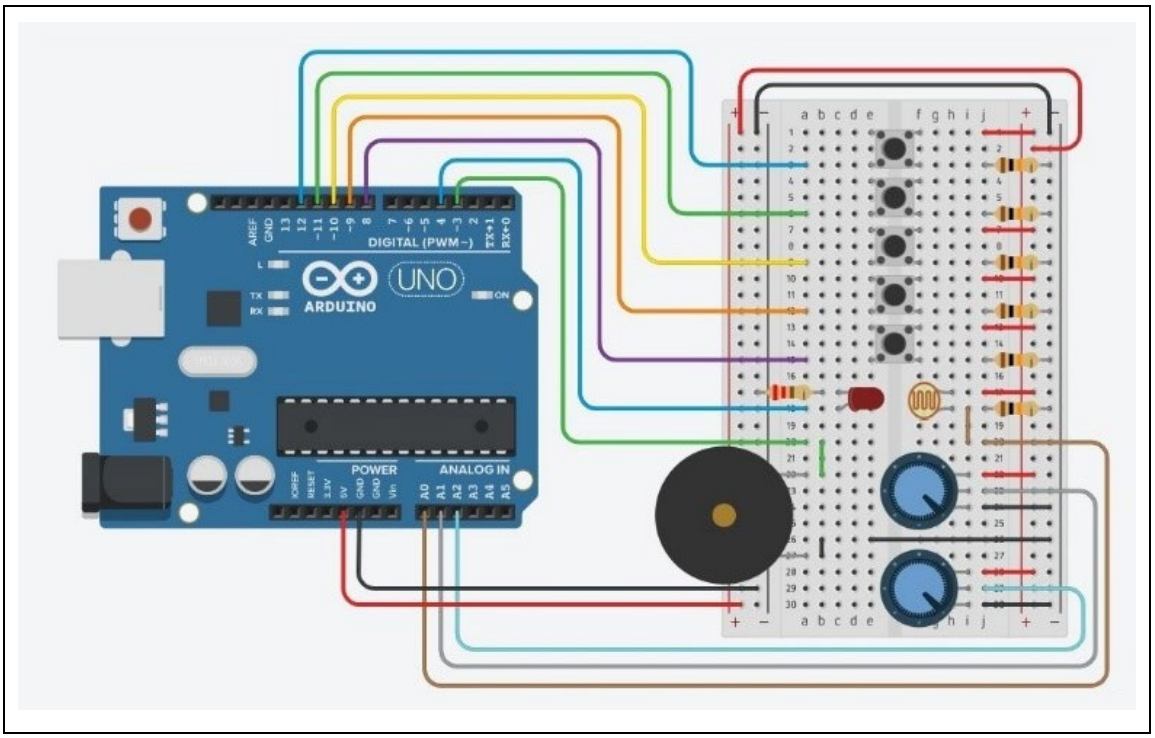
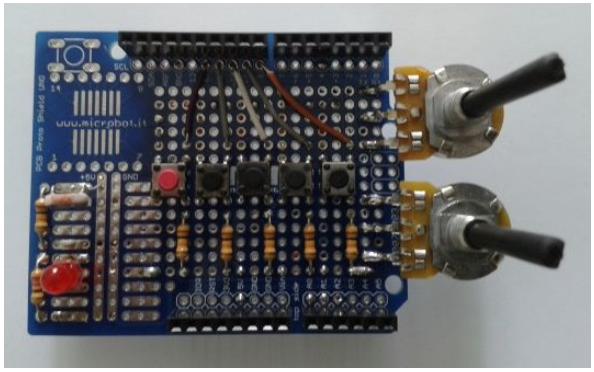
Projet 3 - Ondes sonores : Produire & Exploiter

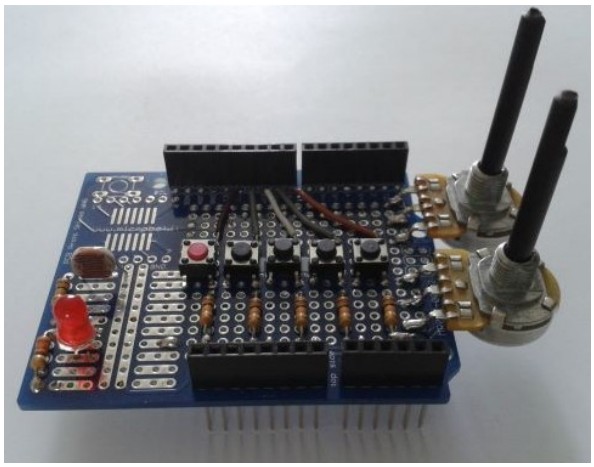
Après avoir bien travaillé, maintenant que nous maitrisons le principe des entrées et sorties de l'Arduino, nous allons nous détendre en écoutant un peu de musique…



- Liste des composants :

. 1 DEL Rouge
. 1 résistance de 220 Ω
. 6 résistances de 10 kΩ
. 2 potentiomètres de 10 kΩ
. 1 photorésistance
. 5 boutons poussoir
. 1 haut-parleur (ou buzzer)
. 1 plaque d'essai
. Fils de connexion

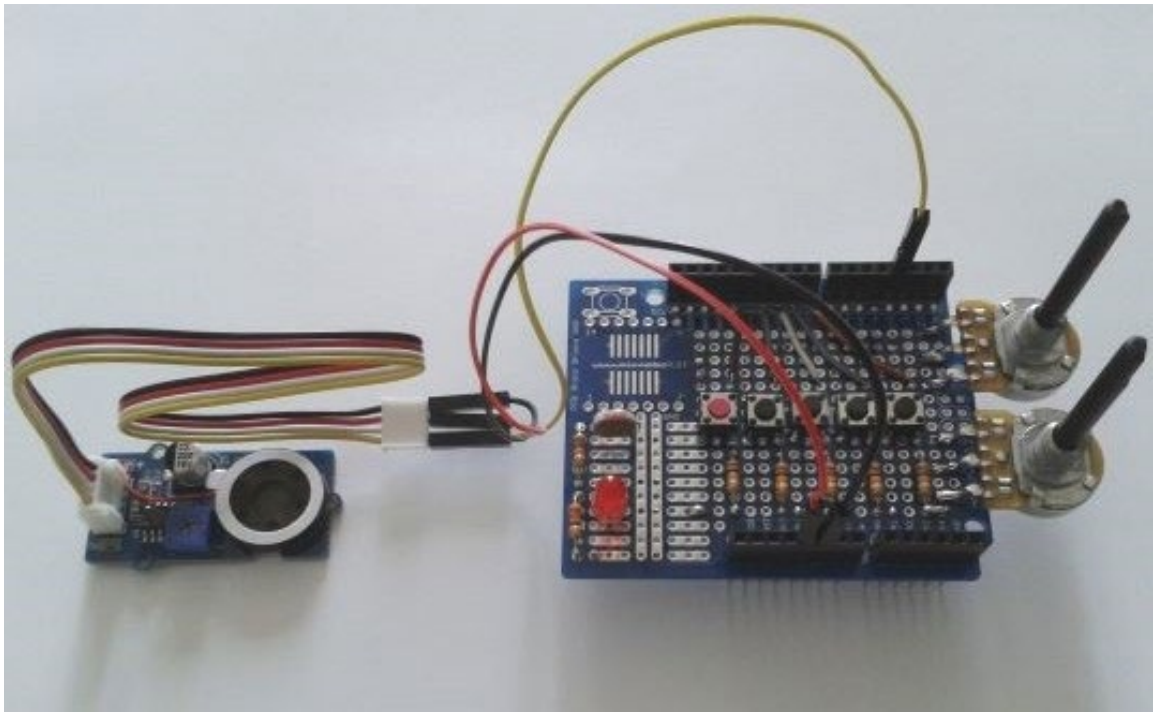




Le circuit sur un "shield" pour Arduino Uno

- Protocole de communication:

. Firmata Express

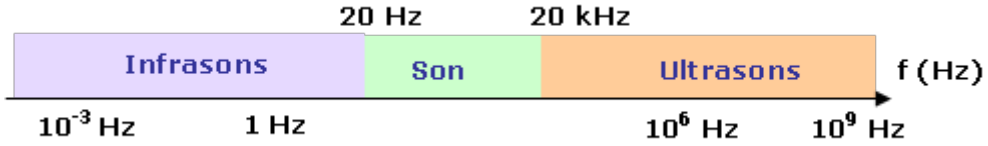


Rappels :

Une onde sonore ou acoustique est une perturbation mécanique périodique qui se propage dans un milieu matériel fluide élastique (l'eau ou l'air par exemple), en s'éloignant de sa source. C'est une onde de compression-dilatation du milieu dans lequel elle se propage.

Par exemple, Une corde de guitare qui vibre, va transmettre sa vibration à l'air. Cette vibration va se propager dans l'air jusqu'à nos oreilles qui vont recevoir la vibration.

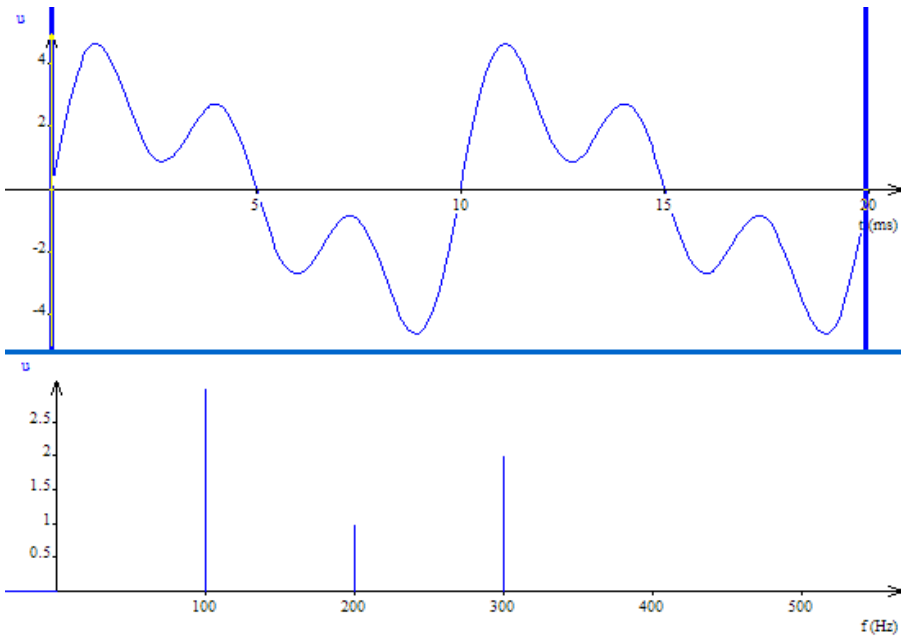
L'être humain peut entendre des sons dont les fréquences s'étalent de 20Hz à 20kHz environ.



Un son pur est un signal sonore sinusoïdale de fréquence **f** (Hz) qui ne varie pas au cours du temps. Le signal sonore délivré par un diapason est un son pur.

La plupart des sons que nous percevons dans notre environnement ne sont pas purs mais complexes. Ils sont composés de plusieurs sons purs de fréquences et d'amplitudes différentes. Un son musical est un cas particulier de son complexe, produit par un instrument de musique. Un son musical est périodique (de période constante au cours du temps), mais n'est pas forcément sinusoïdal. C'est en fait, une superposition de plusieurs signaux sinusoïdaux.

II est possible de décomposer un signal sonore u(t) de fréquence f associé à la propagation d'une onde sonore périodique non sinusoïdale, en une somme infinie de signaux sinusoïdaux, c'est la décomposition de Fourier du signal :



Le signal ci-dessus de fréquence f=100 Hz (T=10 ms) se décompose de la façon suivante :

$$u(t)= 3\times\sin(2\times\pi\times100\times t) +\sin(2\times\pi\times200\times t) + 2\times\sin(2\times\pi\times300\times t)$$

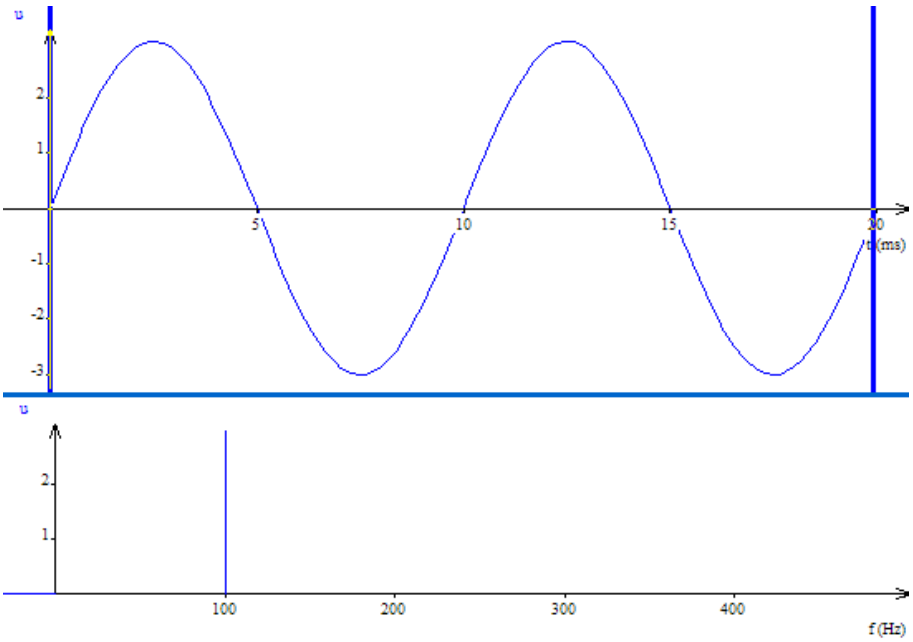
Un signal sonore complexe périodique de fréquence f est donc une superposition de signaux sinusoïdaux :

- un signal sinusoïdal à la fréquence f nommée "fondamental" ou "première harmonique",
- un signal sinusoïdal à la fréquence 2f, la "deuxième harmonique",
- un signal sinusoïdal à la fréquence 3f, la "troisième harmonique", etc…

La représentation de l'amplitude des harmoniques en fonction de la fréquence constitue le spectre du signal (voir image ci-dessus).

Les harmoniques sont des signaux sinusoïdaux de fréquences $f_n=n\times f$. Le nombre ***n*** est un entier positif appelé rang de l'harmonique.

Dans le cas d'un son pur, le spectre du signal sonore ne présente qu'une unique harmonique, le fondamental.



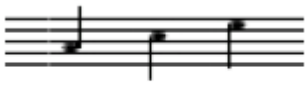
Spectre d'un son pur de fréquence f=100 Hz

Un son musical est caractérisé par sa hauteur et son timbre :

. **<u>Hauteur d'un son musical</u>**

La hauteur d'un son musical est la fréquence **f**, exprimée en hertz (symbole : Hz), de l'onde sonore périodique. C'est la fréquence du fondamental dans la décomposition de Fourier de cette onde.

La hauteur d'un son musical est associée au nom de la note jouée (do, ré, mi, ... d'une certaine octave). Le terme de hauteur vient du fait que les notes sont écrites sur une portée musicale de telle manière que la position verticale de la note sur la portée corresponde à son nom.



La portée musicale

Si la fréquence est multipliée par deux, on passe à l'octave supérieure. À l'inverse si la fréquence est divisée par deux, on passe à l'octave inférieure.

Une note de musique correspond à une fréquence d'un son à toutes les octaves.

La note "**la**" correspond à la fréquence f=440 Hz, mais aussi à 880 Hz (octave supérieure), 220 Hz (octave inférieure), etc.

