

Centre Universitaire Romand d'Implants Cochleariaires (CURIC)

---

## Projet Merspeakers (Logiciels, carte son et haut-parleurs)

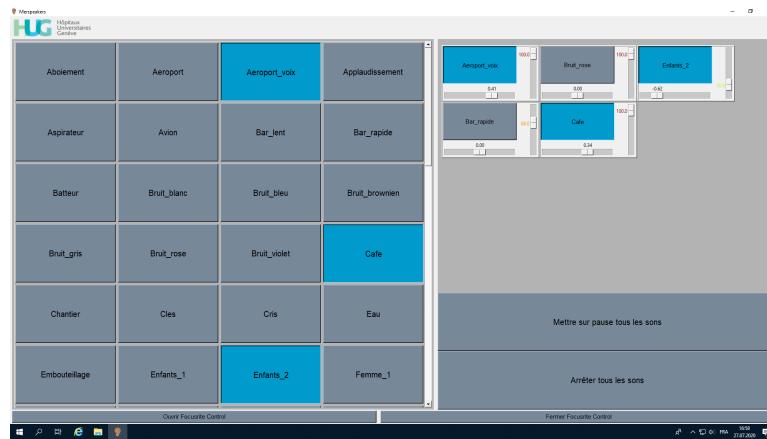
---

Stagiaire : Tristan NERSON

Encadrant : Gregory VALENTINI

Responsable : Angelica PEREZ FORNOS

30 juillet 2020



Licence 1 CMI Acoustique & Vibrations

## Remerciements

Je souhaite remercier tout particulièrement M. Gregory VALENTINI pour l'aide apportée tout au long du stage ; Mme Angelica PEREZ FORNOS et M. Tony COPPOLA pour leur confiance dans l'accomplissement de ce projet ; Mme Anissa BOUTABLA, Mme Maëlys LE MAGADOU, Mme Sabrina LENGLET, M. Samuel CAVUSCENS et M. Maurizio RANIERI pour leurs conseils avisés ; M. Dionisio CUETO LLERANDI pour ses nombreuses explications. Je remercie enfin l'ensemble du personnel du CURIC pour l'accueil chaleureux dont il a fait preuve.

## Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Sonorisation</b>	<b>4</b>
2.1	Haut-parleurs . . . . .	4
2.2	Carte son . . . . .	5
2.3	Câblages . . . . .	5
2.4	Installation du matériel . . . . .	6
<b>3</b>	<b>Logiciels</b>	<b>9</b>
3.1	Fonctionnalités des logiciels . . . . .	9
3.1.1	Utilisation du logiciel Merspeakers . . . . .	9
3.1.2	Utilisation du logiciel Focusrite Control . . . . .	11
3.2	Installation des logiciels . . . . .	13
3.2.1	Installation de Merspeakers . . . . .	13
3.2.2	Installation de Focusrite Control . . . . .	14
3.3	Accessibilité des codes sources et création de nouveaux fichiers exécutables . . . . .	15
3.3.1	Installation de Python . . . . .	15
3.3.2	Du script Python au fichier exécutable . . . . .	16
<b>4</b>	<b>Ajout de sons</b>	<b>17</b>
4.1	Banque de sons . . . . .	17
4.2	Calibration et conversion des échantillons sonores . . . . .	18
<b>5</b>	<b>Conclusion et perspectives</b>	<b>19</b>
<b>6</b>	<b>Annexes</b>	<b>20</b>

## 1 Introduction

Le projet Merspeakers (nommé en hommage à Marin Mersenne, l'un des fondateurs de l'acoustique), a été élaboré lors de mon stage d'un mois au sein du CURIC en juillet 2020. J'ai été épaulé par M. Gregory VALENTINI.

Le but du projet était de développer un logiciel capable de piloter un certain nombres (à définir) d'enceintes afin de reproduire des ambiances sonores représentatives de celles de la vie courante, dans une salle de réglage d'implants cochléaires. En effet, il était nécessaire de pouvoir tester l'efficacité des système de contrôle actif du bruit dont bénéficient certains implants sur des patients. Il fallait donc :

- Choisir et commander du matériel de sonorisation
  - Choisir du matériel capable de reproduire des ambiances fidèles à la réalité
  - Effectuer une étude de marché sur le meilleur matériel dans un budget prédéfini
  - Adapter la taille des hauts-parleurs en fonction de la place allouée à ceux-ci
- Monter le matériel dans la salle de réglage
  - Faire correctement passer les câblages
  - Faire apparaître des marquages pour aider les patients à s'installer au bon endroit dans la pièce
  - Ne pas gêner le passage de patients et ingénieurs dans la pièce
- Développer un logiciel capable de piloter des enceintes :
  - Pouvoir définir quelle(s) enceinte(s) est (sont) allumée(s) à un moment précis
  - Pouvoir jouer sur certaines enceintes des pistes audio prédéfinies
  - Pouvoir facilement rajouter des pistes audio ultérieurement
  - Pouvoir faire fonctionner le logiciel sous n'importe quel ordinateur possédant Windows 10
  - Avoir accès à ce logiciel avec le moins d'installation à effectuer
  - Créer une interface graphique simple d'utilisation et intuitive
- Constituer une banque de sons représentative de ceux entendus dans la vie courante
  - Calibrer chaque son à un même niveau sonore
  - Fournir des références de niveaux sonores aux ingénieurs de recherche

Les outils développés seront manipulés à présent par des ingénieurs de recherche en implants cochléaires.

