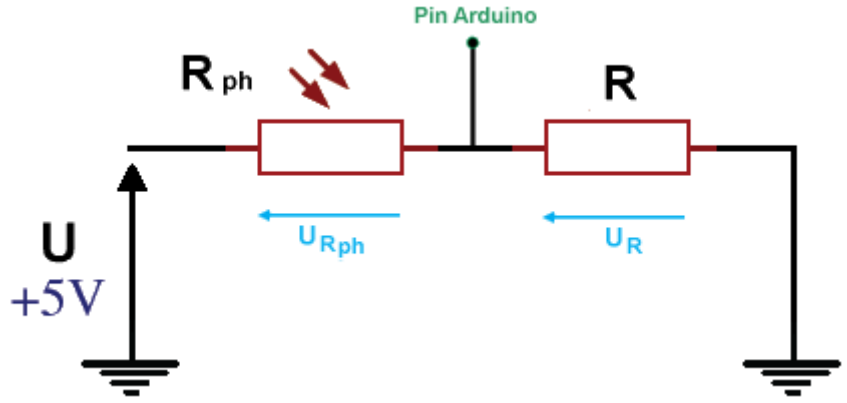


L'avantage des photorésistances est qu'elles sont très bon marché. Par contre, leur réaction à la lumière est différente pour chaque photorésistance, même quand elles ont été produites dans le même lot. On peut ainsi noter une différence de résistance de plus de 50% entre deux photorésistances du même modèle pour la même luminosité. On ne peut donc pas l'utiliser pour une mesure précise de la lumière. Par contre, elles sont idéales pour mesurer des changements simples de la luminosité.

On utilise la photorésistance dans un montage (Cf. ci-dessous), avec une résistance fixe de 10 kΩ, qu'on appelle un **diviseur de tension**.

La photorésistance est alimentée en 5V depuis l'Arduino. Le point entre les deux résistances est relié à une broche analogique de l'Arduino et on mesure la tension de cette broche par la fonction "**analogRead (broche)**".

Tout changement de la tension mesurée est dû à la photorésistance puisque c'est la seule résistance qui change dans ce circuit, en fonction de l'intensité lumineuse qu'elle reçoit.



D'après la loi d'additivité des tensions dans un circuit en série :

$$U = U_{Rph} + U_R = (R_{ph} + R)\, I$$

$$U_R = U - U_{Rph} = U - R_{ph}\, I$$

U_R est la tension appliquée sur l'entrée analogique de l'Arduino. Quand l'intensité lumineuse reçue par la photorésistance augmente, R_{ph} diminue, donc U_R augmente, et au contraire quand la luminosité diminue, R_{ph} augmente et U_R diminue.

On peut exprimer Ur en fonction de U :

$$U = U_{Rph} + U_R = (R_{ph} + R)\, I$$

Donc : $$I = \frac{U}{(R_{ph} + R)}$$

Et : $$U_R = R\, I = \frac{R}{(R_{ph} + R)}\, U$$

C'est la raison pour laquelle on appelle ce montage un diviseur de tension.

. Le programme

Le code (**"Activity2.ino" dans le dossier "Codes/Ondes Sonores"**) pourra être modifié pour voir l'influence des variables (sensibilité du capteur, fréquence de l'onde sonore, durée d'émission, durée de silence).