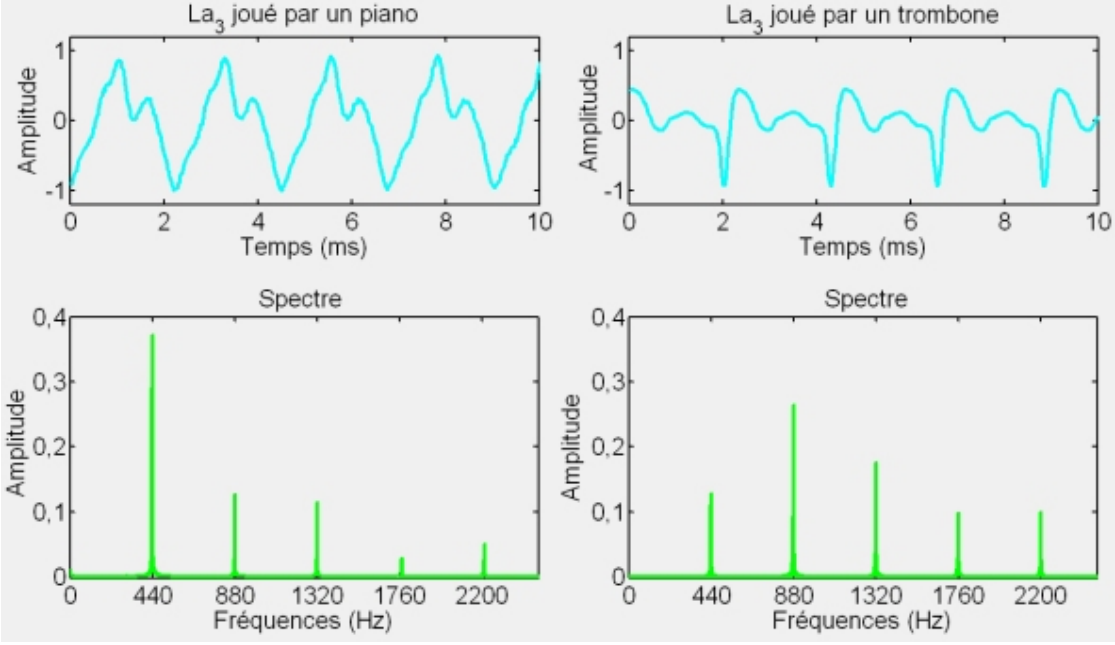
Une onde sonore est d'autant plus aigüe que sa fréquence est grande. Elle est d'autant plus grave que sa fréquence est petite.

. Timbre d'un son musical

Une note de hauteur donnée n'est pas perçue de la même manière selon qu'elle jouée par un diapason ou par un piano. Le timbre du son est différent.

Des sons de même hauteur peuvent donner des sensations différentes en raison de leur timbre. Le timbre d'un son est lié à sa composition spectrale (présence, importance et durée des harmoniques) et à son évolution au cours du temps.

Le timbre d'un son dépend donc de l'instrument qui émet ce son. Les oscillogrammes de deux sons musicaux de même hauteur nous permettent de vérifier que ces deux sons ont même période, donc même fréquence fondamentale. Par contre, les allures de ces deux sons en fonction du temps sont très différentes. Les spectres de Fourier des deux sons font apparaître des harmoniques de mêmes fréquences, mais d'amplitudes différentes d'un son à l'autre.



. **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

Note	Fréquence (Hz)
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
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

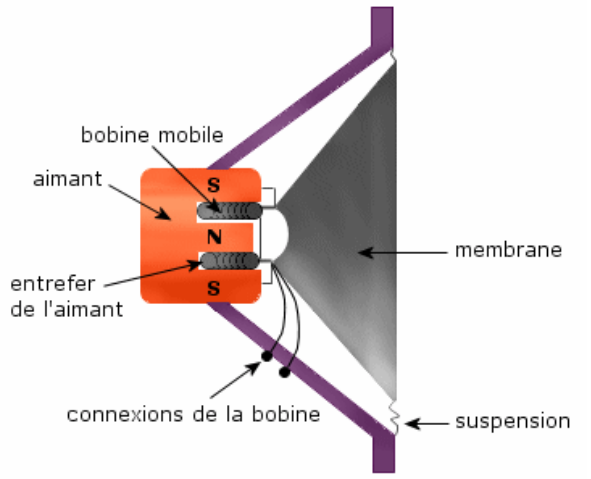
. **Arduino et ondes sonores** :

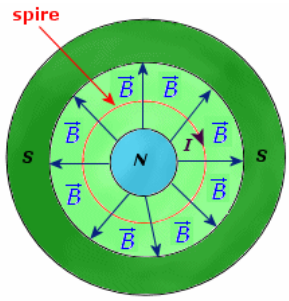
Comme une corde de guitare qui vibre et qui transmet sa vibration à l'air, pour produire un son avec un Arduino, il faut utiliser un matériel qui peut vibrer sur commande !

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.

- Principe de fonctionnement du haut-parleur :

Le haut-parleur est constitué principalement :
• d'un **aimant** circulaire ;
• d'une **bobine circulaire** mobile placée autour d'un des pôles de l'aimant ;
• d'une **membrane** reliée à la bobine ;
• d'un **support** qui contient l'aimant, la bobine et la membrane.
Les fils de la bobine sont connectés à la sortie du haut-parleur.





Le champ magnétique créé par l'aimant est perpendiculaire en tout point au courant qui circule dans chacune des spires.

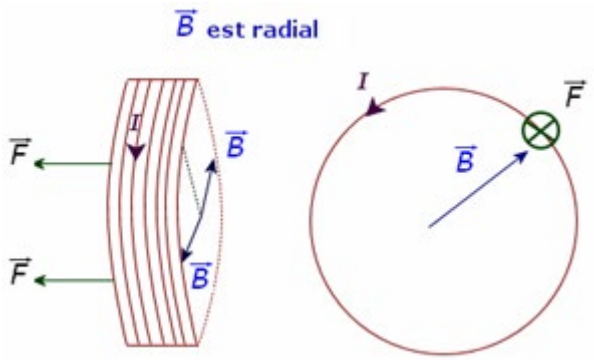
Lorsqu'un courant continu circule dans les spires, il se créé une force appelée force de Laplace. Cette force n'existe que lorsque les spires sont plongées dans un champ magnétique.

La valeur de cette force est proportionnelle à l'intensité du courant I, à la valeur du champ magnétique B et à la longueur L du conducteur qui subit la force.

On écrit : **F = I.L.B** lorsque le champ magnétique et le courant sont perpendiculaires.

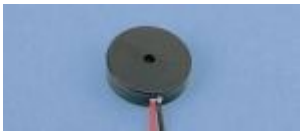
F est exprimé en newton (N), I en ampère (A), L en mètre (m) et B en tesla (T).

La direction et le sens de cette force dépendent de la direction et du sens du champ magnétique et du sens du courant dans les spires.



La sortie de l'Arduino est connectée à la bobine mobile, le courant électrique venant de l'Arduino la traverse, et sous l'action du champ magnétique intense de l'entrefer, va pousser en avant la membrane ou la tirer en arrière suivant la polarité de cette tension électrique. En se déplaçant ainsi d'avant en arrière au rythme du signal électrique, la membrane crée une onde sonore qui se propage dans l'air.

- <u>Principe de fonctionnement d'un buzzer (transducteur) piézo-électrique</u>



Un matériau piézoélectrique est une substance qui produit un courant électrique lorsqu'il est déformé. Et inversement, lorsqu'une tension électrique est appliquée sur la substance, une déformation a lieu.

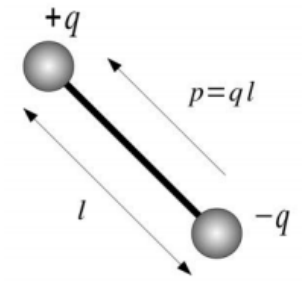
. **Description de la piézoélectricité :**

L'effet piézoélectrique résulte d'un déplacement des atomes (chargés positivement ou négativement) à l'intérieur de certains solides déformables (matériaux piézoélectriques), qui présentent des structures cristallines particulières (on parle de cristal piézoélectrique) ne présentant pas de centre de symétrie.

L'effet piézoélectrique peut être considéré à l'échelle microscopique comme un déplacement interne du barycentre des charges électriques positives et du barycentre des charges électriques négatives dans une même structure cristalline, lorsque tous les

atomes se déplacent les uns par rapport aux autres sous l'effet d'une déformation du cristal.

Lorsque ces barycentres de charges positives et négatives sont distincts, il y a polarisation (électrique) du cristal, qui se traduit par un moment $p = ql$, où q est la charge et l la distance séparant les deux charges (cf figure).



Moment dipolaire

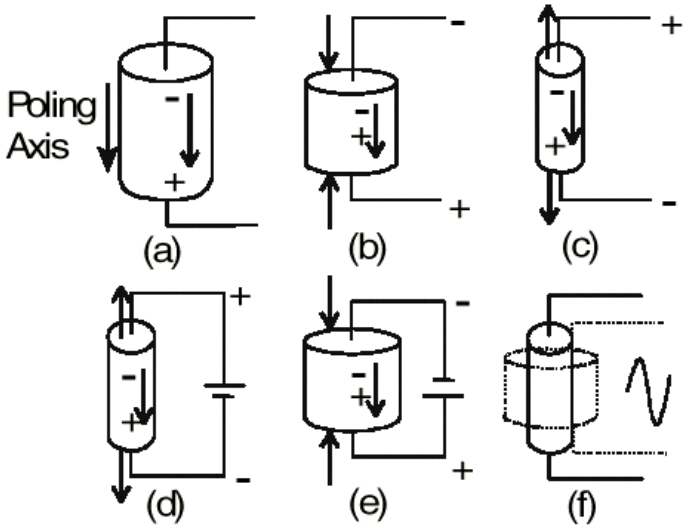
L'effet piézoélectrique peut être considéré à l'échelle macroscopique comme une polarisation électrique d'un solide déformable, sous l'effet de forces appliquées sur sa surface.

Si les faces du solide sont métallisées on peut ramener le problème à un condensateur plan au sein duquel on voit "apparaître" des charges lorsque des forces sont appliquées sur le solide (**figures b et c**)

Réciproquement, si on applique une tension sur les faces du "condensateur", on voit apparaître un champ électrique à l'intérieur du matériau (**figures d, e et f**).

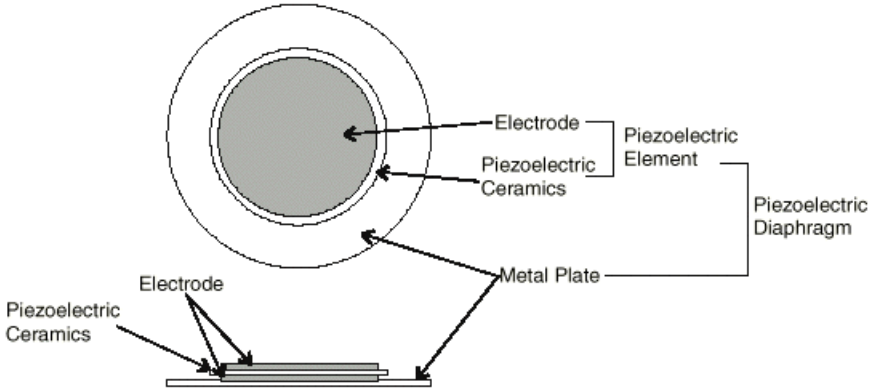
Ce champ sépare les barycentres des charges positives et négatives présentes à l'intérieur du matériau, ce qui peut se traduire soit par une déformation du matériau (si le matériau est libre de se déformer), soit par l'apparition d'une force (si on empêche le matériau de se déformer).

Le schéma ci-dessous illustre l'effet piézo-électrique :



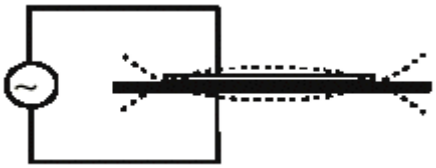
Pour les applications acoustiques, les transducteurs piézoélectriques couramment utilisés sont basés sur le dispositif suivant :

- une plaque piézoélectrique (matériau piézoélectrique entre 2 électrodes métalliques) est collée sur une plaque en laiton, formant ainsi un bilame ;



- lorsqu'un potentiel électrique est appliqué sur les électrodes portées par chaque face du matériau piézoélectrique, celui-ci s'étire ou se compresse, ce qui produit la flexion dans un sens ou dans l'autre de la plaque en laiton et donc du bilame :

- En appliquant une tension sinusoïdale sur les électrodes, le bilame se déforme dans un sens ou dans l'autre à la fréquence du signal appliqué ce qui va créer une onde sonore qui se propage dans l'air :



. La programmation

1. 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.

2. Gestion du son avec le protocole de communication "Firmata Express"

Le protocole de communication "Firmata Express" est une version améliorée de la bibliothèque "Firmata Standard 2.5.8" pour Arduino. Il a été conçu pour être utilisé conjointement avec le client Firmata "pymata-express" dans les programmes en Python.

Il prend en charge toutes les fonctionnalités de la bibliothèque "Firmata Standard 2.5.8" et ajoute en outre la prise en charge de :

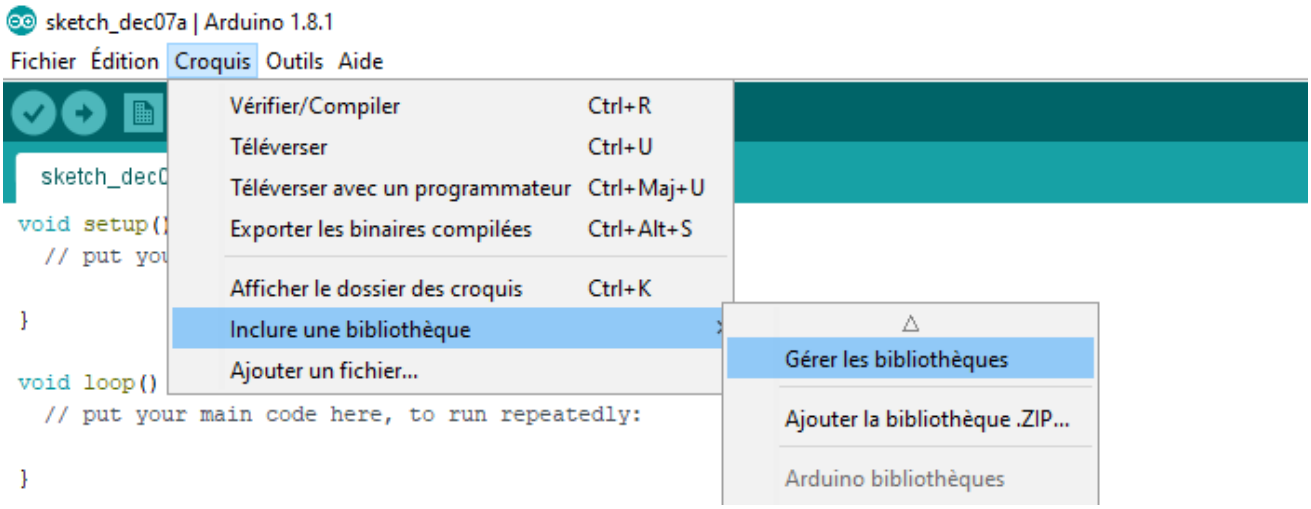
- La bibliothèque Arduino "Tone"
- Capteurs de distance HC-SR04
- Moteurs pas à pas

La liaison série fonctionne à un débit de 115200 bauds, soit deux fois la vitesse de "Firmata Standard".