## 2 Sonorisation

### 2.1 Haut-parleurs

Nous avons utilisé quatre enceintes identiques, afin de mieux reproduire la spatialité du son. Le modèle est le suivant : **Presonus Eris E5 XT**. Ce sont des enceintes de monitoring, ce qui signifie qu'elles possèdent la réponse en fréquence la plus homogène possible (expliquant leur utilisation courante pour diverses applications de production audio professionnelles). Elles sont actives, ce qui signifie que chaque enceinte intègre un module d'amplification totalement adapté au subwoofer, woofer, medium ou tweeter : cela permet de ne pas avoir à amplifier soi-même le son, et ainsi risquer de choisir un amplificateur ayant une puissance non-adaptée. Aussi, le potentiomètre de contrôle du gain situé à l'arrière des enceintes nous a été utile lors de leur installation (voir la sous-section 2.4). Vous pouvez retrouver la fiche technique des enceintes en annexes (section 6, figure 14).

FIGURE 1 – *Vues avant et arrière de l'enceinte Presonus Eris E5 XT*



Lors de l'installation des enceintes, nous nous sommes rendu compte que l'une d'elles (numérotée comme étant la 2<sup>e</sup>), était légèrement défaillante. En effet, lorsque l'on rallume l'enceinte après l'avoir éteinte récemment, un « boom » peut retentir. Cependant, ce problème n'est pas rencontré lorsque l'on allume l'enceinte après quelques dizaines de secondes de repos. Cela implique donc très probablement qu'un condensateur ne fonctionne pas comme il devrait fonctionner, et qu'il se crée une légère surtension tant que celui-ci n'a pas eu le temps de se décharger.

Cela n'est pas grave, puisque le bruit n'est pas très fort, et cela ne risque donc pas d'endommager les membranes de l'enceinte. Une solution pour stopper ce problème pourrait être de renvoyer l'enceinte chez le fabricant, mais il a été choisi de laisser les choses comme telles, puisque l'on est rarement amené à allumer et éteindre plusieurs fois les enceintes dans la même minute, et que le seul problème observé est une gêne fugace de l'ingénieur et du patient.

## 2.2 Carte son

Pour contrôler les enceintes depuis l'ordinateur, il faut une carte son pour servir d'interface. Celle-ci, reliée à l'ordinateur, permet de faire partir un câble vers chacune des enceintes. Nous avons choisi d'utiliser une **Focusrite Scarlett 4i4**, pour son prix, le nombre de *line outputs*, sa taille réduite, et comme vous allez le voir plus tard (sous-section 3.1.2), pour le logiciel propriétaire fourni avec la carte.

FIGURE 2 – Vues avant et arrière de la carte Focusrite Scarlett 4i4



Les quatre *line outputs* sont des prise femelles jack 6.35 mm et il faudra les brancher aux enceintes. Elles sont numérotées de la même manière que sont numérotées les enceintes. En bas à gauche de l'arrière de la carte, vous pouvez voir une prise femelle USB Type-C : cela sert à relier l'ordinateur à la carte son. Le gros potentiomètre « monitor » aurait pu servir à contrôler le volume d'enceintes, mais il contrôle seulement les enceintes des *outputs* 1 et 2 (les deux autres étant à un niveau maximal). On ne se sert donc pas de ce potentiomètre, et on peut contrôler le volume des enceintes depuis l'ordinateur. S'il apparaissait que l'ingénieur souhaite contrôler physiquement les enceintes, il faudrait passer par un amplificateur externe.

## 2.3 Câblages

Comme vous pouvez le voir sur la figure 1, les enceintes présentent 3 types d'entrées femelles : XLR ; TRS ; RCA Unbalanced (aussi appelée « prise cinch »). Cette dernière n'est pas symétrique (comme l'indique son nom), or plus la longueur du câble est grande, plus il est important d'utiliser un câble symétrique afin d'annuler des potentielles interférences qui pourrait parasiter le signal. Il faut choisir d'utiliser un câble TRS, ou un câble XLR. Le câble TRS est un jack 6.35 mm avec un contact en 3 points, reconnaissable par la présence sur la fiche mâle de deux bagues noires (à ne pas confondre avec un jack TS, qui possède une seule bague, et qui est asymétrique). Il est utilisé dans notre cas pour « une liaison symétrique monophonique comme en XLR (Pointe : Point chaud ; Anneau : Point froid ; Manchon : masse) » (source : Wikipédia). Comme les longueurs des fiches mâles XLR et TRS sont sensiblement les mêmes, nous avons choisi arbitrairement de prendre 4 câbles jack TRS 6.35 mm avec des fiches mâles de chaque côté.

Il était fourni avec les enceintes des câbles d'alimentation sur secteur pour des prises « suisses » (norme SEV 1011 : socle T13 et fiche T12, aussi appelée prise de type J), mais il a fallu acheter de câbles plus longs. Les enceintes possédant des socles de type C14 mâles, il faut une fiche femelle de type C13. Vous pouvez retrouver le type de câble qu'il faut en figure 3. Nous avons aussi acheté un câble USB Type-C mâle vers USB Type-A mâle plus long que celui fourni avec la carte son, afin de relier l'ordinateur à la carte.