Activity2

```
// Déclaration des constantes et variables

const int PinLED = 9;
const int PinTone = 8;
const int PinPhotoR = 5;
const int CapteurSensib = 500;
const int TimeSleep = 200;

int ValCapteur = 0;


// Initialisation des entrées et sorties

void setup() {
  pinMode (PinLED, OUTPUT);
  digitalWrite(PinLED, HIGH);
}


// Fonction principale en boucle

void loop() {
  ValCapteur = analogRead(PinPhotoR);
  if (ValCapteur < CapteurSensib) {
    tone(PinTone, 440);
    delay(TimeSleep);
    noTone(PinTone);
    delay(TimeSleep);
  }
  else {
  noTone(PinTone);
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

. **const int PinLed = 9** (constante nombre entier correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. **const int PinTone = 8** (constante nombre entier correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **const int PinPhotoR = 5** (constante nombre entier correspondant au n° de la broche sur laquelle la photorésistance est connectée)

. **const int CapteurSensib = 500** (constante nombre entier correspondant à la sensibilité du capteur)

. **const int TimeSleep = 200** (constante nombre entier correspondant à la durée d'émission sonore et de silence)

. **int ValCapteur = 0** (variable nombre entier pour stocker la valeur du capteur)

- Initialisation des entrées et sorties :

void setup()

initialisation des entrées et sorties

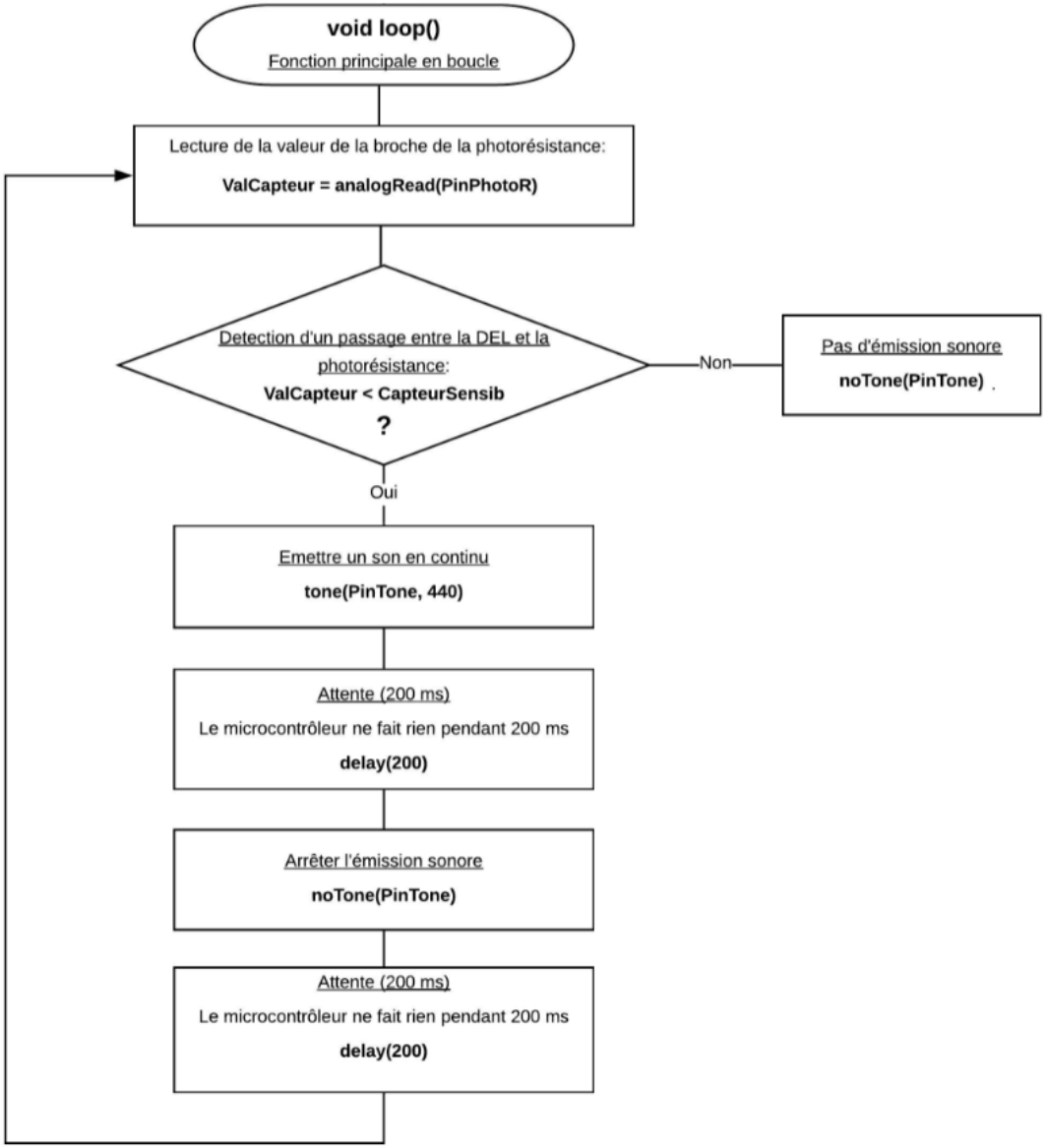
- La broche de la DEL est initialisée comme une sortie digitale. Des données seront donc envoyées depuis le microcontroleur vers cette broche :

pinMode (PinLED, OUTPUT)

- La LED est allumée :

digitalWrite(PinLED, HIGH)

- Fonction principale en boucle :



Activité 3 : Jouer une mélodie avec un Arduino

Dans cette activité, nous allons voir qu'il est possible de jouer une mélodie avec un Arduino.

Avec le même circuit que précédemment, la mélodie sera jouée en continu après un appui sur le bouton poussoir et arrêtée en appuyant de nouveau sur celui-ci.

. Les différentes notes de musique

Chaque note est caractérisée par une fréquence fondamentale déterminée. Lorsque deux notes sont séparées d'une octave, le rapport de leur fréquence est égal à deux.

Dans la pratique, ces deux notes ont le même nom, comme par exemple le "**la**" de l'octave **3** et le "**la**" de l'octave **4**, nommée couramment "**la3**" et "**la4**" pour éviter toute ambigüité entre elles.

On appelle gamme l'ensemble des notes (de Do à Si) composant une octave donnée. Dans la gamme tempérée, c'est-à-dire celle utilisée dans la musique occidentale, l'octave est divisée en 12 demi-tons, ce qui correspond à 12 notes, en comptant les notes diésées (#).

Par exemple, pour l'octave 1, voici les valeurs de fréquence des 12 notes :

Note	Fréquence (Hz)
Do	65
Do# ou Réb	69
Ré	74
Ré# ou Mib	78
Mi	83
Fa	87
Fa# ou Solb	93
Sol	98
Sol# ou Lab	104
La	110
La# ou Sib	117
Si	123

Le # signifie dièse et le b signifie bémol. Ce sont des altérations des notes de la gamme de base (Do, Ré, Mi, Fa, Sol, La, Si). Le dièse augmente la fréquence de la note et le bémol la diminue. Ainsi un La # est situé en fréquence entre le La et le Si

• En notation latine (la notation historique : do ré mi …)

On a l'habitude d'écrire que le diapason (la note qui nous sert de référence à 440 Hz) est le "**la3**". Dans ce système, on peut se donner en repère que la première octave entièrement audible commence par le do0.

• En notation américaine (la notation scientifique : A B C …)

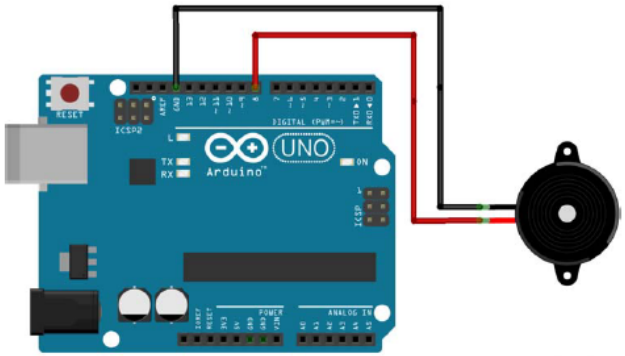
Dans cette notation, le "**la**" du diapason est le **A4**. C'est le système qui est utilisé en langage Arduino. Dans ce système, on a choisi de dire que le C (do) le plus grave d'un clavier de piano à 88 touches est le C1. Suivant ce repère, on a alors **A4 = 440 Hz**.

Voici un tableau qui référence la fréquence des notes de musique en hertz de la gamme tempérée (notation américaine en noir et notation latine en rouge) :

Note \octave	**0 (-1)**	**1 (0)**	**2 (1)**	**3 (2)**	**4 (3)**	**5 (4)**	**6 (5)**	**7 (6)**	**8 (7)**
C (Do)	16,35	32,70	65,41	130,81	261,63	523,25	1046,50	2093,00	4186,01
C♯ – Db (Do♯)	17,32	34,65	69,30	138,59	277,18	554,37	1108,73	2217,46	4434,92
D (Ré)	18,35	36,71	73,42	146,83	293,66	587,33	1174,66	2349,32	4698,64
D♯ – Eb (Ré♯)	19,45	38,89	77,78	155,56	311,13	622,25	1244,51	2489,02	4978,03
E (Mi)	20,60	41,20	82,41	164,81	329,63	659,26	1318,51	2637,02	5274,04

Note \octave	0 (-1)	1 (0)	2 (1)	3 (2)	4 (3)	5 (4)	6 (5)	7 (6)	8 (7)
F (Fa)	21,83	43,65	87,31	174,61	349,23	698,46	1396,91	2793,83	5587,65
F♯ – Gb (Fa♯)	23,12	46,25	92,50	185,00	369,99	739,99	1479,98	2959,96	5919,91
G (Sol)	24,50	49,00	98,00	196,00	392,00	783,99	1567,98	3135,96	6271,93
G♯ – Ab (Sol♯)	25,96	51,91	103,83	207,65	415,30	830,61	1661,22	3322,44	6644,88
A (La)	27,50	55,00	110,00	220,00	440,00	880,00	1760,00	3520,00	7040,00
A♯ – Bb (La♯)	29,14	58,27	116,54	233,08	466,16	932,33	1864,66	3729,31	7458,62
B (Si)	30,87	61,74	123,47	246,94	493,88	987,77	1975,53	3951,07	7902,13

. Jouer une mélodie avec un Arduino



On utilisera la fonction **tone()** pour jouer les notes de musique (voir tableau des fréquences des notes) de la mélodie pendant la durée définie pour chaque note.

La durée des notes est généralement calculée en attribuant une seconde (1000 ms) à une ronde. Une blanche représente alors 500 ms (ronde/2), une noire, 250 ms (blanche/2, ou ronde /4), une croche,125 ms (noire/2 ou ronde/8), etc…

Pour bien distinguer les notes jouées par l'Arduino, il faut respecter un temps de pause entre chaque note, égal à la durée de la note + 30 % :

Temps de pause (en ms) = Durée de la note (en ms) x 1,3

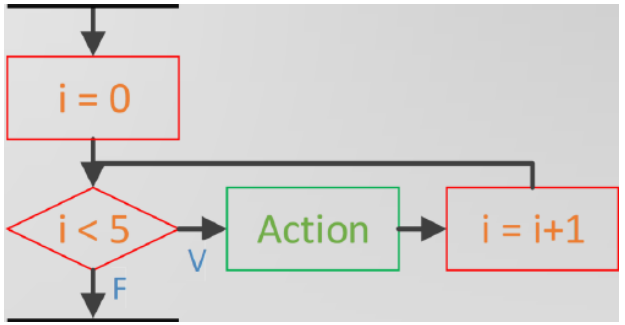
Pour cela, le plus simple est de créer une liste des fréquences (Freq) des notes à jouer et une liste des durées (Dur) puis de demander à l'Arduino de jouer chaque élément, i, des listes à l'aide d'une boucle for :