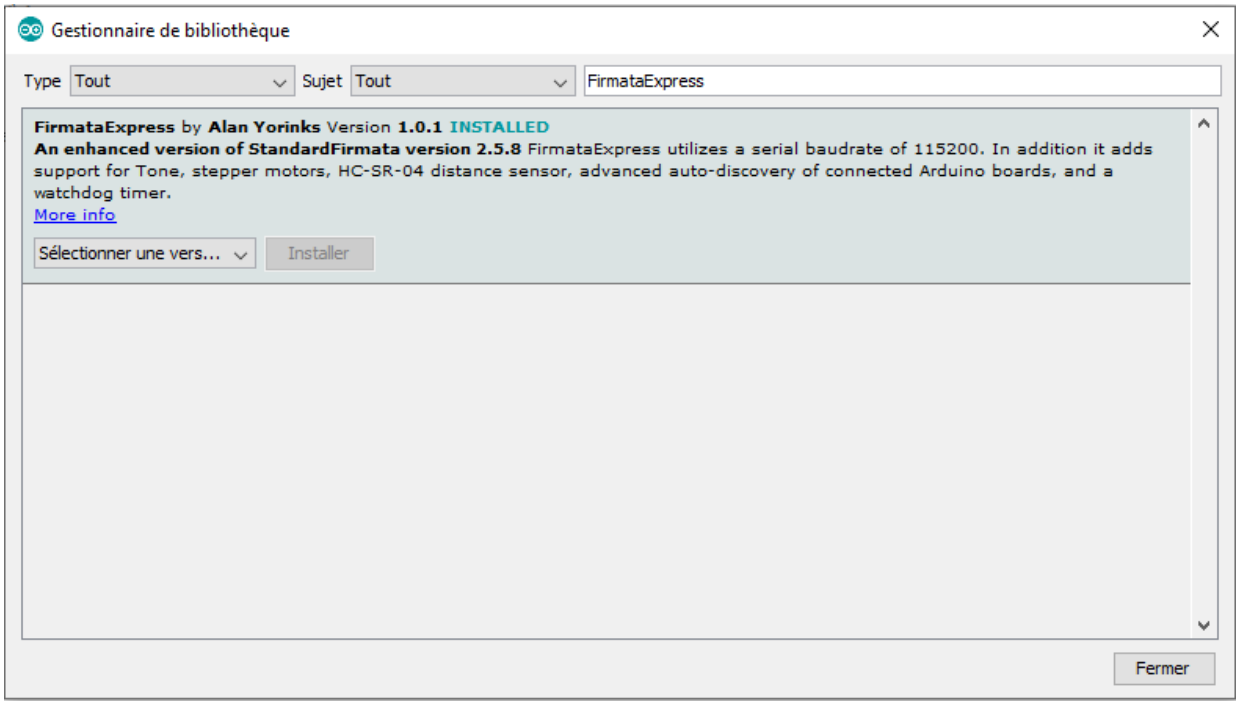
Pour que le programme Python contrôlant l'Arduino fonctionne, le chargement, dans la mémoire de l'Arduino, du code " **Firmata Express** " doit être fait avant son lancement, à l'aide du logiciel "**IDE ARDUINO**".

. **Chargement du code "Firmata Express"** :

- Brancher l'Arduino via un port USB,
- Ouvrir le logiciel "**IDE ARDUINO**",
- Sélectionner "**Croquis/Inclure une bibliothèque/Gérer les bibliothèques**",



- Entrer "**FirmataExpress**" dans la zone de recherche :



- et cliquer sur "**installer**".

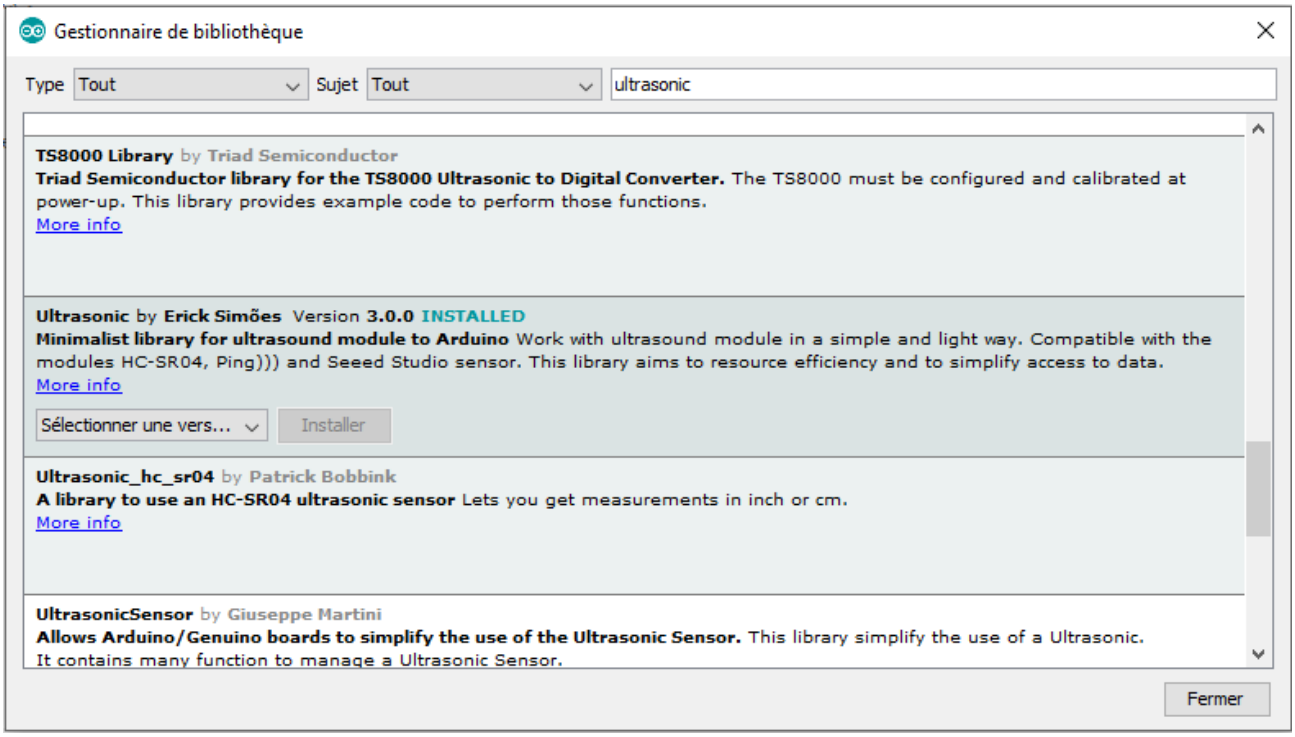
Firmata Express nécessite également que la librairie "Ultrasonic by Erick Simões" soit installée.

De même que précédemment, en utilisant le logiciel Arduino IDE :

- Sélectionner "**Croquis/Inclure une bibliothèque/Gérer les bibliothèques**",

- Entrer "**Ultrasonic**" dans la zone de recherche,

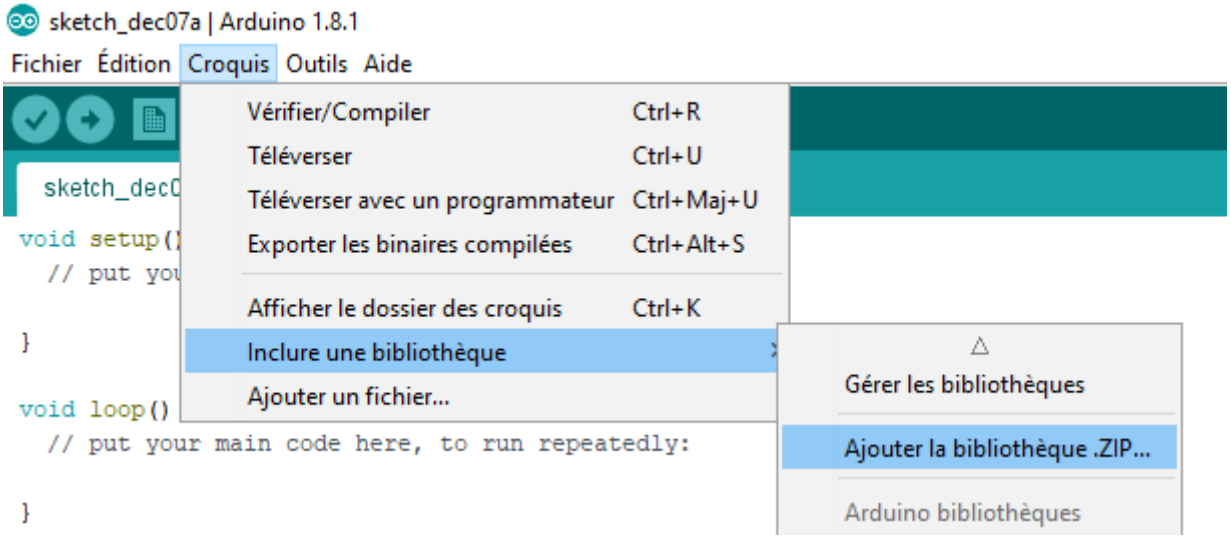
- Sélectionner "**Ultrasonic by Erick Simões**"



- et cliquer sur "**installer**".

Sans connexion internet, les librairies peuvent être installées à partir des fichiers "**zip**" présents dans le dossier "**Support/Librairies Arduino**" :

- Ouvrir le logiciel "**IDE ARDUINO**",
- Sélectionner "**Croquis/Inclure une bibliothèque/Ajouter la bibliothèque .ZIP**",
- Sélectionner le fichier .ZIP de la librairie à installer.



. <u>Installation de la bibliothèque "pymata-express" dans Python:</u>

La bibliothèque "**pymata-express**" n'est compatible qu'avec la version 3.7 de Python au minimum.
Pour faire fonctionner, un programme en Python qui contrôle l'Arduino via le protocole de communication **"Firmata Express"**, Python doit disposer de la bibliothèque "**pymata-express**". Celle-ci peut être installée via **"pip"**, à l'aide de la ligne de commande :

pip install pymata-express

Pour utiliser la bibliothèque **"pymata-express"** dans un programme python, il faut importer le module "**PymataExpress**", à l'aide de l'instruction :

from pymata_express.pymata_express import PymataExpress

La connexion avec le microcontrôleur, via le port série, est réalisée avec ce module en précisant le port COM sur lequel l'Arduino est connecté :

board = PymataExpress(com_port = PortComArduino)

Une fois la connexion établie, il est possible d'interroger ou de modifier les entrées et sorties numériques ou analogiques de l'Arduino et de produire un son.

Avec "**pymata-express**" pour utiliser une broche de l'Arduino en mode "tone", il faut au préalable la déclarer avec l'instruction :

loop.run_until_complete(board.set_pin_mode_tone(pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté le piezo ou le haut-parleur,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en mode "tone" :

```
def Set_Tone_Pin(board,pin):
  loop.run_until_complete(board.set_pin_mode_tone(pin))
```

La syntaxe pour déclarer la broche N° 3 en mode "tone" est alors plus simple :

Set_Tone_Pin(board,3)

Ensuite, on peut produire un son de fréquence "**F**" en Hz pendant une durée "**D**" en ms avec un piezo connecté sur la broche "**pin**" au moyen de cette instruction :

loop.run_until_complete(board.play_tone(pin, F, D))

L'onde sonore peut être émise de façon continue :

loop.run_until_complete(board.play_tone_continuously(pin, F))

Le plus simple est de réunir les deux instructions dans une seule fonction dont les arguments sont le N° de la broche, la fréquence et la durée, puis d'utiliser l'une ou l'autre instruction en fonction de la valeur de la durée :

```
def Tone(board,pin,freq,duration):
  if duration>0:
    loop.run_until_complete(board.play_tone(pin, freq, duration))
  else:
    loop.run_until_complete(board.play_tone_continuously(pin, freq))
```

Ainsi, la commande pour émettre un son, avec un piezo connecté à la broche N°3, à une fréquence de 440 Hz est :

- Pendant 1 s : **Tone(board,3,440,1000)**

- De façon continue : **Tone(board,3,440,0)**

Une onde sonore émise en continu est arrêtée avec l'instruction :

loop.run_until_complete(board.play_tone_off(pin))

où "**pin**" est le numéro de la broche du microcontrôleur sur laquelle est connecté le piezo ou le haut-parleur.

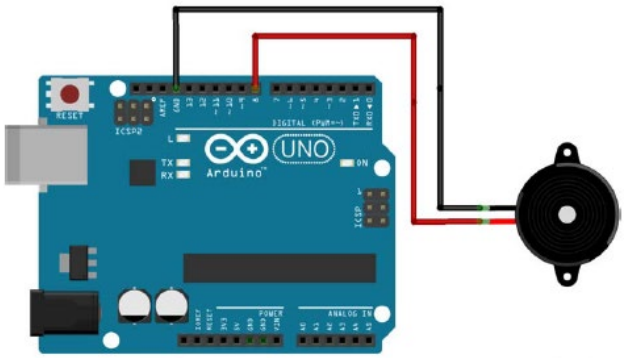
On définira une fonction utilisant cette instruction :

```
def No_Tone(board,pin):
   loop.run_until_complete(board.play_tone_off(pin))
```

Et pour arrêter l'émission sonore sur la broche N°3, l'instruction devient :

No_Tone(board,3)

. <u>Jouer une mélodie avec un Arduino</u> :



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 % :

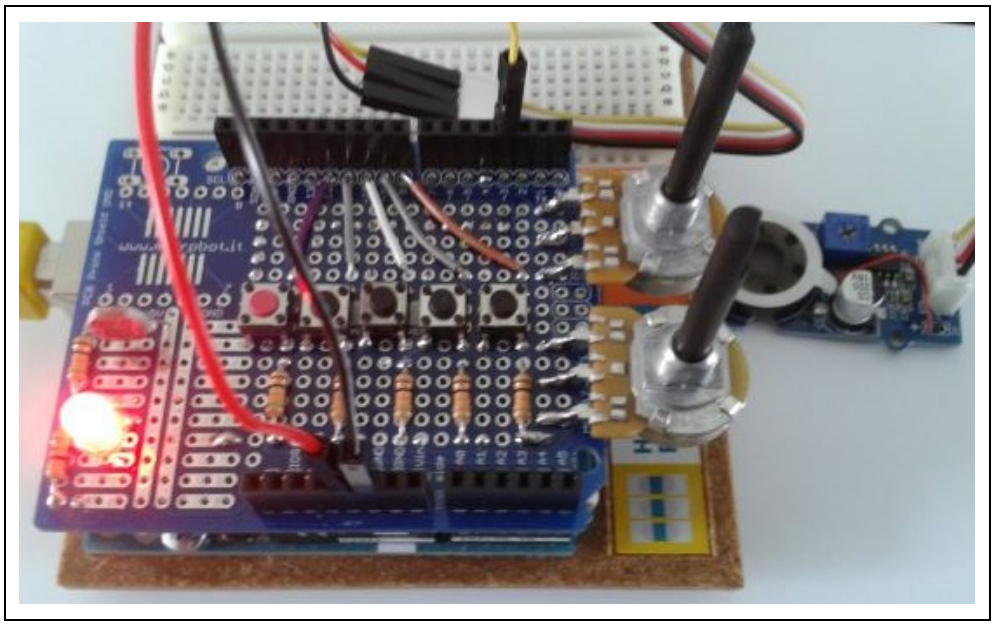
Temps de pause (en ms) = Durée de la note (en ms) x 1,3

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 :

tone (broche du piezo, Freq[i], Dur[i])

delay(Dur[i] x 1.3)

- 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 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 piezo ou un petit haut-parleur, synchrone avec le clignotement de la diode.