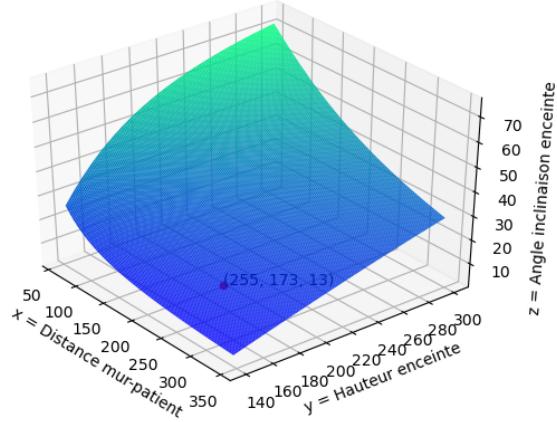
FIGURE 3 – *Exemple de câble utilisé, C13 vers T12*

## 2.4 Installation du matériel

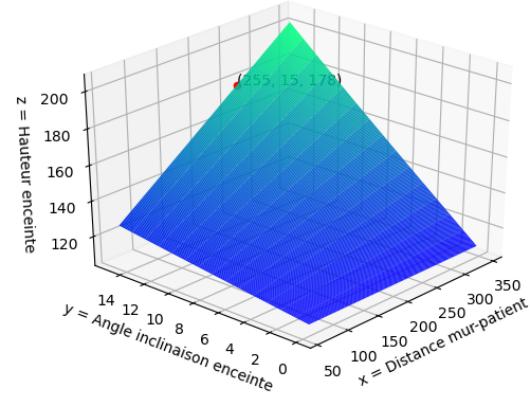
L'un des plus importants challenges de ce projet était de réfléchir constamment à l'agencement final des éléments matériels dans la pièce. En effet, puisque des patients circulent dans la pièce, il faut à tout prix veiller à ce que les câbles ne gênent pas, et à ce que le matériel ne puisse pas être bousculé. Alors qu'une configuration acoustique optimale aurait été possible en disposant à plat les enceintes telles que les tweeters se situent au niveau de l'oreille du patient, nous avons choisi de surélever les enceintes pour diverses raisons décrites ci-après. Il a donc fallu les incliner vers le bas, et pour cela nous avons choisi des accroches murales : des **Vogel's VLB 200 Speaker Wall Mounts**, s'inclinant jusqu'à 15° verticalement. Comme vous pouvez le voir dans la fiche technique des enceintes en annexes (section 6), le son est perçu de manière relativement homogène en-dessous de 30° d'inclinaison verticale de l'enceinte par rapport à l'axe horizontal au centre du haut-parleur (la dispersion verticale est de 60°, soit 30° au-dessus et 30° en-dessous de l'axe).

Afin de mieux se rendre compte de la faisabilité d'une telle installation, des simulation en Python ont été effectuées. Il s'agissait de calculer ici l'angle d'inclinaison de l'enceinte nécessaire pour le mur à l'avant et à l'arrière du patient. Nous avons ainsi choisi de placer la tête du patient à un point fixe : à 90 cm du mur arrière et à 255 cm du mur avant (pour un pièce de largeur 345 cm). Il s'agissait de la distance la plus éloignée du mur arrière possible - bien que non optimale - car il fallait à tout pris garder un grand espace à l'avant pour que l'ingénieur puisse travailler correctement (passer, s'asseoir,...). Nous avons aussi choisi la hauteur des enceintes : il fallait que le bas des enceintes arrières soit au minimum situé à 180 cm du sol (soit à 203 cm pour le tweeter), et à 150 cm du sol pour l'avant (soit 173 cm pour le tweeter). Cela était lié aux spécifications du projet : les patients devaient pouvoir circuler sans risquer de se cogner (enceintes arrière) et les enceintes avant devaient ne pas gêner la pose d'objets sur le meuble situé à l'avant !

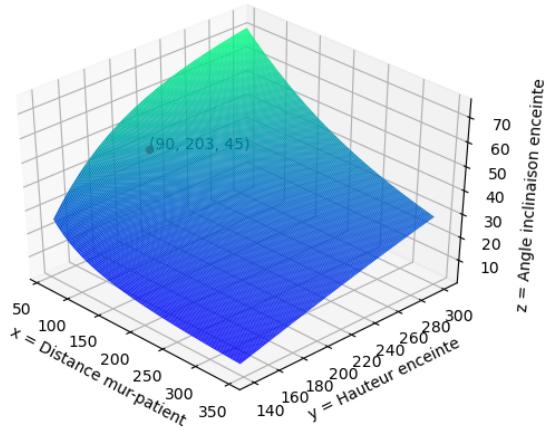
En fixant ces paramètres, nous avons pu calculer les angles d'inclinaisons nécessaires pour que les tweeters soient dirigés vers les oreilles du patient (estimées à environ 110 cm de hauteur du sol en position assise). En figure 4a, vous pouvez voir que l'angle d'inclinaison nécessaire est inférieur à la limite d'inclinaison des accroches de haut-parleurs (13° contre une limite technique de 15°). Pour les enceintes arrière, vous pouvez voir que l'angle nécessaire est supérieur à 15° : on ne pourra pas avoir les enceintes dans l'axe du tweeter... Cependant, nous avons vu que nous pouvions « rajouter » 30° dans lesquels le son est selon la fiche technique approximativement du même niveau sonore que dans l'axe. Cela signifie qu'en plaçant les enceintes d'une telle sorte, nous seront à la limite d'une configuration pseudo-optimale qui nous contentera. Il convient tout-de-même de préciser que nous avons estimé que les tweeters étaient collés aux murs, ce qui n'est pas juste : nous devrions ajouter quelques centimètres de distances murs-patient (ou plutôt tweeters-patient, donc), non décisifs dans nos choix de placements d'enceintes.