```
for(int i = 0 ; i < 11 ; i++)
  {
  tone (broche du piezo, Freq[i], Dur[i])
  delay(Dur[i] x 1.3)
  }
```

. Rappel sur la programmation d'une boucle **for**:

Boucle FOR (Pour)

```
for(int i=0; i<5; i=i+1)
{
   // Action;
}
```



. Le programme

Le code (**"Activity3.ino" dans le dossier "Codes/Ondes Sonores"**) pourra être modifié pour voir l'influence des variables (fréquence des notes, durée des notes, durée des silences entre les notes).

Activity3

```
// Déclaration des constantes et variables

const int PinTone = 8;
const int PinButton = 12;
const int Notes[] = {262,294,330,262,294,330,349,392,330,349,392};
const int NoteDurations[] = {4,4,4,4,4,4,4,8,4,4,8};

int State = 0;
int ValButton = 0;
int OldValButton = 0;


// Initialisation des entrées et sorties

void setup() {
  pinMode (PinButton, INPUT);
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);
  if (ValButton == HIGH and OldValButton == LOW) {
    State = 1 - State;
  }
  if (State == 1) {
    for(int i = 0 ; i < 11 ; i++)
    {
      int NoteDuration = int(1000/NoteDurations[i]);
      tone(PinTone, Notes[i], NoteDuration);
      delay(1.3*NoteDuration);