Rappels :

Pour modifier l'état d'une sortie numérique avec **"pymata-express"** (l'équivalent d'un "digitalWrite" en langage Arduino), il faut au préalable déclarer la broche en sortie digitale à l'aide de l'instruction suivante :

loop.run_until_complete(board.set_pin_mode_digital_output(pin))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express** ",
- "**pin**" est le numéro de la broche du microcontrôleur que l'on souhaite déclarer en sortie digitale,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

Le plus simple est de créer une fonction qui sera appelée dans nos programmes en Python pour déclarer une broche en sortie numérique :

```
def Set_DigitalOutput_Pin(board, pin):
   loop.run_until_complete(board.set_pin_mode_digital_output(pin))
```

Ainsi, l'instruction pour déclarer la broche "11", de l'objet "board", en sortie numérique est :

Set_DigitalOutput_Pin(board, 11)

L'état logique de la sortie est alors modifié à l'aide de l'instruction suivante :

loop.run_until_complete(board.digital_write(pin, val))

De même, on peut définir une fonction pour l'écriture sur une sortie digitale :

```
def Digital_Write(board, pin, val):
   loop.run_until_complete(board.digital_write(pin, val))
```

Ainsi, l'instruction pour mettre la sortie numérique "9", de l'objet "board", à l'état haut est :

Digital_Write(board, 9, 1)

Le code pourra être modifié pour voir l'influence des variables (fréquence de l'onde sonore, durée d'émission, durée de silence).

. Programme en Python (Projet3\Activity1\PY\Activity1.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinTone = 3
PinLED = 4
FreqTone = 440
TimeSleep1 = 0.5
TimeSleep2 = 0.5


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_Tone_Pin(board, PinTone)
Set_DigitalOutput_Pin(board, PinLED)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter")


# Boucle principale du programme

while True:
    try:
        Digital_Write(board, PinLED, 1)
        Tone(board, PinTone, FreqTone, 0)
        time.sleep(TimeSleep1)
        Digital_Write(board, PinLED, 0)
        No_Tone(board, PinTone)
        time.sleep(TimeSleep2)

    except KeyboardInterrupt:
        No_Tone(board, PinTone)
        Digital_Write(board, PinLED, 0)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…) ,

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinLED = 4** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. PinTone = 3 (constante correspondant au n° de la broche sur laquelle le buzzer est connecté)

. FreqTone = 440 (constante correspondant à la fréquence de l’onde sonore)

. TimeSleep1 = 0.5 (cst correspondant à la durée en s de l’émission sonore et d’allumage de la DEL)

. TimeSleep2 = 0.5 (cst correspondant à la durée en s du silence et de l’extinction de la DEL)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d’ouverture du port COM sélectionné et connexion à l’Arduino :

```
PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)
```

. Si la connexion à l’Arduino est réussie :

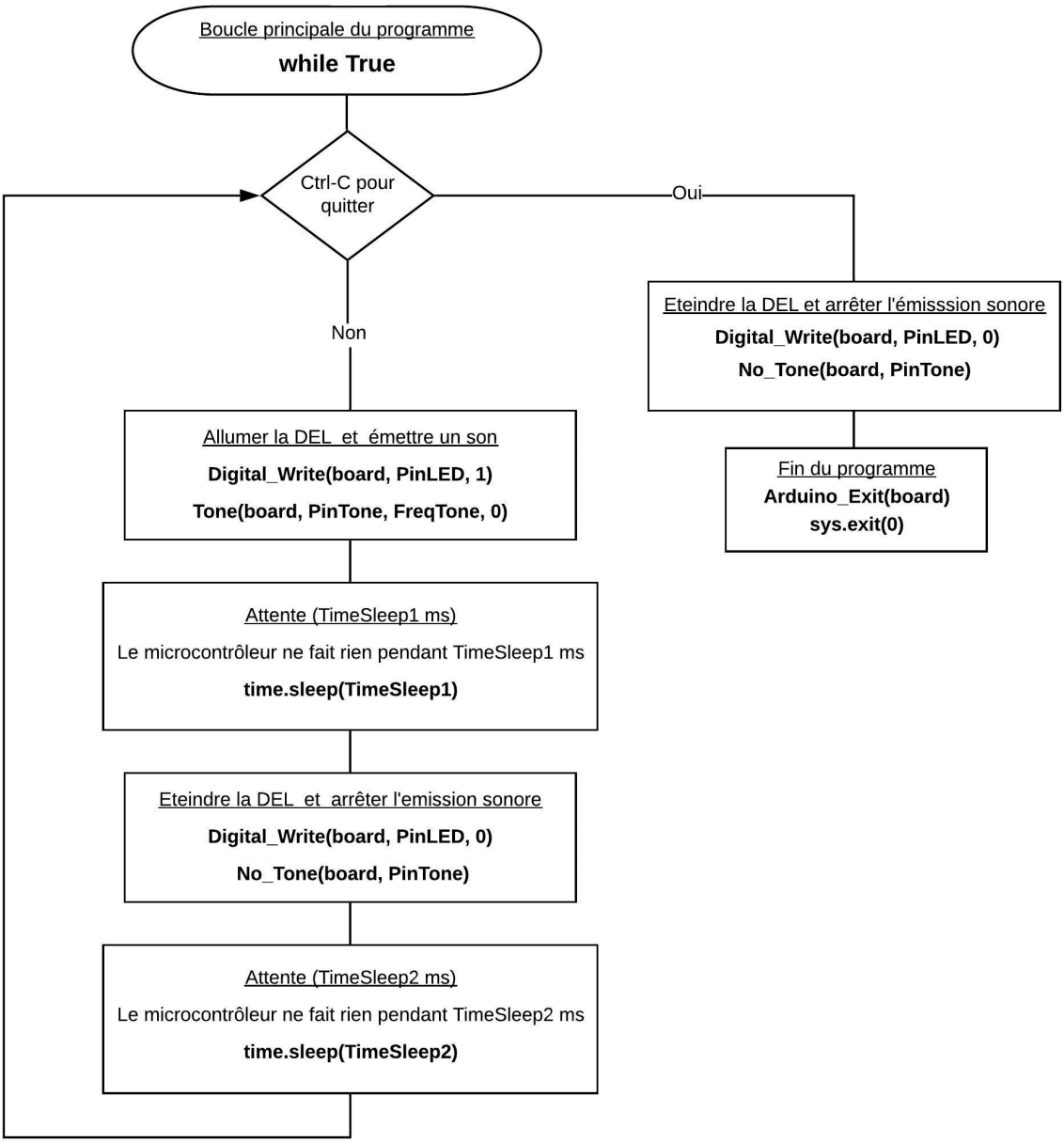
- Déclaration de la broche du buzzer en mode ”Tone” :

```
Set_Tone_Pin(board, PinTone)
```

- Déclaration de la broche de la DEL en sortie digitale :

```
Set_DigitalOutput_Pin(board, PinLED)
```

- Boucle principale du programme (boucle "while True") :

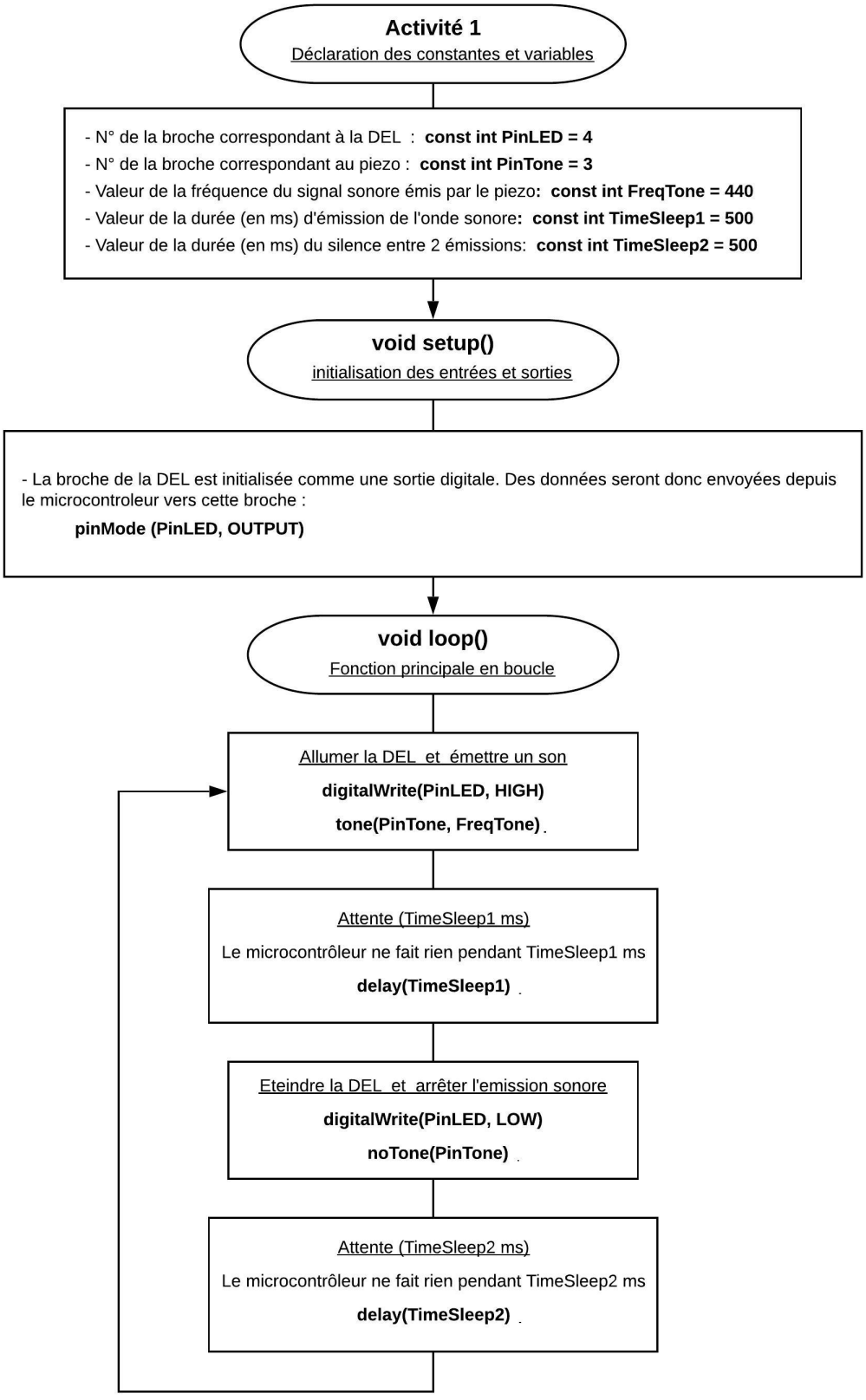


. Programme en langage Arduino (Projet3\Activity1\INO\Activity1.ino)

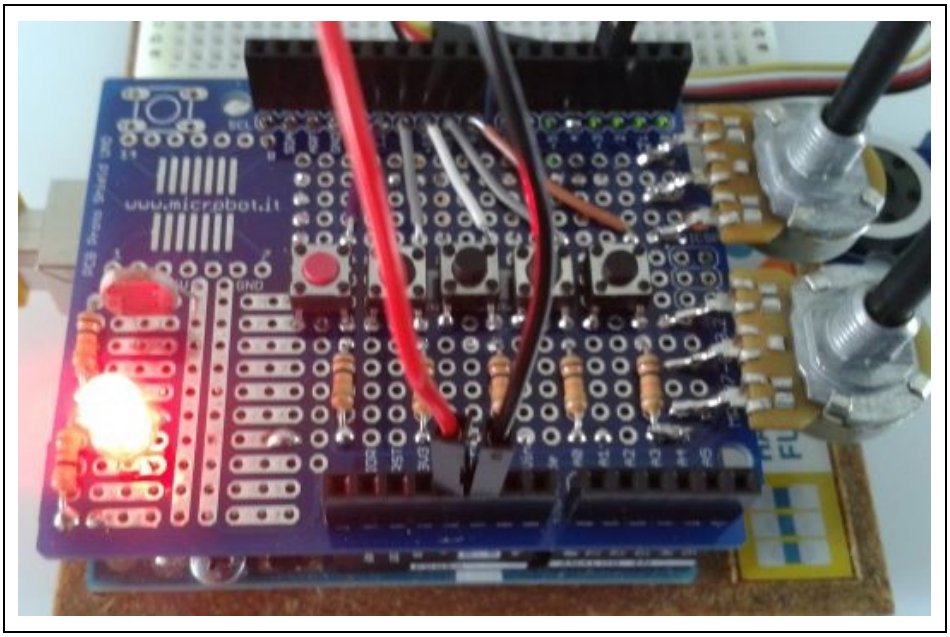
Activity1

```
// Déclaration des constantes et variables

const int PinLED = 4;
const int PinTone = 3;
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);
}
```

Déroulement du programme :



- Activité 2 : Alarme sonore par détection de passage



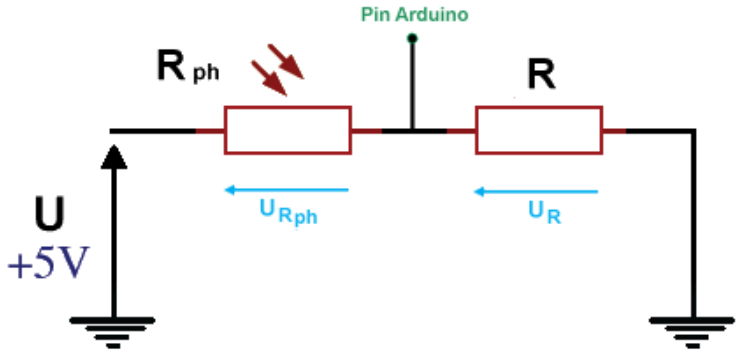
Dans cette activité, le programme de production d'un "beep" de l'activité précédente va être utilisé comme alarme de détection de passage.

On utilise pour cela une photorésistance éclairée par une DEL rouge. La sortie de la photorésistance est connectée à l'entrée analogique **A0** de l'Arduino.
La valeur de la broche **A0** 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 A0 diminue et quand celle-ci est inférieure à un seuil (la sensibilité du capteur définie initialement), l'alarme sonore est déclenchée.

Rappels :

Le circuit avec la photorésistance est représenté ci-dessous :



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 A0 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 : $$U_R = R\, I = \frac{R}{(R_{ph} + R)}\, U$$

Gestion des entrées analogiques par **"pymata-express"**

Avec **"pymata-express"**, pour lire la valeur de la tension d'une entrée analogique (par exemple, la broche A0), il faut la déclarer au préalable en entrée analogique avec la commande suivante :

loop.run_until_complete(board.set_pin_mode_analog_input(0))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express**",
- "**0**" est le numéro de la broche A0 du microcontrôleur que l'on souhaite déclarer en entrée analogique,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en entrée analogique :

```
def Set_AnalogInput_Pin(board, pin):
   loop.run_until_complete(board.set_pin_mode_analog_input(pin))
```

La syntaxe pour déclarer la broche A0 en entrée numérique est alors plus simple :

Set_AnalogInput_Pin(board,0)

Ensuite, on pourra lire la valeur de la tension de la broche au moyen de cette instruction :

value = loop.run_until_complete(board.analog_read(0))

qui retourne une liste dont le premier élément (**value[0])** est un nombre entier décimal entre 0 et 1023 représentatif d'une tension entre 0 et 5V.

De même, on peut définir une fonction pour lire la valeur de la tension d'une entrée analogique :

```
def Analog_Read(board, pin):
    value = loop.run_until_complete(board.analog_read(pin))
    return value[0]
```

Ainsi, l'instruction pour lire la valeur de la tension de la broche A0 devient :

ValA0 = Analog_Read(board, 0)

Le code pourra être modifié pour voir l'influence des variables (sensibilité du capteur, fréquence de l'onde sonore, durée d'émission, durée de silence).