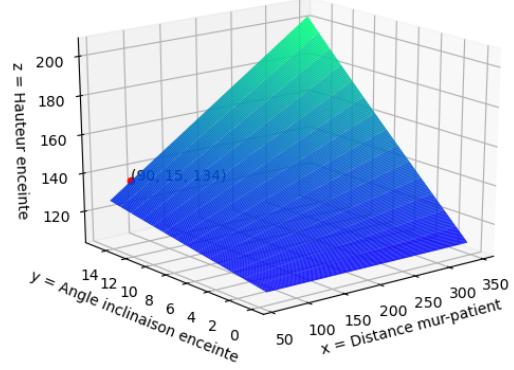
(a) Angle d'inclinaison en fonction de la distance mur-patient et de la hauteur du tweeter, point fixe pour  $x = 255$  ;  $y = 173 \Rightarrow z = 13$



(b) Hauteur du tweeter en fonction de la distance mur-patient et de l'angle d'inclinaison, point fixe pour  $x = 255$  ;  $y = 15 \Rightarrow z = 178$



(c) Angle d'inclinaison en fonction de la distance mur-patient et de la hauteur du tweeter, point fixe pour  $x = 90$  ;  $y = 203 \Rightarrow z = 45$



(d) Hauteur du tweeter en fonction de la distance mur-patient et de l'angle d'inclinaison, point fixe pour  $x = 90$  ;  $y = 15 \Rightarrow z = 134$

FIGURE 4 – Ensemble des simulations avec Python. En annexes (section 6), (a) et (c) sont à retrouver en Code 2, (b) et (d) à retrouver en Code 3

Vous pouvez voir en figure 5 le montage final. Comme les enceinte arrières sont plus proches du patient, celui-ci les entend plus fort qu'à l'avant. Nous avons donc réglé empiriquement les enceintes arrière en s'asseyant à la place du patient : il a fallu baisser les gains des enceintes №3 et №4 grâce au bouton à l'arrière de celles-ci, en comparant la mesure du niveau sonore à la place du patient en jouant les enceintes arrière à celle en jouant les enceintes avant. Il sera toujours possible modifier cet ajustement selon les besoins.