      ValButton = digitalRead(PinButton);
      delay(10);
      if (ValButton == HIGH and OldValButton == LOW) {
      State = 1 - State;
      break;
      }
      OldValButton = ValButton;
    }
  }
  else {
    noTone(PinTone);
  }
  OldValButton = ValButton;

}
```

Déroulement du programme :

- Déclaration des constantes et variables :

  . **const int PinButton = 12** (constante nombre entier correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

  . **const int PinTone = 8** (constante nombre entier correspondant au n° de la broche sur laquelle le buzzer est connecté)

  . **const int Notes[]** (liste de constantes nombres entiers correspondant aux fréquences des notes à jouer)

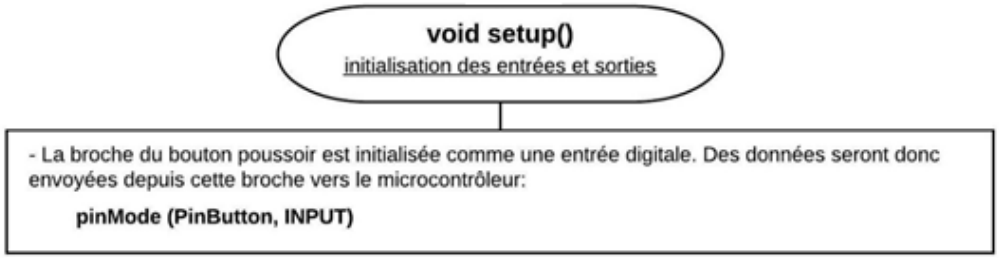
  . **const int NoteDurations[]** (liste de constantes nombre entier correspondant aux durées des notes en fraction de seconde)

  . **int State = 0** (variable nombre entier correspondant à l'action à effectuer)

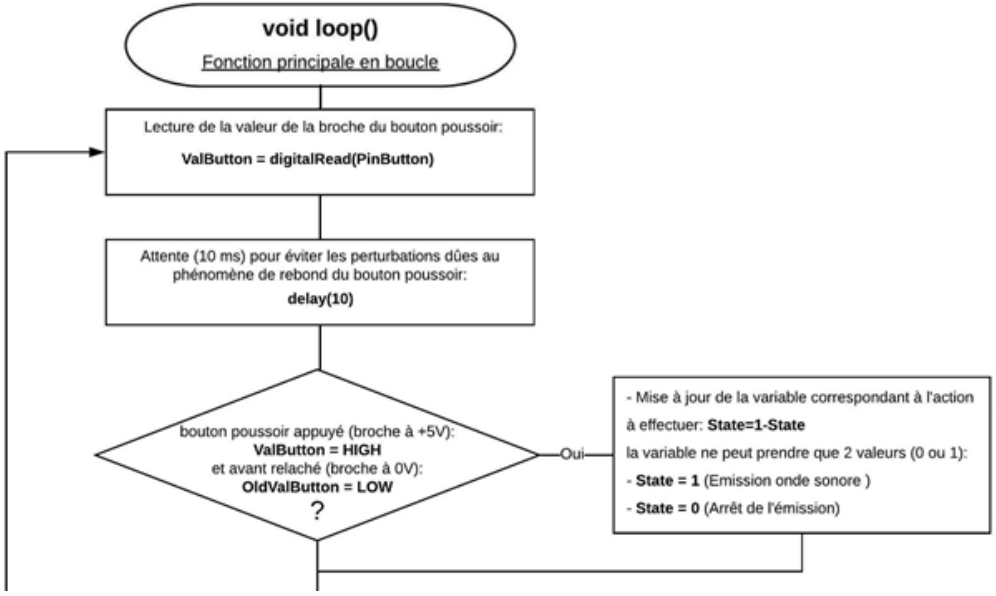
. **int ValButton = 0** (variable nombre entier pour stocker la valeur de la broche du bouton poussoir)

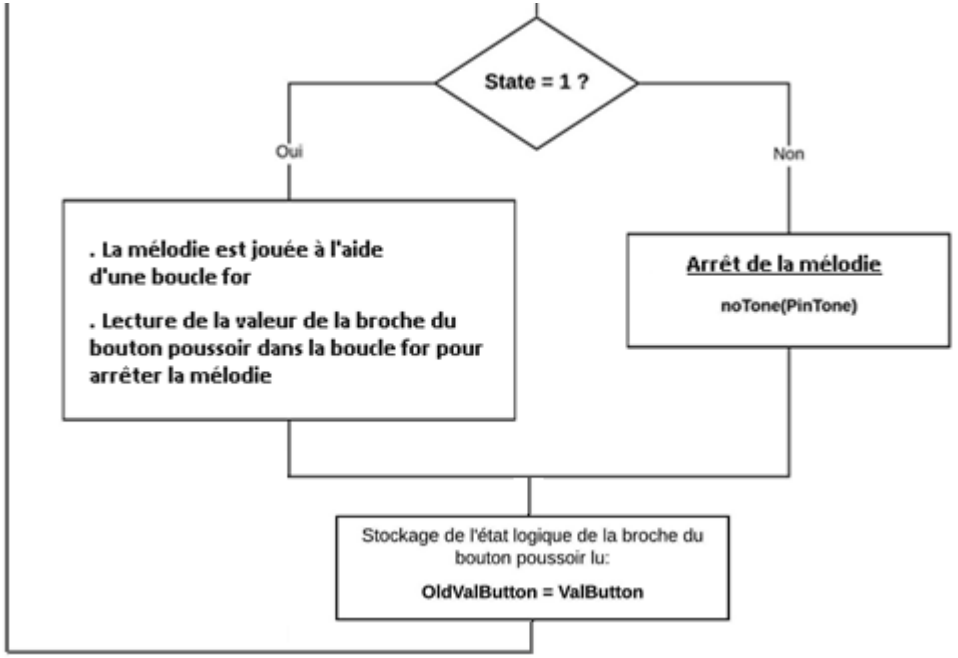
. **int OldValButton = 0** (variable nombre entier pour stocker la valeur précédente de la broche du bouton poussoir)

- Initialisation des entrées et sorties :



- Fonction principale en boucle :





Activité 4 : Régler la fréquence d'une onde sonore avec un potentiomètre

Toujours avec le même circuit, dans cette activité, l'appui sur le bouton-poussoir produit une onde sonore dont la fréquence est réglée à l'aide du potentiomètre. L'émission sonore est arrêtée en appuyant de nouveau sur le bouton poussoir.

Le potentiomètre est connecté sur la broche **A0** de l'Arduino. La tension de cette broche varie donc entre **0** et **5** V (voir le principe de fonctionnement du potentiomètre) en fonction de la position du curseur du potentiomètre.

La lecture de la valeur de la broche **A0** convertie par le convertisseur analogique numérique de l'Arduino donne donc un nombre entier entre **0** et **1023** qui sera utilisé pour régler la fréquence du son émis, en le multipliant par un coefficient préalablement choisi par l'intermédiaire du moniteur série.

. Envoi de données du moniteur série vers l'Arduino

Nous avons vu, avec le programme d'apprentissage des entrées analogiques, que l'on pouvait envoyer des données de l'Arduino vers le moniteur série avec la fonction **"Serial.print()"**.

Il est également possible d'envoyer des données du moniteur série vers l'Arduino, en réponse à une demande d'information par le programme téléversé dans sa mémoire.

Pour cela, on va utiliser la fonction **"Serial.available()"** qui donne le nombre de caractères (octets) disponible pour lecture dans la file d'attente (buffer) du port série et la fonction **"Serial.read()"** qui lit les données contenues dans la mémoire tampon (buffer) du port série.

Les données seront envoyées sous la forme d'une chaine de caractères par le moniteur série. L'Arduino va lire la chaine, caractère par caractère, et la reconstituer.

Programme pour la réception d'une chaine de caractères :

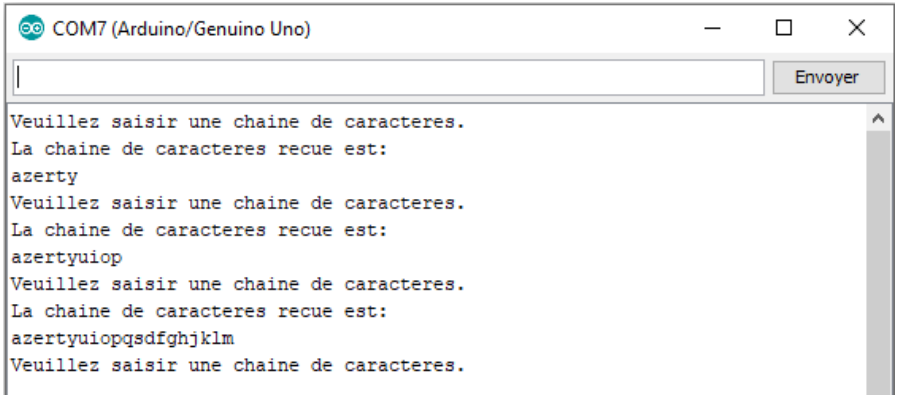
```
void setup() {
Serial.begin(9600);
}