. Programme en Python (**Projet3\Activity2\PY\Activity2.py**)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinTone=3
PinPhotoR=0
PinLED=4
CapteurSensib = 700
ValCapteur = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_Tone_Pin(board, PinTone)
Set_AnalogInput_Pin(board, PinPhotoR)
Set_DigitalOutput_Pin(board, PinLED)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter")

Digital_Write(board, PinLED, 1)


# Boucle principale du programme

while True:
    try:
        ValCapteur = Analog_Read(board,PinPhotoR)
        if ValCapteur < CapteurSensib:
            Tone(board,PinTone,440,0)
            time.sleep(0.2)
            No_Tone(board,PinTone)
            time.sleep(0.2)
        else:
            No_Tone(board,PinTone)

    except KeyboardInterrupt:
        Digital_Write(board, PinLED, 0)
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinLED = 4** (constante correspondant au n° de la broche sur laquelle la DEL rouge est connectée)

. PinTone = 3 (constante correspondant au n° de la broche sur laquelle le buzzer est connecté)

. PinPhotoR = 0 (constante correspondant au n° de la broche A0 de la photorésistance)

. CapteurSensib = 700 (constante nombre entier entre 0 et 1023 correspondant à la valeur de la sensibilité du capteur)

. ValCapteur = 0 (variable pour stocker la valeur de la broche de la photorésistance)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d’ouverture du port COM sélectionné et connexion à l’Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l’Arduino est réussie :

- Déclaration de la broche du buzzer en mode ”Tone” :

Set_Tone_Pin(board, PinTone)

- Déclaration de la broche de la DEL en sortie digitale :

Set_DigitalOutput_Pin(board, PinLED)

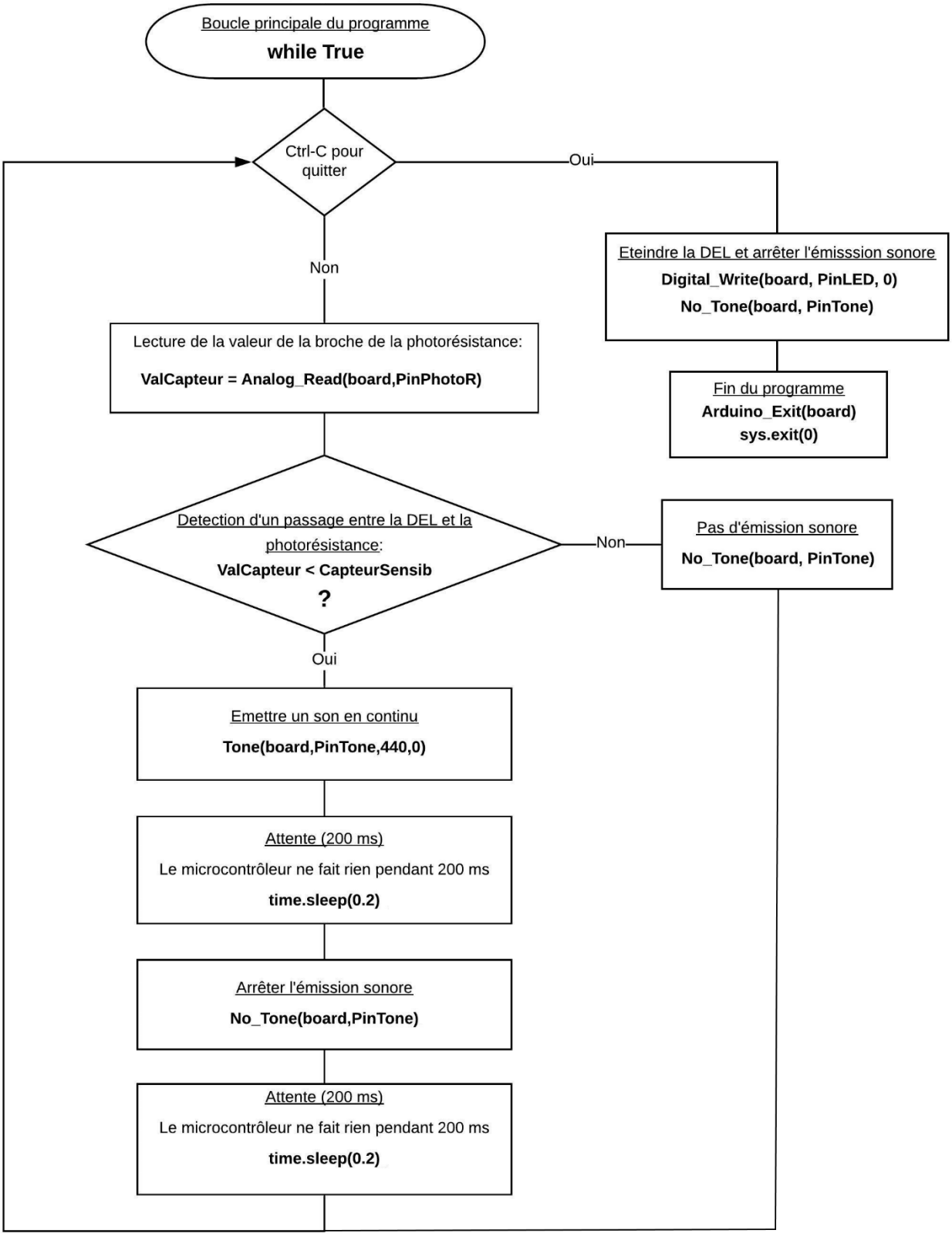
- Déclaration de la broche de la photorésistance en entrée analogique :

Set_AnalogInput_Pin(board, PinPhotoR)

- La DEL rouge est allumée :

Digital_Write(board, PinLED, 1)

- Boucle principale du programme (boucle "while True") :

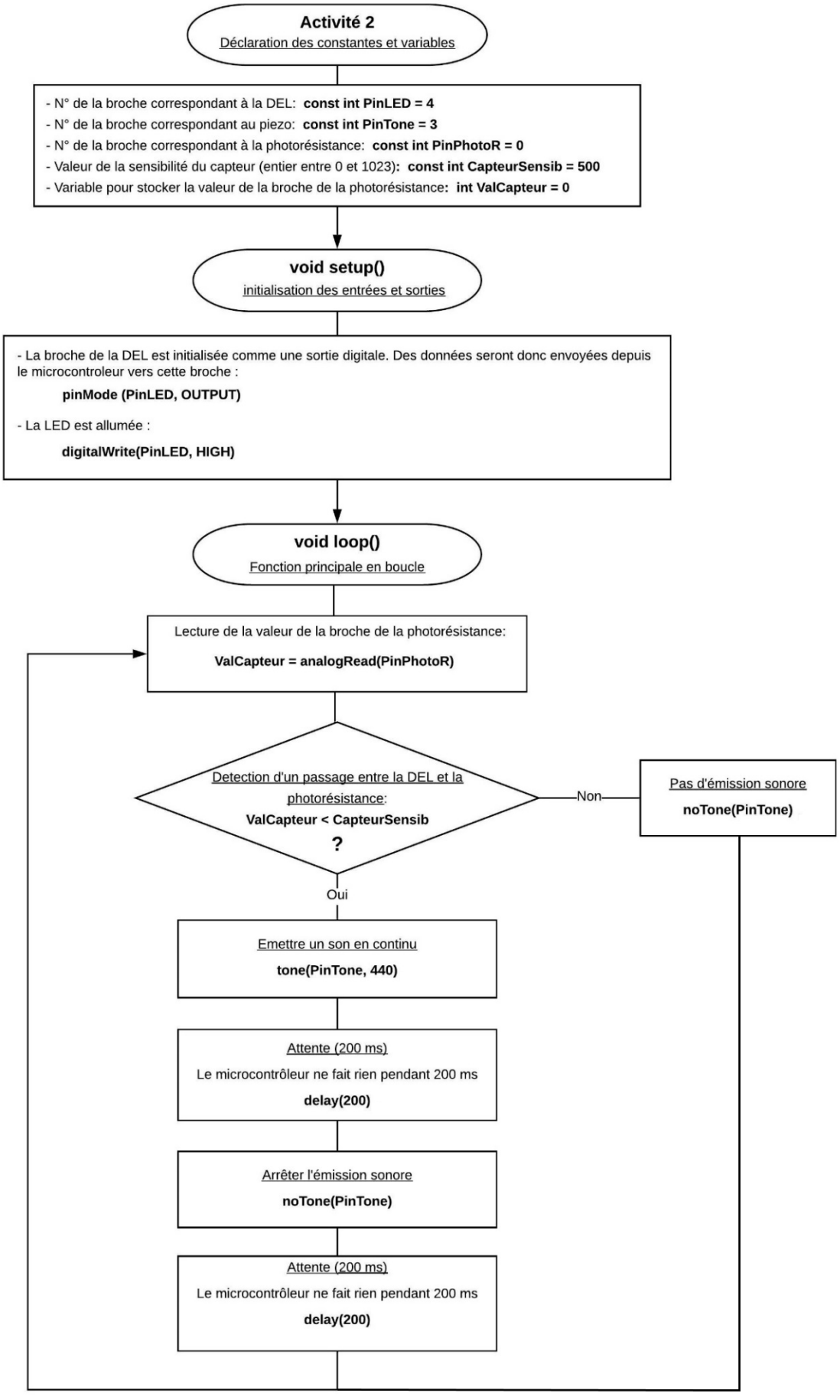


. Programme en langage Arduino (Projet3\Activity2\INO\Activity2.ino)

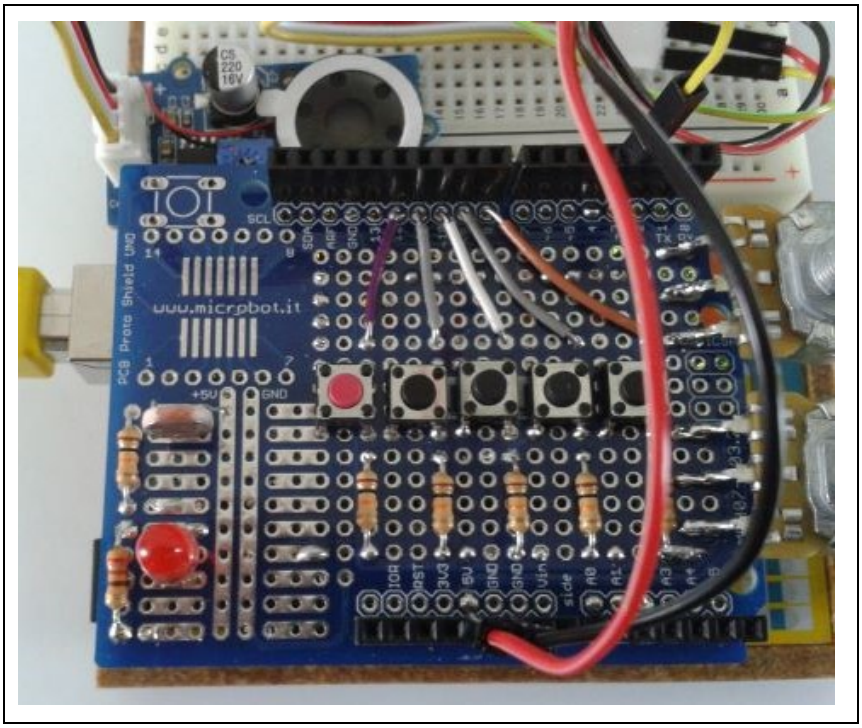
Activity2

```
// Déclaration des constantes et variables

const int PinLED = 4;
const int PinTone = 3;
const int PinPhotoR = 0;
const int CapteurSensib = 500;

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(200);
    noTone(PinTone);
    delay(200);
  }
  else {
  noTone(PinTone);
  }
}
```

Déroulement du programme :



- Activité 3 : Emettre une onde sonore avec un bouton-poussoir



Dans cette activité, nous allons commander la production d'une onde sonore de fréquence préalablement choisie en appuyant sur le premier bouton poussoir. L'émission est arrêtée en relâchant le bouton poussoir.

Rappels :

Avec **"pymata-express"**, pour lire l'état logique d'une broche numérique (par exemple, la broche N°12), il faut la déclarer au préalable en entrée avec la commande suivante :

loop.run_until_complete(board.set_pin_mode_analog_input(12))

où :

- **"board"** est l'objet créé lors de l'appel de la méthode "**PymataExpress**" du module "**pymata-express** ",
- "**12**" est le numéro de la broche du microcontrôleur que l'on souhaite déclarer en entrée digitale,
- "**loop**" est la boucle des tâches "**asyncio**" déclarée ainsi **:**

loop = asyncio.get_event_loop()

On peut définir une fonction déclarant plus facilement une broche en entrée numérique :

```
def Set_DigitalInput_Pin(board, pin):
    loop.run_until_complete(board.set_pin_mode_digital_input(pin))
```

La syntaxe pour déclarer la broche N°12 en entrée numérique est alors plus simple :

Set_DigitalInput_Pin(board,12)

Ensuite on pourra lire l'état logique de la broche au moyen de cette instruction :

value = loop.run_until_complete(board.digital_read(12))

qui retourne une liste dont le premier élément (**value[0])** est égale à "**1**" lorsque l'entrée est à **5 V**, et "**0**" lorsqu'elle est à **0 V**.

De même, on peut définir une fonction pour lire l'état logique d'une entrée digitale :

```
def Digital_Read(board, pin):
    value = loop.run_until_complete(board.digital_read(pin))
    return value[0]
```

Ainsi, l'instruction pour lire l'état logique de la broche N°12 devient :

ValPin12 = Digital_Read(board, 12)

Le code pourra être modifié pour voir l'influence des variables (fréquence onde sonore en Hz).