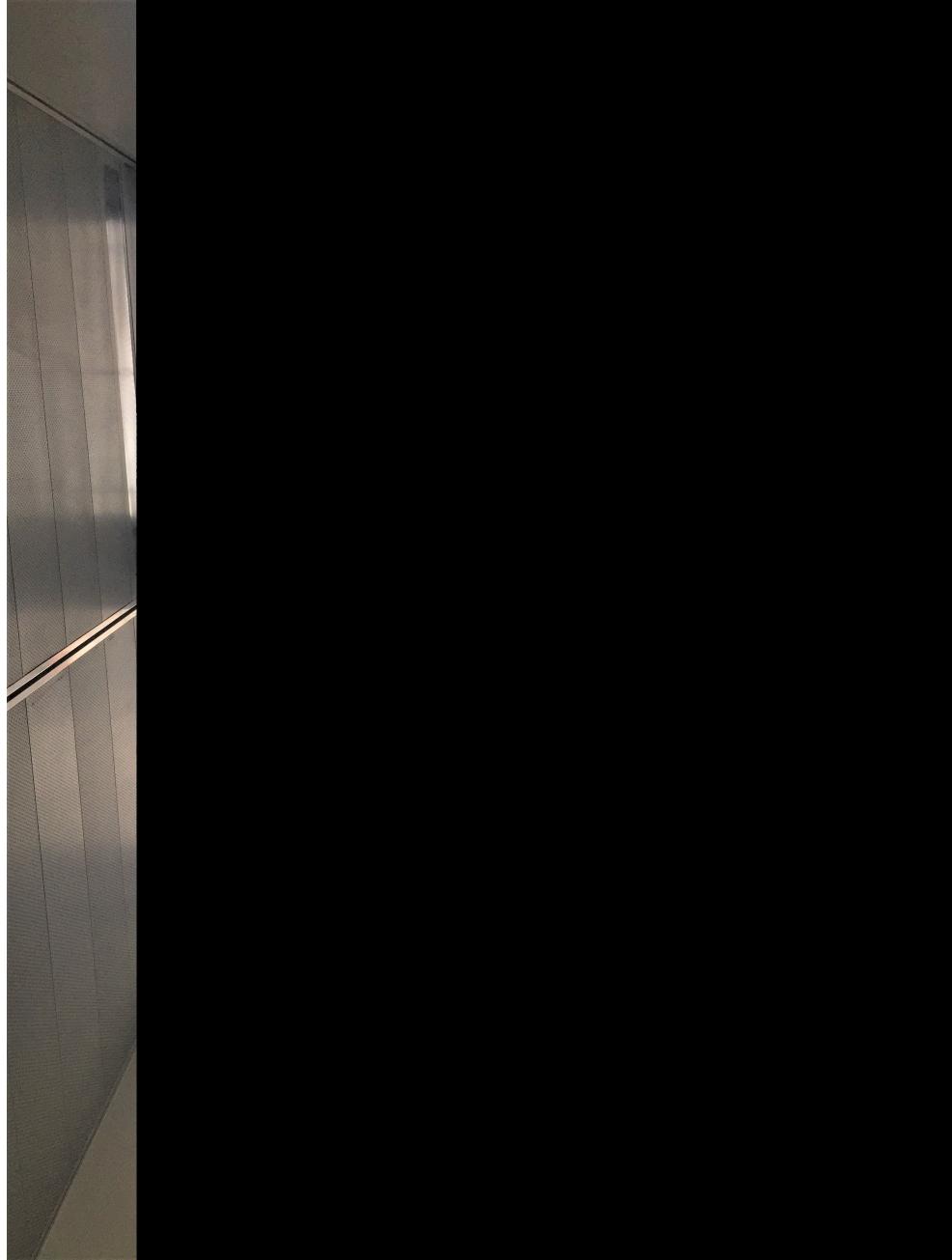


FIGURE 5 – Montage du système de sonorisation final. Les enceintes sont numérotées de 1 à 4 afin de mieux s'y retrouver, la croix verte représente l'emplacement du patient, et la croix jaune l'emplacement de l'ingénieur.

## 3 Logiciels

Le petit logiciel Merspeakers a été développé en Python notamment grâce à la bibliothèque Tkinter d'interface graphique, fourni dans la librairie standard, et au module mixer de la bibliothèque Pygame fournie à part, permettant un contrôle avancé de canaux sonores. La bibliothèque Cx\_Freeze a servie à créer fichier exécutable à partir de script Python. Comme vous allez le voir, nous sommes aussi amené à utiliser un autre logiciel, propriétaire : Focusrite Control.

### 3.1 Fonctionnalités des logiciels

#### 3.1.1 Utilisation du logiciel Merspeakers

*Merspeakers.exe* est une application permettant de visualiser des boutons prenant le nom de fichiers sons appartenant à un même dossier, que nous décrirons par la suite (paragraphe 3.2.1 et section 4). Vous pouvez voir en figure 6 ce à quoi ressemble l'application au démarrage. Il s'agit de la version finale de l'application en 2020, mais une version  $\alpha$  avait été développée et avait une tout autre apparence (voir la figure 15 en annexes, section 6) : la concertation de membres du CURIC a permis d'imaginer les améliorations à effectuer sur le logiciel final. La majeure modification consiste en la possibilité de créer deux listes au lieu d'une, l'une dynamique en temps réel, alors que la version  $\alpha$  comprenait seulement une liste figée. Cela est détaillé ci-après.

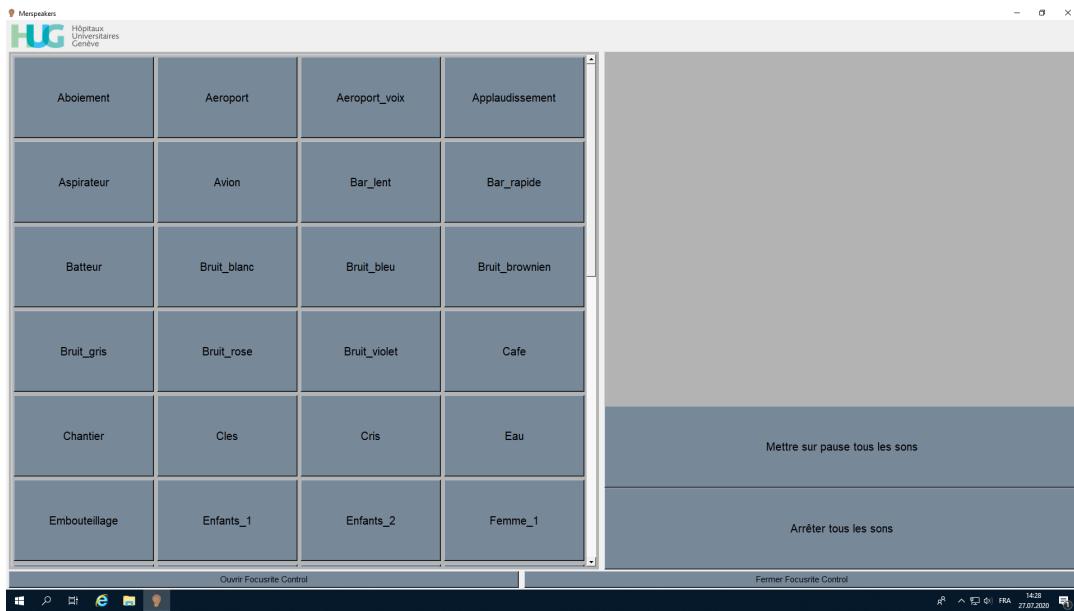


FIGURE 6 – Application Merspeakers au démarrage

Lorsque vous cliquez sur un des boutons dans la liste de gauche, cela initialise un bouton correspondant en haut à droite de l'écran, accompagné d'une échelle de volume et d'une échelle de balance gauche-droite. Vous pouvez lancer un nombre illimité de sons en même temps, mais la taille de l'écran que vous utilisez déterminera le nombre de boutons affichés en haut à droite (les autres seront cachés) : cela forme toujours trois colonnes de X lignes. Par exemple, sur l'écran utilisé en 2020 en salle de réglage, on peut afficher trois colonnes de quatre lignes, soit douze boutons simultanément (voir la figure 7), alors que sur un ordinateur portable de 14 pouces, trois lignes (soit neuf boutons) sont visibles. La liste de droite est dynamique en temps réel : cela signifie que la liste change d'apparence à chaque action de souris.