void loop() {
 int Val=0;
 char tampon[20]="";
 Serial.println("Veuillez saisir une chaine de caracteres.");
 while(Val==0)
  {
  delay(200);
  Val=Serial.available();
  }
 for (int i=0; i < Val; i++)
  {
```

```
  tampon[i]=Serial.read();
  delay(15);
  }
 Serial.println("La chaine de caracteres recue est:");
 Serial.println(tampon);
 Serial.flush();
}
```

Dans ce programme, il est demandé de saisir une chaine de caractères. Tant que le nombre de caractères disponibles sur le port série est nul (**Val==0**), la mémoire tampon du port série est vérifiée (**Val=Serial.available()**), toutes les 200 ms, jusqu'à ce que tous les caractères de la chaine soient comptabilisés. Ceux-ci sont alors lu un par un à travers une boucle "**for**" (**tampon[i]=Serial.read()**) et sont stockés dans un tableau de 20 caractères au maximum appelé "**tampon**" qui est ensuite affiché dans le moniteur série. Puis la mémoire du port série est vidée et il est de nouveau demander de saisir une chaine de caractères.

Voici le résultat dans le moniteur série :



Remarque :

Il est souvent nécessaire de convertir des chaines de caractères en une variable représentant un nombre entier ou décimal pour effectuer des calculs.

Il existe, pour cela, trois fonctions "**atoi()**", "**atol()**" et "**atof()**", permettant respectivement de transformer une chaine de caractères en nombre entier court, en nombre entier long et en nombre décimal.

Si notre chaine de caractères "**tampon**" représente un nombre, pour la convertir par exemple en nombre entier, on utilisera l'instruction :

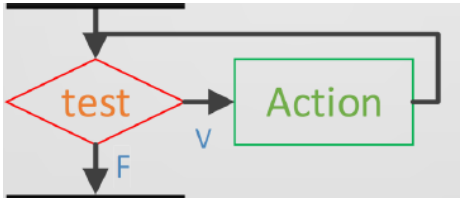
Int nombre = atoi(tampon)

Ces fonctions retournent une valeur nulle en cas d'erreur (comme par exemple, si on essaye de convertir des lettres en nombre).

. Rappel sur la programmation d'une boucle **while**:

Boucle WHILE (Tant … que)