. Programme en Python (**Projet3\Activity3\PY\Activity3.py**)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinTone = 3
PinButton = 12
FreqWave = 440
ValButton = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_Tone_Pin(board, PinTone)
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter")


# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)

        if ValButton == 1: Tone(board, PinTone, FreqWave, 0)
        else:  No_Tone(board, PinTone)

    except KeyboardInterrupt:
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinTone = 3** (constante correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **FreqTone = 440** (constante correspondant à la fréquence de l’onde sonore)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l’état logique de la broche du bouton poussoir)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d’ouverture du port COM sélectionné et connexion à l’Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l’Arduino est réussie :

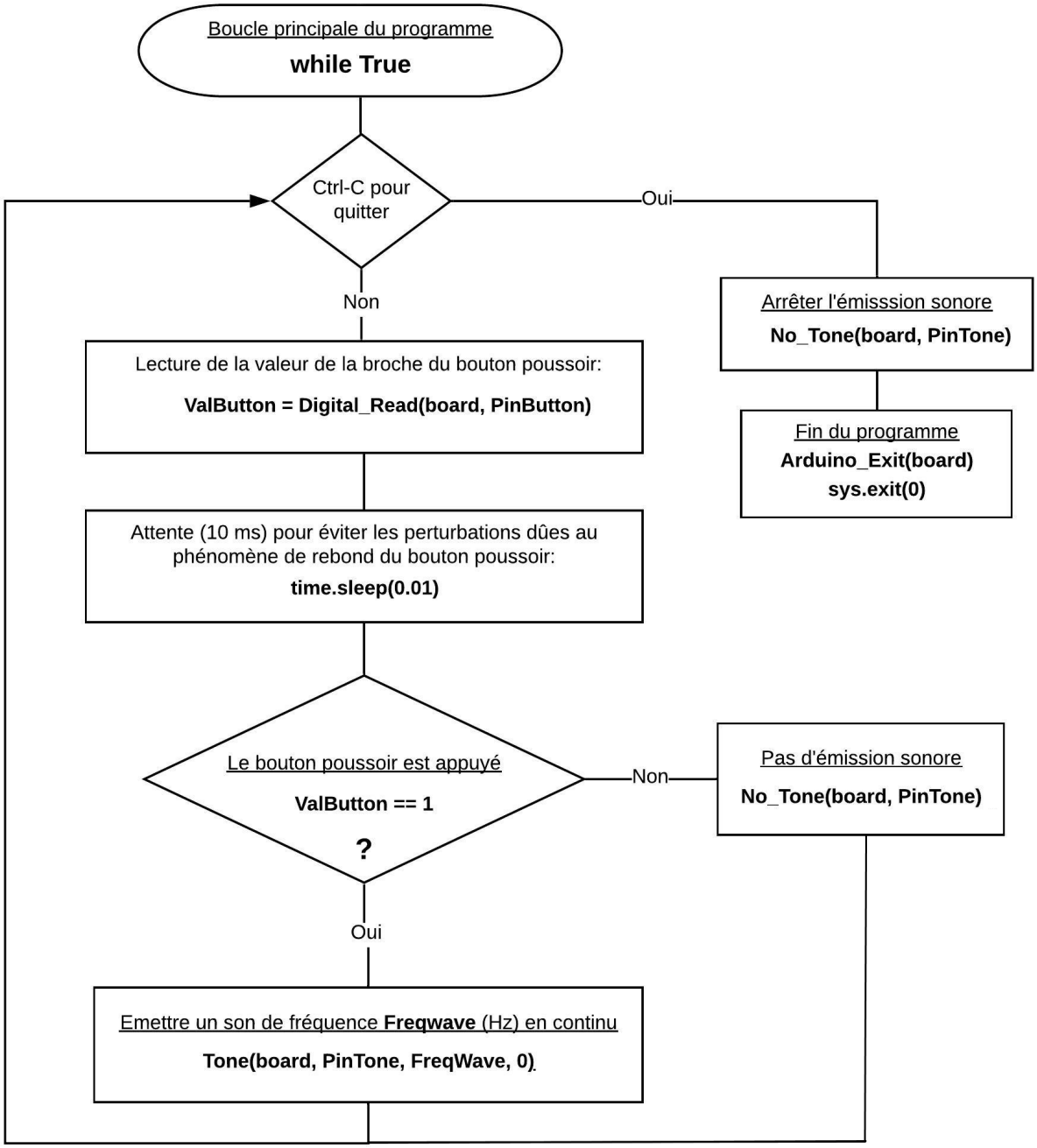
- Déclaration de la broche du bouton poussoir en entrée digitale :

Set_DigitalInput_Pin(board, PinButton)

- Déclaration de la broche du buzzer en mode ”Tone” :

Set_Tone_Pin(board, PinTone)

- Boucle principale du programme (boucle "while True") :

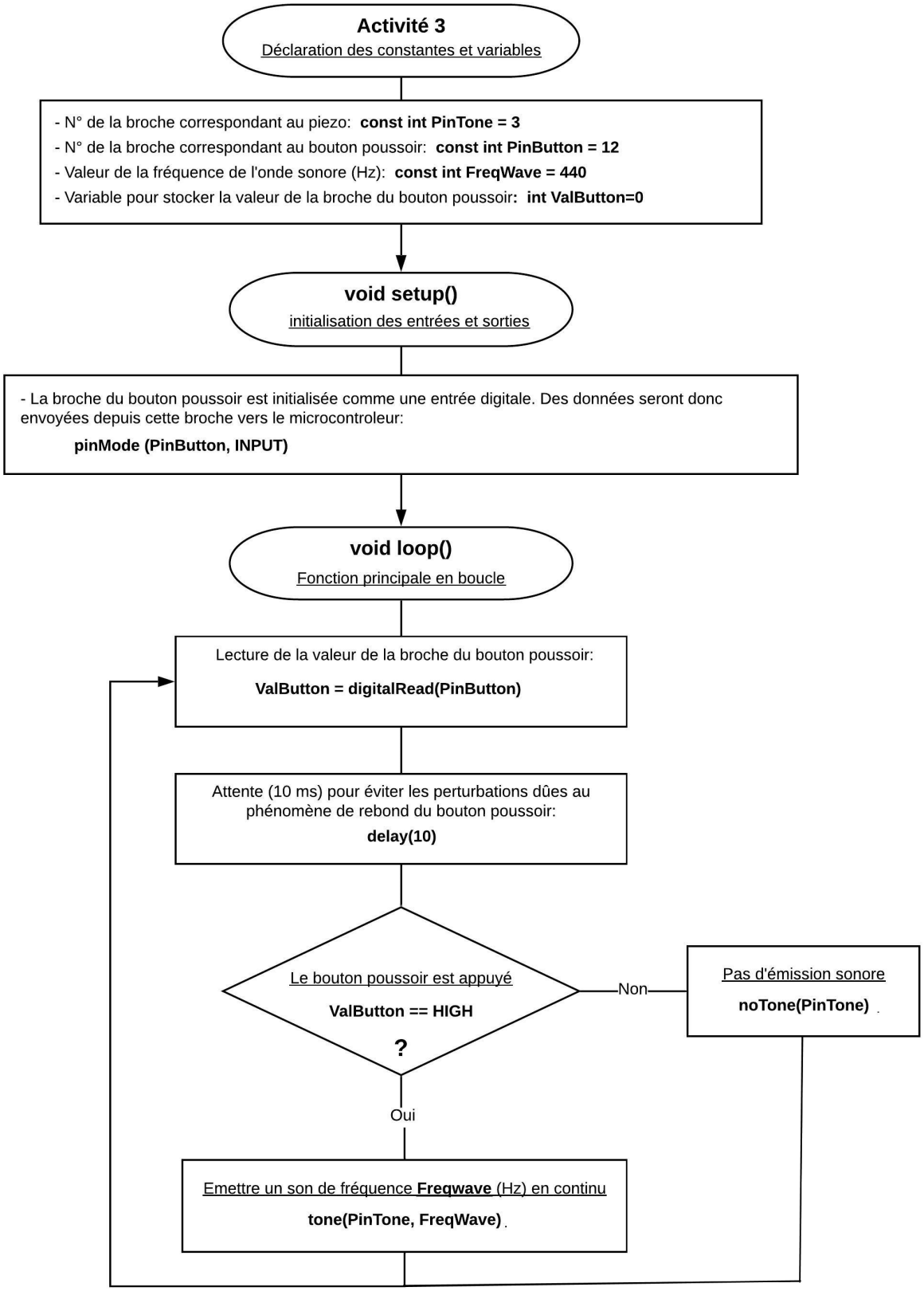


. Programme en langage Arduino (Projet3\Activity3\INO\Activity3.ino)

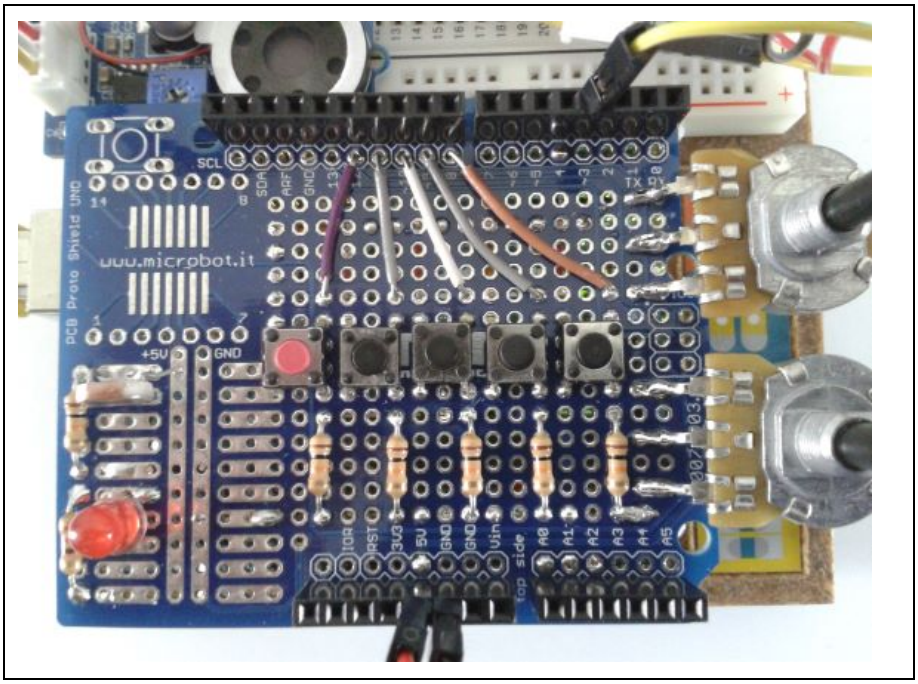
Activity3

```
// Déclaration des constantes et variables

const int PinTone = 3;
const int PinButton = 12;
const int FreqWave = 440;

int ValButton=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) {
    tone(PinTone, FreqWave);
  }
  else {
    noTone(PinTone);
  }
}
```

Déroulement du programme :



- Activité 4 : Jouer une mélodie avec des boutons poussoir

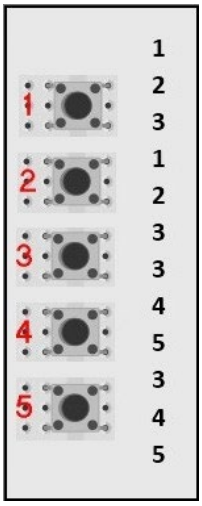


Dans cette activité, nous allons voir qu'il est possible de jouer une mélodie avec un Arduino et des boutons poussoir qui vont simuler les touches d'un piano :

- On dispose de 5 boutons poussoir que l'on associe chacun à une note de musique (une onde sonore de fréquence déterminée en Hz - voir tableau des fréquences des notes de musique) et à une durée d'émission.

- L'appui sur un bouton poussoir permet de jouer la note associée au bouton pendant la durée définie.

- L'exemple de mélodie proposé dans le code sera jouée en suivant la séquence ci-contre :



Le code pourra être modifié pour voir l'influence des variables (fréquence des notes associées aux boutons en Hz, durée de la note).

. Programme en Python (Projet3\Activity4\PY\Activity4.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time


# Déclaration des constantes et variables

PinTone=3
PinButton=[12,11,10,9,8]
Notes=[262,294,330,349,392]
NoteDurations=[4,4,4,4,8]

PlayNote = False
Note = 0
NoteDuration = 0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)

# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_Tone_Pin(board, PinTone)
for i in range(5):
    Set_DigitalInput_Pin(board, PinButton[i])

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter")


# Boucle principale du programme

while True:
    try:
        if Digital_Read(board, PinButton[0])==1:
            PlayNote = True
            Note = 0
        elif Digital_Read(board, PinButton[1])==1:
            PlayNote = True
            Note = 1
        elif Digital_Read(board, PinButton[2])==1:
            PlayNote = True
            Note = 2
        elif Digital_Read(board, PinButton[3])==1:
            PlayNote = True
            Note = 3
        elif Digital_Read(board, PinButton[4])==1:
            PlayNote = True
            Note = 4

        if PlayNote == True :
            NoteDuration = int(1000/NoteDurations[Note])
            Tone(board, PinTone, Notes[Note], NoteDuration)
            time.sleep(1.3*NoteDuration/1000)
            PlayNote = False

    except KeyboardInterrupt:
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause.

- Déclaration des constantes et variables :

. **PinTone = 3** (constante correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **PinButton = [12,11,10,9,8]** (liste de constantes correspondant aux n° des broches des boutons-poussoir)

. **Notes = [262,294,330,349,392]** (liste des fréquences en Hz des notes associées aux boutons-poussoir)

. **NoteDurations = [4,4,4,4,8]** (liste des durées des notes en fraction de seconde)

. **PlayNote = False** (variable booléenne indiquant si une note doit être jouée)

. **Note = 0** (variable correspondant au N° de la note à jouer)

. **NoteDuration = 0** (variable pour stocker la durée de la note en ms)

. **PortComArduino** (port COM sur lequel l’Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d’ouverture du port COM sélectionné et connexion à l’Arduino :

```
PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)
```

. Si la connexion à l’Arduino est réussie :

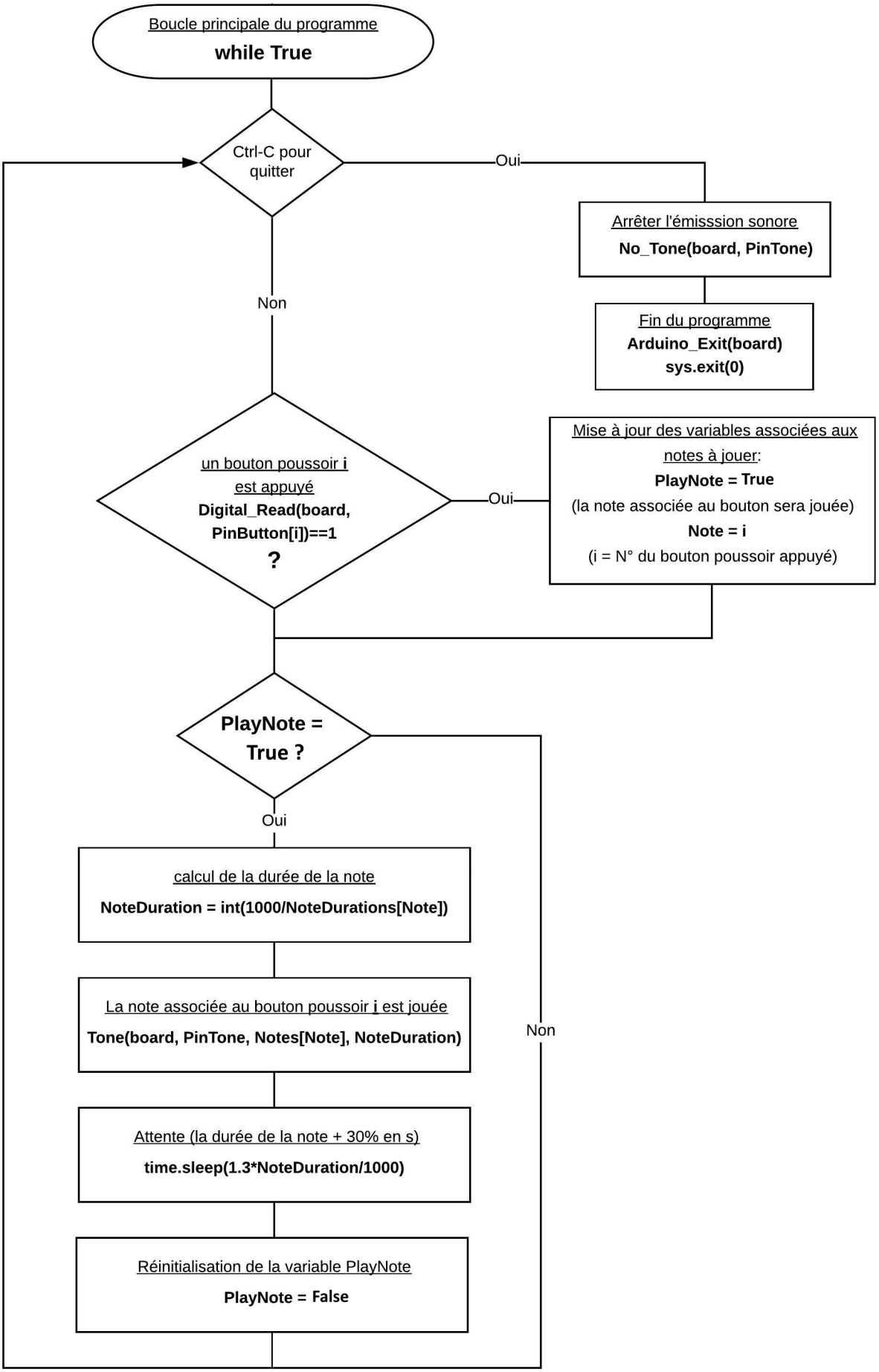
- Déclaration de la broche du buzzer en mode ”Tone” :

```
Set_Tone_Pin(board, PinTone)
```