FIGURE 7 – Application Merspeakers lorsque l'on ajoute 12 sons, sur un écran relativement grand

Un élément pratique de l'application dans sa version finale est la possibilité de gérer les pauses : vous pouvez cliquer sur un bouton en haut à droite pour le mettre en pause, ou bien mettre tous les boutons actifs en pause grâce au bouton en bas à droite. Cela permet de sélectionner quelques sons dans la liste de gauche à faire apparaître dans la liste de droite, et après de jouer uniquement avec les boutons de droite (voir la figure 8). Pour enlever un son de la liste de droite, il suffit de cliquer sur le bouton correspondant dans la liste de gauche (double clic si le son est en pause). Vous pouvez aussi utiliser le bouton « Arrêter tous les sons » en bas à droite.

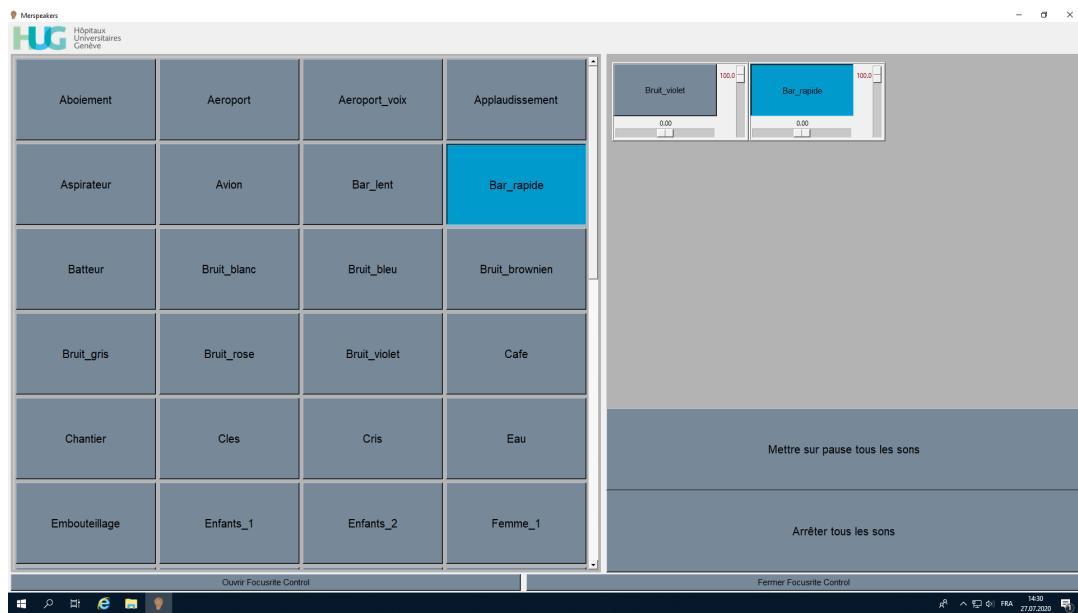


FIGURE 8 – Application Merspeakers lorsque l'on met un son sur pause et que l'on joue un autre

Vous pouvez régler le volume de chaque son sur son échelle verticale, et la balance gauche-droite sur son échelle horizontale. Les paramètres de chaque son sont conservés quand il est mis en pause, et remis à 0 lorsque le son est arrêté (voir la figure 9).

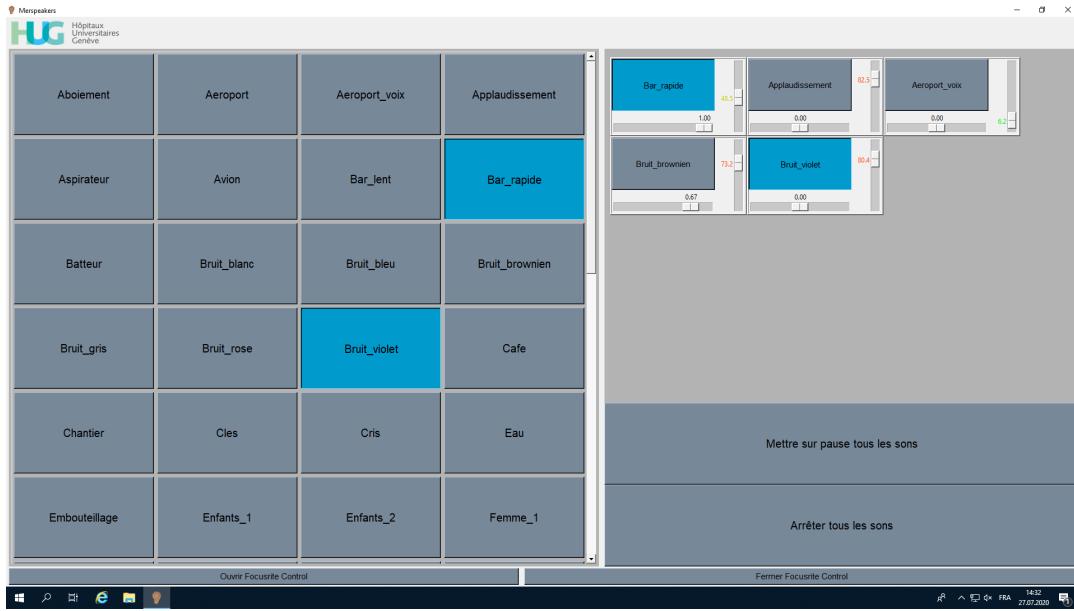


FIGURE 9 – Application Merspeakers lorsque l'on modifie les réglages de plusieurs sons

Les deux autres boutons en bas de l'écran servent à ouvrir et fermer le logiciel *Focusrite Control*. Fermer ce logiciel depuis *Merspeakers* prend moins de temps que de le fermer normalement.