```
while(test)
{
   // Action;
}
```



. Le programme

Voici le code de l'activité (**"Activity4.ino" dans le dossier "Codes/Ondes Sonores"**) :

Activity4

```
// Déclaration des constantes et variables

const int PinPOT = 0;
const int PinTone = 8;
const int PinButton = 12;

int State = 0;
int ValButton = 0;
int OldValButton = 0;
int ValPot = 0;
int Coeff=0;


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode (PinButton, INPUT);
  while(Coeff<1 || Coeff>4)
  {
  int Val=0;
  char tampon[10]="";
  Serial.println("Veuillez choisir la gamme de frequence (valeur entre 1 et 4):")
  Serial.println("1 : 0 - 1023 Hz :");
  Serial.println("2 : 0 - 2046 Hz :");
  Serial.println("3 : 0 - 3069 Hz :");
  Serial.println("4 : 0 - 4092 Hz :");
  Serial.println(" ");
  while(!Val)
    {
    delay(200);
    Val=Serial.available();
    }
  for (int i=0; i < Val; i++)
    {
    tampon[i]=Serial.read();
    delay(15);
    }
  Coeff = atoi(tampon);
  }
  Serial.print("Gamme de frequence choisie : "); Serial.println(Coeff);
}


// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);
  if (ValButton == HIGH and OldValButton == LOW) {
    State = 1 - State;
  }
  OldValButton = ValButton;
  if (State == 1) {
    ValPot = analogRead(PinPOT);
    tone(PinTone, ValPot*Coeff);
  }
  else {
    noTone(PinTone);
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

. **const int PinButton = 12** (constante nombre entier correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **const int PinTone = 8** (constante nombre entier correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **const int PinPot = 0** (constante nombre entier correspondant au n° de la broche sur laquelle le potentiomètre est connecté)

. **int State = 0** (variable nombre entier correspondant à l'action à effectuer)

. **int ValButton = 0** (variable nombre entier pour stocker la valeur de la broche du bouton poussoir)

. **int OldValButton = 0** (variable nombre entier pour stocker la valeur précédente de la broche du bouton poussoir)

. **int ValPot = 0** (variable nombre entier pour stocker la valeur de la broche du potentiomètre)

. **int Coeff = 0** (variable nombre entier pour stocker la valeur du coefficient multiplicateur)

- Initialisation des entrées et sorties :

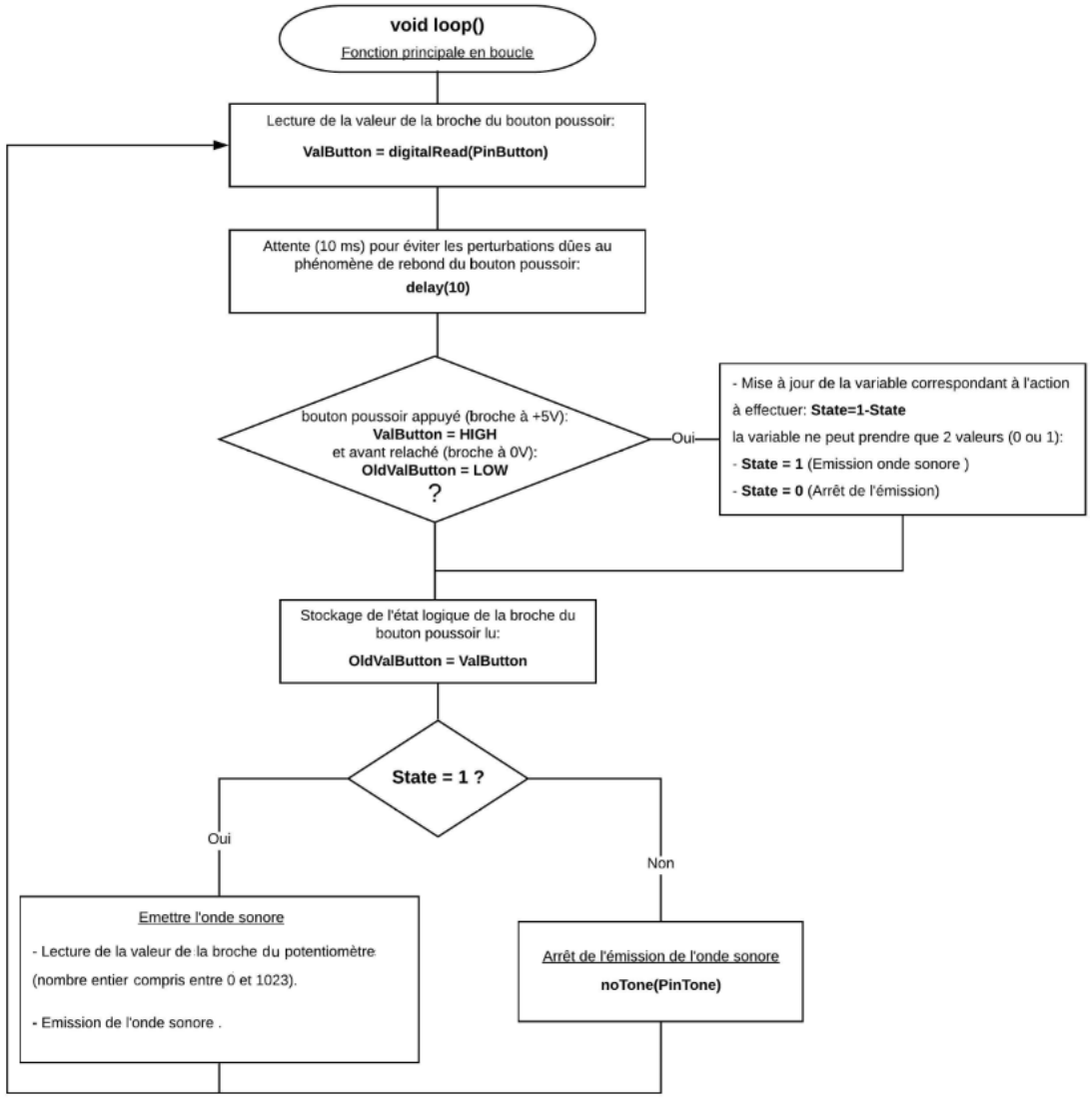
. La liaison série est initialisée à 9600 bauds,