- déclaration des broches des boutons-poussoir en entrées digitales :

```
for i in range(5):
  Set_DigitalInput_Pin(board, PinButton[i])
```

- Boucle principale du programme (boucle "while True") :

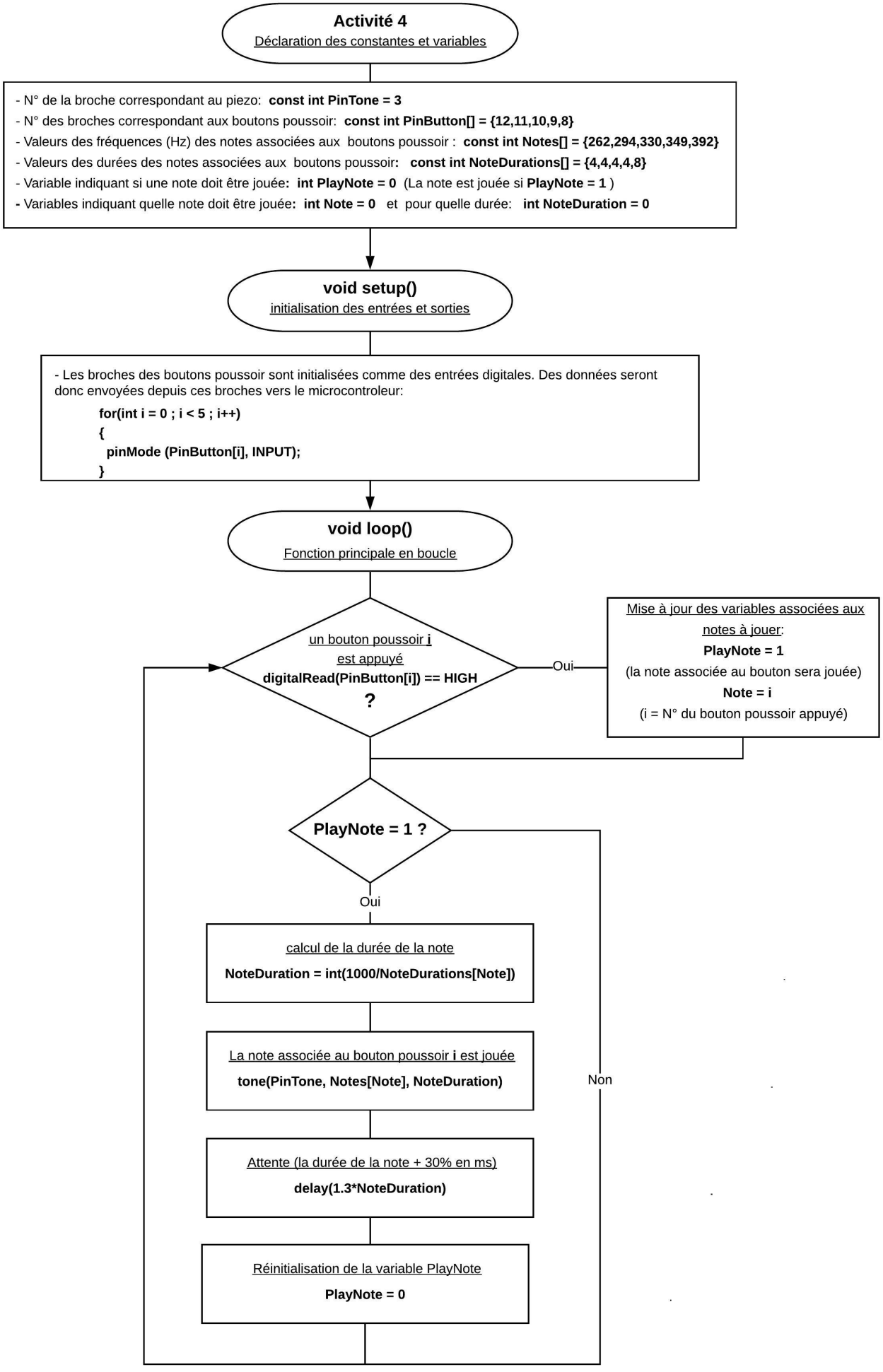


. **Programme en langage Arduino (Projet3\Activity4\INO\Activity4.ino)**

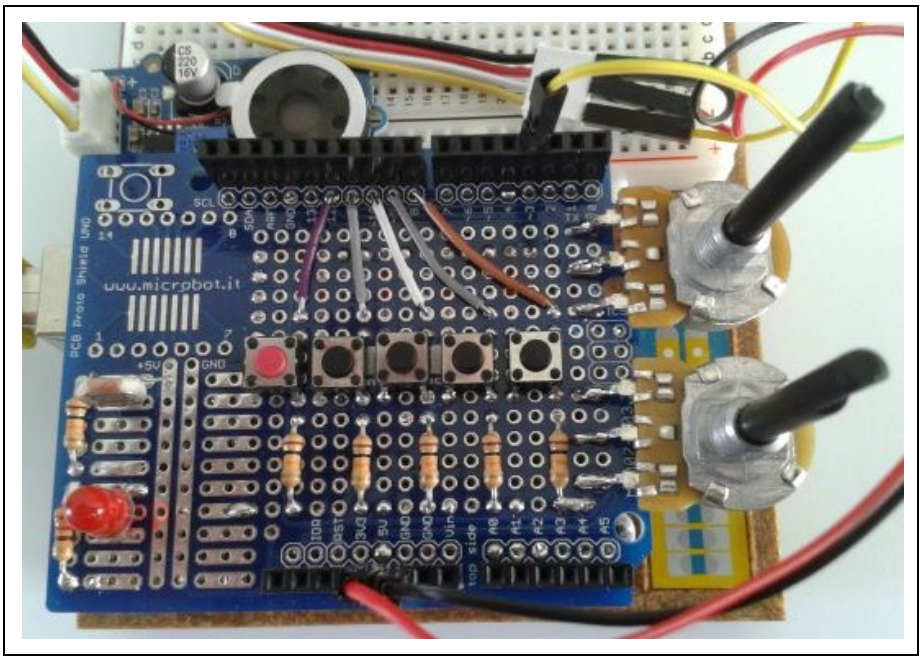
Activity4

```
// Déclaration des constantes et variables

const int PinTone = 3;
const int PinButton[] = {12,11,10,9,8};
const int Notes[] = {262,294,330,349,392};
const int NoteDurations[] = {4,4,4,4,8};

int PlayNote = 0;
int Note = 0;
int NoteDuration = 0;


// Initialisation des entrées et sorties

void setup() {
for(int i = 0 ; i < 5 ; i++)
{
  pinMode (PinButton[i], INPUT);
}
}


// Fonction principale en boucle

void loop() {

  if (digitalRead(PinButton[0]) == HIGH) {
    PlayNote = 1;
    Note = 0;
  }
  if (digitalRead(PinButton[1]) == HIGH) {
    PlayNote = 1;
    Note = 1;
  }
  if (digitalRead(PinButton[2]) == HIGH) {
    PlayNote = 1;
     Note = 2;
  }
  if (digitalRead(PinButton[3]) == HIGH) {
    PlayNote = 1;
    Note = 3;
  }
  if (digitalRead(PinButton[4]) == HIGH) {
    PlayNote = 1;
    Note = 4;
  }
  if (PlayNote == 1) {
    NoteDuration = int(1000/NoteDurations[Note]);
    tone(PinTone, Notes[Note], NoteDuration);
    delay(1.3*NoteDuration);
    PlayNote = 0;
  }
}
```

Déroulement du programme :



- Activité 5 : Régler la fréquence d'une onde sonore avec deux potentiomètres



Dans cette dernière activité, l'appui sur le premier bouton-poussoir produit une onde sonore dont la fréquence est réglée à l'aide de 2 potentiomètres :

- le premier potentiomètre permet un réglage rapide de la fréquence entre 0 et 4080 Hz,
- le deuxième potentiomètre effectue un réglage fin de la fréquence sur une plage de 255 Hz,
- l'émission sonore est arrêtée en appuyant de nouveau sur le bouton poussoir.

Le potentiomètre de réglage rapide est connecté sur la broche **A1** 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 **A1** convertie par le convertisseur analogique numérique de l'Arduino donne donc un nombre entier entre **0** et **1023**.

Ce nombre est divisé par 4 de façon à obtenir un nombre entier compris entre **0** et **255** qui sera convertie en nombre binaire (sur 8 bits) :

0 en décimal = **00000000** en binaire
255 en décimal = **11111111** en binaire

Ce nombre binaire sur 8 bits est convertie en nombre binaire sur 12 bits en ajoutant 4 bits de poids faibles, **0000**, à sa fin. On obtient donc un nombre binaire (sur 12 bits) compris entre **000000000000** et **111111110000**, soit en décimal, un nombre entier entre **0** et **4080.**

Le potentiomètre de réglage fin est connecté sur la broche **A2** de l'Arduino. Selon le même principe que précédemment, la lecture de la broche **A2** donne une valeur comprise entre **0** et **1023**.

Ce nombre est également divisé par 4 et convertie en nombre binaire sur 12 bits. On obtient donc un nombre binaire compris entre **0000000000** et **00001111111111** (entre 0 et 255 en décimal).

La conversion en décimal de l'addition des deux nombres binaires (issus de A1 et A2) nous donnent la valeur de la fréquence en Hz de l'onde sonore, soit entre **0** et **4335** Hz avec un pas de réglage de 1 Hz.

Rappel :

En informatique, outre la base 10, on utilise très fréquemment le système binaire (base 2) puisque la logique booléenne est à la base de l'électronique numérique. Deux symboles suffisent : 0 et 1. Cette unité élémentaire ne pouvant prendre que les valeurs 0 et 1 s'appelle un bit (de l'anglais binary digit). Une suite de huit bits s'appelle un octet.

Le tableau ci-dessous montre la représentation des nombres de 0 à 15 dans les bases 10 et 2 :

Décimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binaire	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

. Conversion du décimal en binaire :

La méthode de conversion la plus simple est celle de la division euclidienne par 2. Elle est facile à utiliser en programmation (il est facile d'en faire un algorithme). Voilà comment on fait :

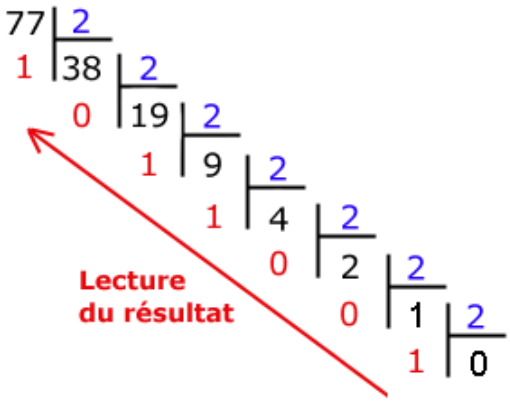
. On a notre nombre en décimal, par exemple : **77**

. On le divise par 2 et on note le reste de la division (c'est soit un 1 soit un 0).

. On refait la même chose avec le quotient précédent, et on met de nouveau le reste de côté.

. On réitère la division, et ce jusqu'à ce que le quotient soit égale à 0.

Le nombre en binaire apparaît alors. le premier bit à placer est le dernier reste non nul. Ensuite, on remonte en plaçant les restes que l'on avait. On les place à droite du premier 1 :



77 s'écrit donc en base 2 : **1001101**

. <u>Conversion du binaire en décimal</u> :

Le nombre binaire **1001101** est composé de 7 bits et chaque bit correspond à une puissance de 2. Le premier (en partant de la droite) est le bit de la puissance 0, le deuxième celui de la puissance 1, le troisième celui de la puissance 3, etc…

Pour le convertir un nombre binaire en décimal, on procède de la manière suivante :

On multiplie par 2^0 la valeur du premier bit, par 2^1 la valeur du deuxième bit, par 2^2 la valeur du troisième bit , […], par 2^{10} la valeur du onzième bit, etc… et on fait la somme des résultats :

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	1	1	0	1

Le nombre binaire **1001101** en base 10 est :

$2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77$

. Programme en Python (Projet3\Activity5\PY\Activity5.py)

```
# Importations des librairies et définition des fonctions

from PymataExpressDef import *
from ConnectToArduino import *
import time

def dec2bin(d, nb=0):
    if d == 0:
        b = "0"
    else:
        b = ""
        while d != 0:
            b = "01"[d & 1] + b
            d = d >> 1
    return b.zfill(nb)

def bin2dec(b):
    return int(b, 2)

def CalculFreq(valpot1, valpot2):
    valpot1bin = dec2bin(int(valpot1/4))
    valpot1bin= valpot1bin+'0000'
    valpot2bin = dec2bin(int(valpot2/4),12)
    freq = bin2dec(valpot1bin) + bin2dec(valpot2bin)
    return freq


# Déclaration des constantes et variables

PinButton = 12
PinTone = 3
PinPot = [1,2]
ValButton=0
OldValButton=0
State = 0
ValPot=[0,0]
FreqWave=0


# Connexion à l'Arduino

PortComArduino = SelectPortCOM()
board = OpenPortCom(PortComArduino)


# Connexion à l'Arduino réussie - Déclaration des entrées et sorties

Set_Tone_Pin(board, PinTone)
Set_AnalogInput_Pin(board, PinPot[0])
Set_AnalogInput_Pin(board, PinPot[1])
Set_DigitalInput_Pin(board, PinButton)

print("Connexion à l'arduino établie - Appuyez sur Ctrl-C pour quitter")
```

```
# Boucle principale du programme

while True:
    try:
        ValButton = Digital_Read(board, PinButton)
        time.sleep(0.01)
        if ValButton == 1 and OldValButton==0:
            State = 1 - State
        OldValButton = ValButton
        if State==1:
            for i in range(2):
                ValPot[i]= Analog_Read(board, PinPot[i])
            FreqWave = CalculFreq(ValPot[0],ValPot[1])
            Tone(board, PinTone, FreqWave, 0)
        else:
            No_Tone(board, PinTone)

    except KeyboardInterrupt:
        No_Tone(board, PinTone)
        Arduino_Exit(board)
        sys.exit(0)
```

Déroulement du programme :

- Importation des librairies et définition de fonctions :

. Le module **”ConnectToArduino.py”,** contenant les fonctions de connexion à l’Arduino via le protocole **”Firmata Express”**,

. Le module "**PymataExpressDef.Py**" regroupant toutes les fonctions utiles à l’utilisation de " **Pymata-express** " (fonction de déclaration des entrées et sorties, de lectures, d’écritures…),

. La bibliothèque **"time"** pour la gestion des temps de pause,

. La fonction **”dec2bin”** pour convertir un nombre décimal en nombre binaire,

. La fonction **”bin2dec”** pour convertir un nombre binaire en nombre décimal,

. La fonction **”CalculFreq”** pour calculer la valeur de la fréquence de l’onde sonore en fonction de la valeur des broches des potentiomètres.