### 3.1.2 Utilisation du logiciel Focusrite Control

*Focusrite Control* est un logiciel propriétaire fourni par la marque Focusrite afin de contrôler certaines cartes son de la marque, comme celle utilisée ici (voir la sous-section 2.2). Il faut bien entendu brancher une carte son de la sorte afin que le logiciel fonctionne.

L'application fonctionne en colonne : vous pouvez visualiser les quatre enceintes branchées et couper/allumer le son d'une ou plusieurs enceintes grâce au bouton « mute », ainsi qu'aux échelles de volumes (voir la figure 10). En haut à gauche, vous pouvez appuyer sur « file » puis sur « load snapshot » pour ouvrir un fichier de configuration qui permettra de contrôler les enceintes par paires. Dans le dossier global *Merspeakers* se trouvent deux fichiers de configuration *gauche\_droite.ff* et *avant\_arrière.ff*.

Pour mieux comprendre ce qui va suivre, référez-vous à la figure 11. En chargeant le premier fichier, vous identifierez sur le même « playback » les enceintes selon la gauche et la droite. Concrètement, l'enceinte avant-gauche (Nº1) et l'enceinte arrière-gauche (Nº3) sont identifiées comme appartenant au canal gauche de la stéréo, tandis que l'enceinte avant-droite (Nº2) et l'enceinte arrière-droite (Nº4) seront identifiées comme appartenant au canal droit de la stéréo. Dans l'application *Merspeakers*, cela signifiera que l'échelle de balance gauche-droite fonctionnera normalement. L'autre fichier de configuration propose une balance avant-arrière : cela signifie que l'enceinte avant-gauche (Nº1) et l'enceinte avant-droite (Nº2) sont identifiées comme appartenant au canal gauche de la stéréo, tandis que l'enceinte arrière-gauche (Nº3) et l'enceinte arrière-droite (Nº4) sont identifiées comme appartenant au canal droit de la stéréo. Ainsi, dans l'application *Merspeakers*,

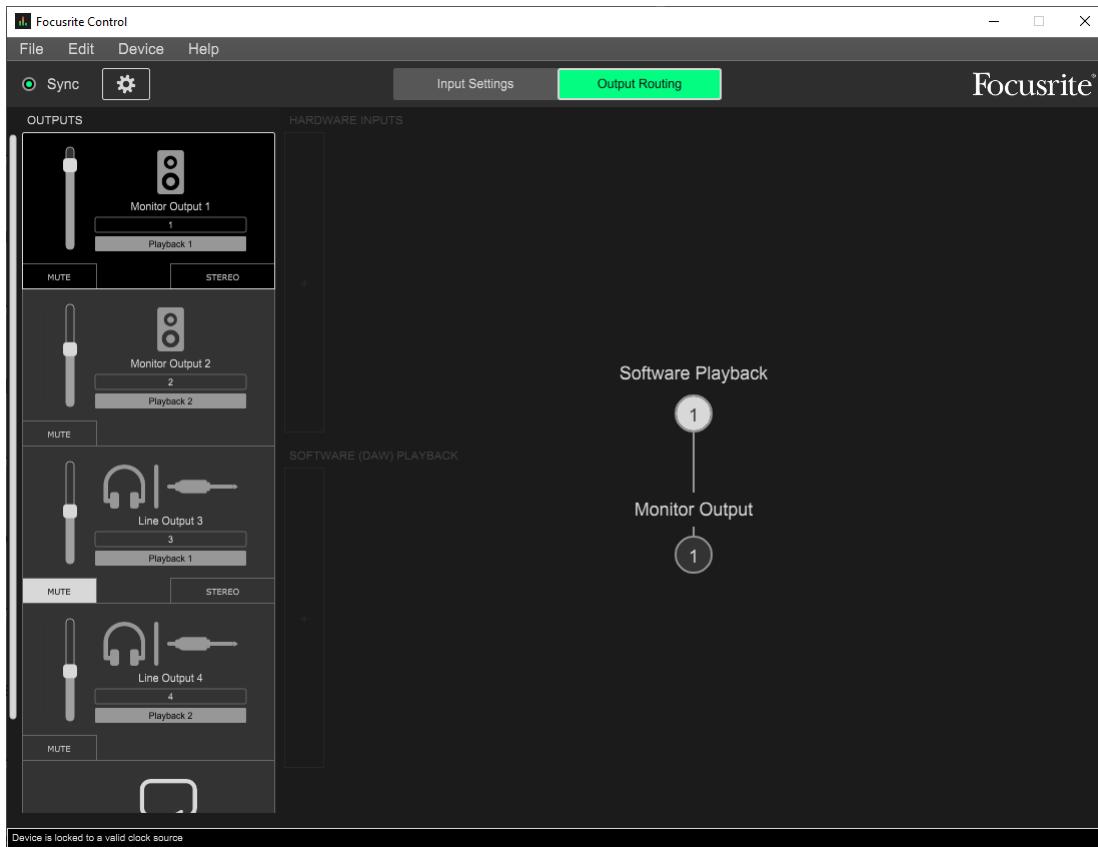


FIGURE 10 – *Focusrite Control* en réglant les volumes des enceintes (Nº3 muette)

régler l'échelle de balance sur la gauche jouera les enceintes avant (Nº1 et Nº2), et régler l'échelle de balance sur la droite jouera les enceintes arrière (Nº3 et Nº4).