. La broche du bouton poussoir est initialisée comme une entrée digitale,

. Saisie du coefficient multiplicateur (nombre entier entre 1 et 4) dans le moniteur série,

. Affichage dans le moniteur série du coefficient choisi.

- Fonction principale en boucle :

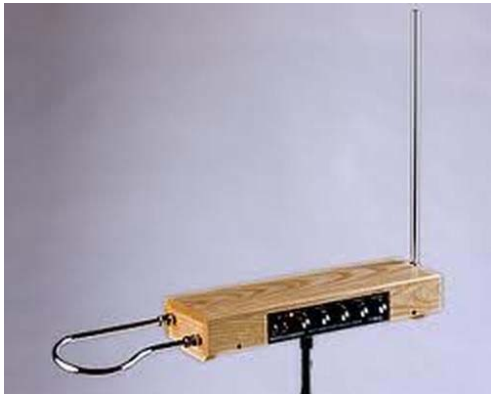


Activité 5 : Régler la fréquence d'une onde sonore avec une photorésistance

L'objectif de cette activité est de réaliser un pseudo-thérémine.

Le thérémine est un des plus anciens instruments de musique électronique, inventé en 1920 par le Russe Lev Sergueïevitch Termen (connu sous le nom de « Léon Theremin »).

Composé d'un boîtier électronique équipé de deux antennes, l'instrument a la particularité de produire de la musique sans être touché par l'instrumentiste. Dans sa version la plus répandue, la main droite commande la hauteur de la note (la fréquence du son), en faisant varier sa distance à l'antenne verticale. L'antenne horizontale, en forme de boucle, est utilisée pour faire varier le volume selon sa distance à la main gauche.



Dans cette activité, on ne simulera que la commande de la hauteur de la note, à l'aide de la photorésistance de notre montage d'étude des ondes sonores.

La sortie de la photorésistance est connectée à l'entrée analogique **A5** de l'Arduino. La valeur de cette broche (nombre entier entre 0 et 1023), lue par la fonction **"analogRead()"**, est alors proportionnelle à l'intensité lumineuse reçue par la photorésistance.

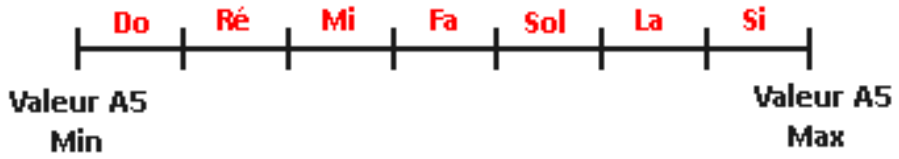
En approchant la main de la photorésistance, la tension mesurée au niveau de la broche A5 diminue et inversement en reculant la main la tension mesurée augmente.

Ainsi en fonction de la position de la main par rapport à la photorésistance, on va pouvoir jouer la gamme de notes de l'octave 3 sans les notes diésées (#) :

Notes	Do	Ré	Mi	Fa	Sol	La	Si
Fréquence (Hz)	262	294	330	350	392	440	494

En premier, il faut étalonner la photorésistance de façon à connaitre les valeurs minimale et maximale de la broche A5 en fonction des conditions d'éclairement de la photorésistance.

Puis on attribue pour chaque note de musique une plage de valeurs de tension réparties entre les valeurs minimale et maximale :



Chaque plage de tension représente : $\frac{\text{Max}-\text{Min}}{7}$

On crée alors une liste de notes à jouer et une liste de fréquences associée aux notes :

String Notes[]={"Do","Re","Mi","Fa","Sol","La","Si"};

const int FreqNotes[] = {262,294,330,350,392,440,494};

Alors :

Le "Do" est la note "0" : **Notes[0]** (fréquence : **FreqNotes[0]**),

Le "Ré" est la note "1" : **Notes[1]** (fréquence : **FreqNotes[1]**),

…

Et la plage de tension pour chaque note "i" :

$$\text{Min} + i \times \left(\frac{\text{Max}-\text{Min}}{7}\right) < \textbf{Valeur A5} < \text{Min} + (i+1) \times \left(\frac{\text{Max}-\text{Min}}{7}\right)$$

Il suffira ensuite de lire la valeur de la broche A5, de tester dans quelle plage de mesure, elle se trouve et de jouer la note associée.