- Déclaration des constantes et variables :

. **PinTone = 3** (constante correspondant au n° de la broche sur laquelle le buzzer est connecté)

. **PinButton = 12** (cst correspondant au n° de la broche sur laquelle le bouton poussoir est connecté)

. **ValButton = 0** (variable pour stocker la valeur de l’état logique de la broche du bouton poussoir)

. **OldValButton = 0** (variable pour stocker la valeur précédente de l’état logique de la broche du bouton poussoir)

. **State=0** (variable correspondant à l’action à effectuer)

. **PinPot = [1,2]** (liste de constantes correspondant au n° des broches des potentiomètres : A1 et A2)

. **ValPot = [0,0]** (liste de variables pour stocker la valeur des broches des potentiomètres)

. **FreqWave = 0** (variable pour stocker la fréquence en Hz de l'onde sonore)

. **PortComArduino** (port COM sur lequel l'Arduino est connecté)

- Connexion à l'Arduino :

. Détection du port COM, tentative d'ouverture du port COM sélectionné et connexion à l'Arduino :

PortComArduino = SelectPortCOM()

board = OpenPortCom(PortComArduino)

. Si la connexion à l'Arduino est réussie :

- Déclaration de la broche du buzzer en mode "Tone" :

Set_Tone_Pin(board, PinTone)

- déclaration de broche du bouton-poussoir en entrée digitale :

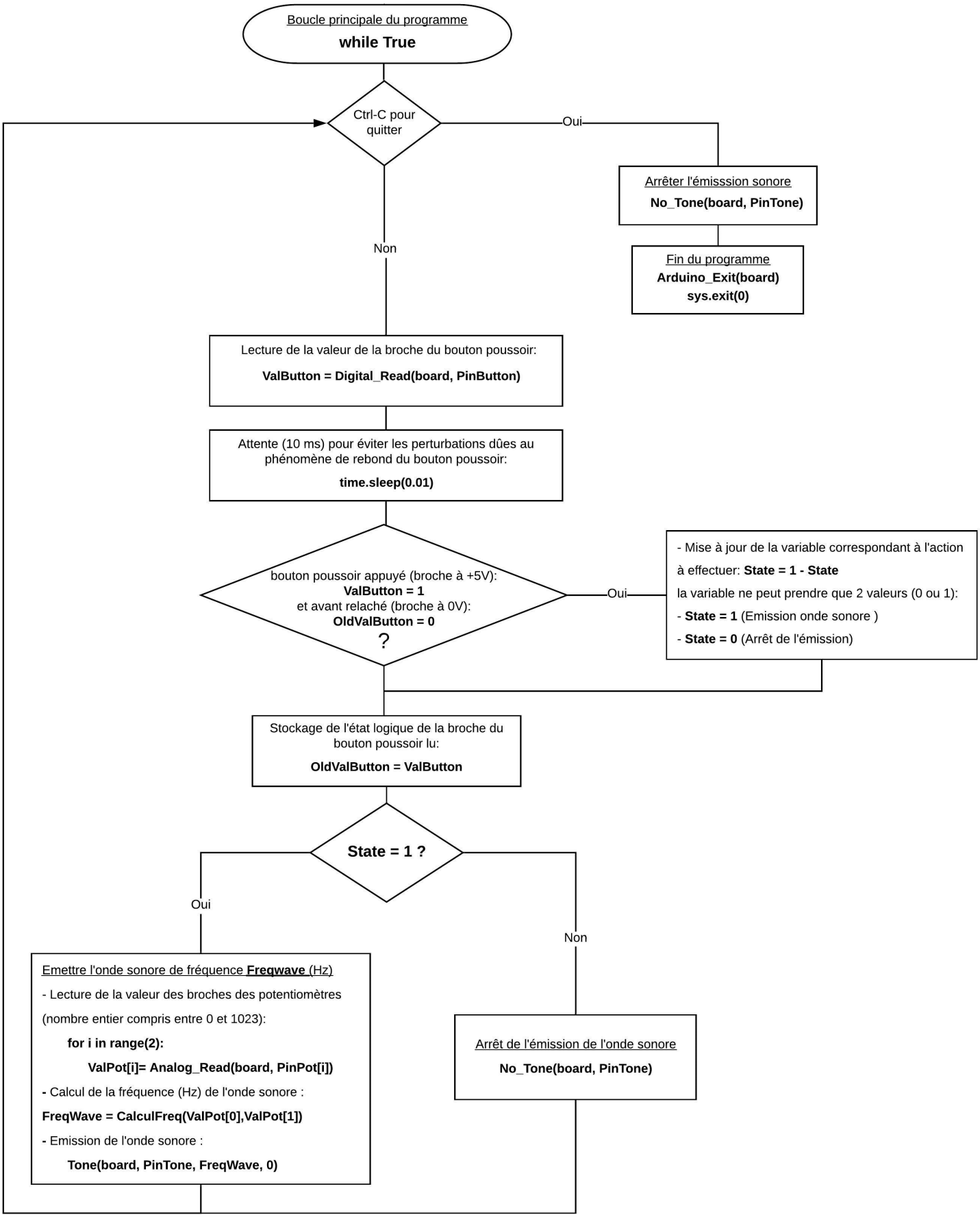
Set_DigitalInput_Pin(board, PinButton)

- déclaration des broches des potentiomètres en entrée analogique :

Set_AnalogInput_Pin(board, PinPot[0])

Set_AnalogInput_Pin(board, PinPot[1])

- Boucle principale du programme (boucle "while True") :



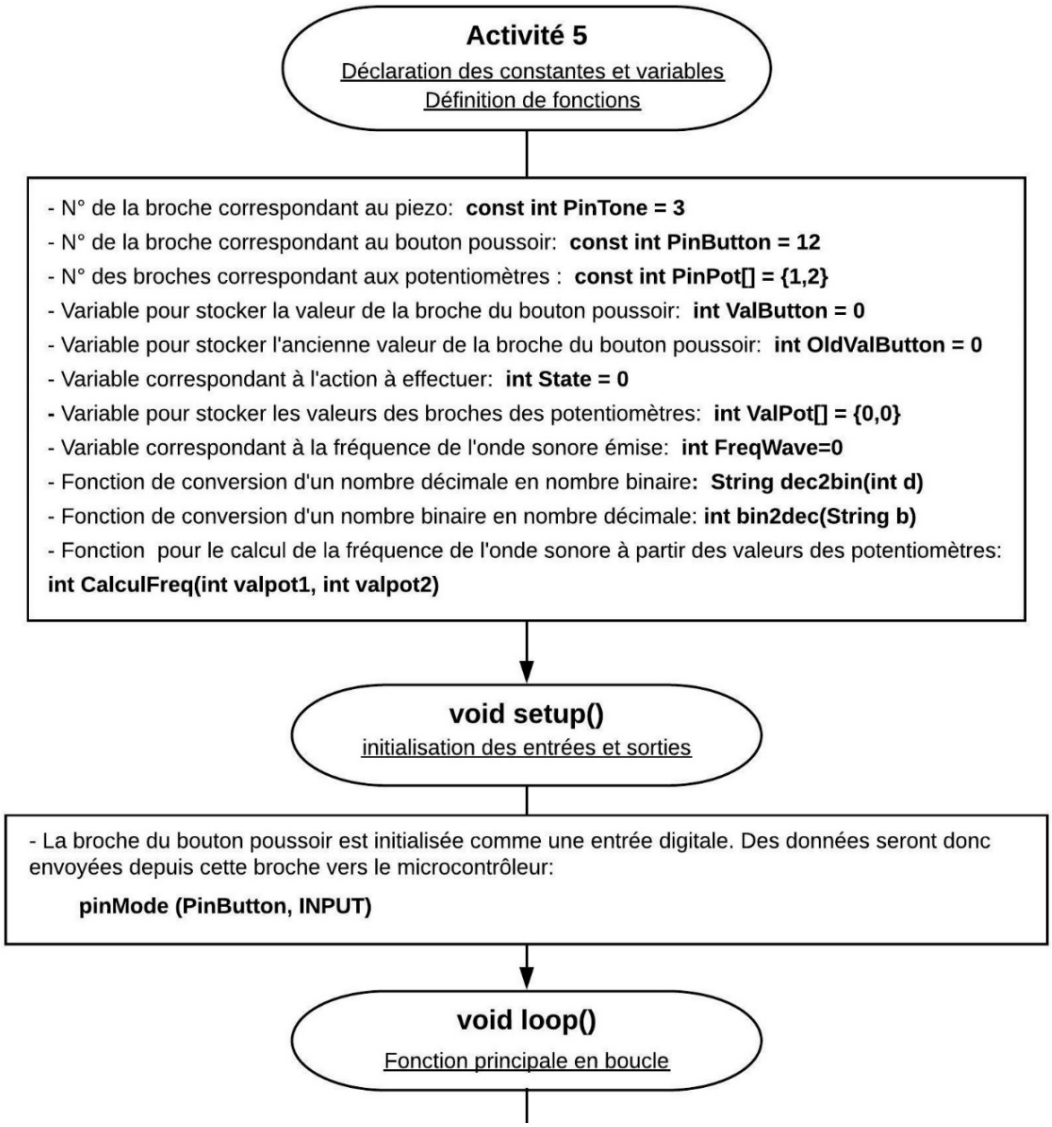
. Programme en langage Arduino (Projet3\Activity5\INO\Activity5.ino)

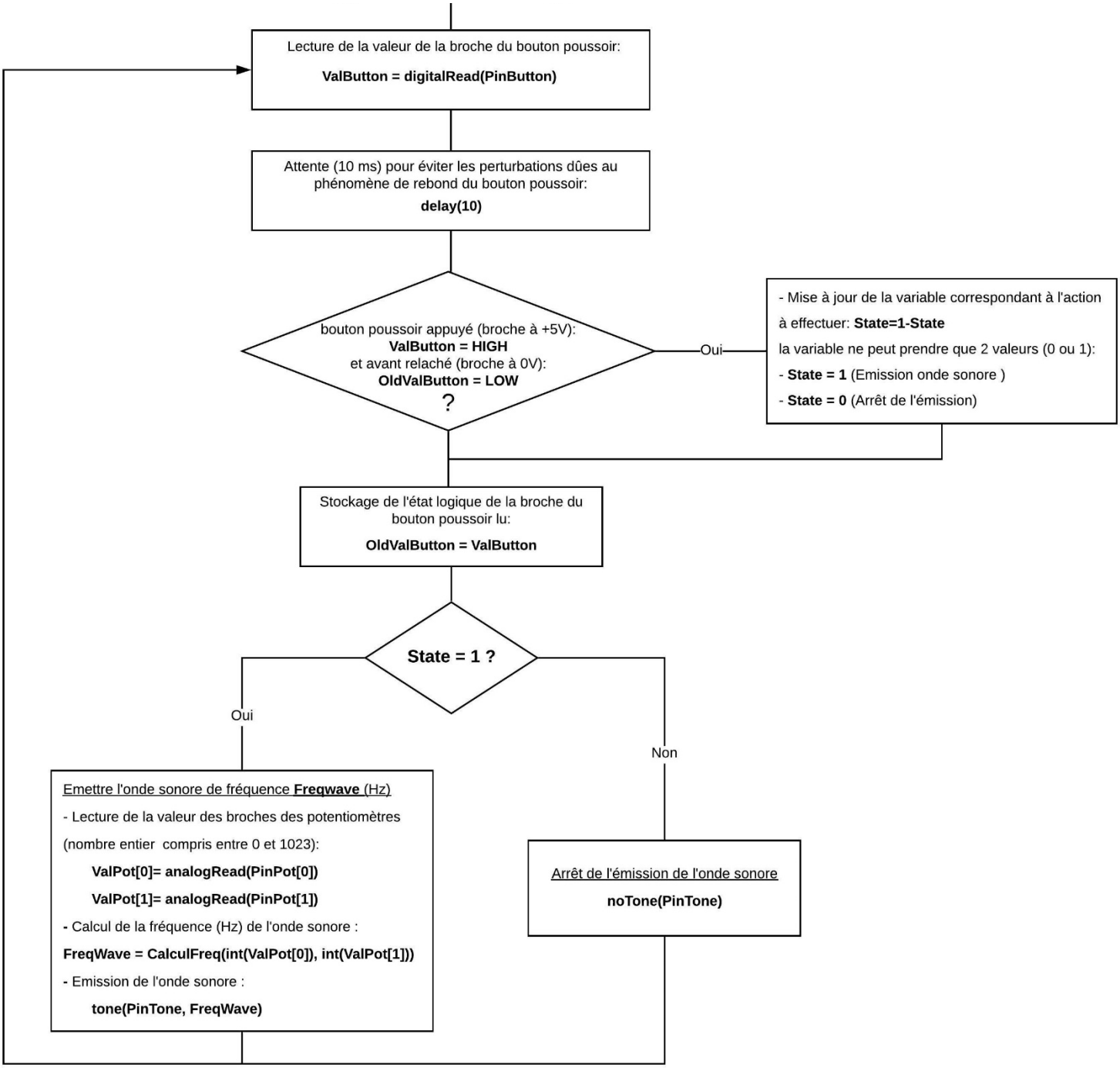
Activity5

```
// Déclaration des constantes et variables

const int PinButton = 12;
const int PinTone = 3;
const int PinPot[] = {1,2};

int ValButton=0;
int OldValButton=0;
int State = 0;
int ValPot[] = {0,0};
int FreqWave=0;


// Définition de fonctions

String dec2bin(int d){
String b;
    if (d == 0){
        b = "0";
        }
    else{
        b = "";
        while (d != 0){
            b = "01"[d & 1] + b;
            d = d >> 1;}
    }
    return b;
}

int bin2dec(String b){
int i, len;
int result=0;
len = b.length();
for(i=0; i<len; i++)
  {
    result = result*2+int(b[i]- '0');
  }
return result;
}

int CalculFreq(int valpot1, int valpot2){
String valpot1bin, valpot2bin;
int Freq;
    valpot1bin = dec2bin(int(valpot1/4));
    valpot1bin= valpot1bin+"0000";
    valpot2bin = dec2bin(int(valpot2/4));
    valpot2bin = "0000"+dec2bin(int(valpot2/4));
    Freq = bin2dec(valpot1bin) + bin2dec(valpot2bin);
    return Freq ;
}
```

```
// 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;
  }
  OldValButton = ValButton;
  if (State==1){
    ValPot[0]= analogRead(PinPot[0]);
    ValPot[1]= analogRead(PinPot[1]);
    FreqWave = CalculFreq(int(ValPot[0]), int(ValPot[1]));
    tone(PinTone, FreqWave);
  }
  else{
    noTone(PinTone);
  }
}
```

Déroulement du programme :




Lecture de la valeur de la broche du bouton poussoir:
ValButton = digitalRead(PinButton)
Attente (10 ms) pour éviter les perturbations dûes au phénomène de rebond du bouton poussoir:
delay(10)
bouton poussoir appuyé (broche à +5V):
ValButton = HIGH
et avant relaché (broche à 0V):
OldValButton = LOW
?
Oui
- Mise à jour de la variable correspondant à l'action à effectuer: State=1-State
la variable ne peut prendre que 2 valeurs (0 ou 1):
- State = 1 (Emission onde sonore)
- State = 0 (Arrêt de l'émission)
Stockage de l'état logique de la broche du bouton poussoir lu:
OldValButton = ValButton
State = 1 ?
Oui
Non
Emettre l'onde sonore de fréquence Freqwave (Hz)
- Lecture de la valeur des broches des potentiomètres (nombre entier compris entre 0 et 1023):
ValPot[0]= analogRead(PinPot[0])
ValPot[1]= analogRead(PinPot[1])
- Calcul de la fréquence (Hz) de l'onde sonore :
FreqWave = CalculFreq(int(ValPot[0]), int(ValPot[1]))
- Emission de l'onde sonore :
tone(PinTone, FreqWave)
Arrêt de l'émission de l'onde sonore
noTone(PinTone)