. Le programme

Voici le code de l'activité (**"Activity5.ino" dans le dossier "Codes/Ondes Sonores"**) :

Activity5

```
// Déclaration des constantes et variables

const int PinPhotoR = 5;
const int PinTone = 8;
const int PinButton = 12;

int State = 0;
int ValButton = 0;
int OldValButton = 0;
int ValCapteur = 0;
int MinPhotoR = 0;
int MaxPhotoR = 0;
String Notes[]={"Do","Re","Mi","Fa","Sol","La","Si"};
const int FreqNotes[] = {262,294,330,350,392,440,494};


// Définition de fonctions

void Etalonnage_PhotoR() {
  char c=' ';
  Serial.println("Etalonnage de la photoresistance:");
  Serial.println("");
  while(c!='C')
  {
    int Val=0;
    Serial.println("Veuillez cacher la photoresistance et saisir 'C' pour continuer.");
    while(!Val)
      {
      delay(200);
      Val=Serial.available();
      ValCapteur = analogRead(PinPhotoR);
      }
    c = Serial.read();
  }
  MinPhotoR=ValCapteur;
  Serial.print("Valeur minimale de la broche de la photoresistance: ");
  Serial.println(MinPhotoR);
  Serial.println("");
```

```
  c=' ';
  while(c!='C')
    {
    int Val=0;
    Serial.print("Veuillez exposer la photoresistance a la lumiere ");
    Serial.println("et saisir 'C' pour continuer.");
    while(!Val)
      {
      delay(200);
      Val=Serial.available();
      ValCapteur = analogRead(PinPhotoR);
      }
    c = Serial.read();
    }
  MaxPhotoR=ValCapteur;
  Serial.print("Valeur maximale de la broche de la photoresistance: ");
  Serial.println(MaxPhotoR);
  Serial.println("");
}


// Initialisation des entrées et sorties

void setup() {
  Serial.begin(9600);
  pinMode (PinButton, INPUT);
  Etalonnage_PhotoR();
  boolean Etalonnage=true;
  while (Etalonnage == true)
  {
    char c=' ';
    while(c!='O' and c!='N')
      {
      Serial.println("Voulez vous recommencer l'etalonnage (Saisir 'O' ou 'N')?");
      Serial.println("");
      int Val=0;
      while(!Val)
        {
        delay(200);
        Val=Serial.available();
        }
      c = Serial.read();
      }
    if (c=='O') {
      Etalonnage_PhotoR();
    }
    else{
      Etalonnage=false;
      Serial.println("Appuyez sur le bouton poussoir pour jouer des notes de musique...");
    }
  }
}

// Fonction principale en boucle

void loop() {
  ValButton = digitalRead(PinButton);
  delay(10);
  if (ValButton == HIGH and OldValButton == LOW) {
    State = 1 - State;
  }
  OldValButton = ValButton;
  if (State == 1) {
    ValCapteur = analogRead(PinPhotoR);
    for(int i = 0 ; i < 7 ; i++) {
      if (ValCapteur> MinPhotoR+ i*(MaxPhotoR-MinPhotoR)/7 and
      ValCapteur< MinPhotoR+ (i+1)*(MaxPhotoR-MinPhotoR)/7){
        Serial.println(Notes[i]);
        tone(PinTone,FreqNotes[i]);
        break;
      }
    }
  }
  else {
    noTone(PinTone);
  }
}
```

Déroulement du programme :

- Déclaration des constantes et variables :

  . **const int PinButton = 12** (constante nombre entier correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

  . **const int PinTone = 8** (constante nombre entier correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **const int PinPhotoR = 5** (constante nombre entier correspondant au n° de la broche sur laquelle la photorésistance est connectée)

. **int State = 0** (variable nombre entier correspondant à l'action à effectuer)

. **int ValButton = 0** (variable nombre entier pour stocker la valeur de la broche du bouton poussoir)

. **int OldValButton = 0** (variable nombre entier pour stocker la valeur précédente de la broche du bouton poussoir)

. **int ValCapteur = 0** (variable nombre entier pour stocker la valeur de la broche de la photorésistance)

. **int MinPhotoR = 0** (variable nombre entier pour stocker la valeur minimale de la broche de la photorésistance)

. **int MaxPhotoR = 0** (variable nombre entier pour stocker la valeur maximale de la broche de la photorésistance)

. **String Notes[]={"Do","Re","Mi","Fa","Sol","La","Si"}**
const int FreqNotes[] = {262,294,330,350,392,440,494}
(Listes des notes et leurs fréquences)

- Définition de fonctions :

void Etalonnage_PhotoR() : fonction permettant l'étalonnage de la photorésistance

→ Utilisation du moniteur série pour demander à l'utilisateur de cacher la photorésistance,
→ lecture de la valeur de la broche A5,
→ l'utilisateur doit envoyer le caractère 'C' à l'aide du moniteur série pour continuer l'étalonnage,
→ Affichage de la valeur minimale mesurée de la broche A5,
→ L'utilisateur doit exposer la photorésistance à la lumière ambiante,
→ lecture de la valeur de la broche A5,
→ l'utilisateur doit envoyer le caractère 'C' pour quitter l'étalonnage,
→ Affichage de la valeur maximale mesurée de la broche A5.

- Initialisation des entrées et sorties :

. La liaison série est initialisée à 9600 bauds,

. La broche du bouton poussoir est initialisée comme une entrée digitale,

. Appel de la fonction d'étalonnage,

. Possibilité de recommencer l'étalonnage.

- Fonction principale en boucle :