En jouant avec les deux playbacks disponibles (1 : canal stéréo gauche et 2 : canal stéréo droite), vous pouvez créer n'importe quelle configuration. On peut alors imaginer par exemple demander à *Merspeakers* que lorsque l'échelle de balance est réglée en position gauche, l'enceinte avant-gauche (Nº1) joue seule, et lorsque l'échelle de balance est réglée en position droite, les trois autres enceintes jouent (on règle alors sur un playback l'enceinte Nº1 et sur un l'autre les autres enceintes). On peut enregistrer une configuration en appuyant en haut à gauche de l'écran sur « file » puis « save snapshot ».

Vous remarquerez probablement que quatre playbacks sont disponibles à l'écran au lieu des deux premiers que l'on utilise (figure 12). Puisque nous avons quatre enceintes, il serait envisageable de jouer un son différent pour chaque enceinte, en utilisant ces quatre playbacks. Cependant, cela impliquerait de pouvoir jouer un son à quatre canaux, ce qui n'est pas commun ! C'est ce que l'on appelle le son multicanal (ou « surround » en anglais), et cela n'est pas gérable par l'application *Merspeakers*. Un logiciel de mixage de son en 2D permettrait un tel usage, mais ce n'était pas l'objet premier de *Merspeakers*.



FIGURE 11 – *Focusrite Control* avec le fichier de configuration « gauche\_droite.ff » (à gauche) et avec « avant\_arrière.ff » (à droite)

## 3.2 Installation des logiciels

### 3.2.1 Installation de Merspeakers

Lorsque vous allez ouvrir le dossier *Merspeaker*, vous pourrez ouvrir directement *Merspeakers.exe*. Celle-ci lira directement dans le fichier *sounds* pour former une liste de sons utilisables dans l’application. Il est alors tout à fait conseillé d’enlever et rajouter des sons à votre convenance (cf. section 4 pour connaître la méthode d’ajout de sons). Un dossier *Other sounds* est aussi fourni : vous pouvez ranger ici tous les sons que vous souhaitez conserver sans pour autant les afficher à la prochaine ouverture de l’application. Il faut par ailleurs (ré)ouvrir l’application pour appliquer les changements effectués sur le dossier *sounds* en cours d’utilisation de celle-ci.

Pour faciliter l’accès à l’application, deux manipulations peuvent être effectuées. Premièrement, vous pouvez déplacer le raccourci de *Merspeakers* qui est dans le dossier principal vers n’importe quel autre dossier (le bureau par exemple). Vous pouvez créer une infinité de copies de ce raccourci. Créer une simple copie de l’application ne fonctionnera cependant pas sans copier l’ensemble des fichiers du dossier *Merspeakers*.

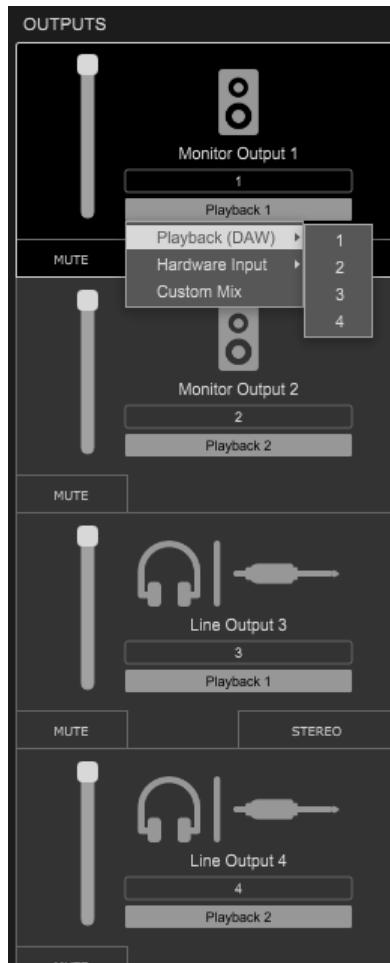


FIGURE 12 – Réglage des playbacks

La seconde manipulation consiste dans Windows 10 à créer un raccourci vers le menu Démarrer, permettant à la fois de le ranger à la lettre « M » et à la fois d'accéder à l'application en effectuant une recherche rapide.

Pour rajouter le raccourci à tous les utilisateurs :

**Windows + R** ⇒ “C :\ProgramData\Microsoft\Windows\Start Menu\Programs” ⇒ Créez dans ce dossier une copie du raccourci de *Merspeakers* fourni dans le dossier principal de l'application.

### 3.2.2 Installation de Focusrite Control

Pour installer Focusrite Control, il faut se rendre sur le site de Focusrite, qui fournit avec la carte son le logiciel gratuitement : le lien en 2020 est « [focusrite.com/fr/get-started](https://focusrite.com/fr/get-started) », et la version du logiciel est la 3.6.0. Il faut ensuite sélectionner le modèle (*Scarlett 3rd Generation* ⇒ *Scarlett 4i4*), cliquer sur « Let's get you started » puis sur « JUST GIVE ME MY ESSENTIAL SOFTWARE » en bas de la page, et télécharger le logiciel.

Vous pouvez alors lancer l'installateur. Lorsque celui-ci vous demande de spécifier l'emplacement du dossier de téléchargement, copiez-le. Collez-le alors dans le fichier *path.json* (qui peut s'ouvrir dans un simple bloc-note) qui décrit l'emplacement et le nom du logiciel Focusrite Control, afin de l'ouvrir depuis *Merspeakers*,

qui lit ici le chemin d'accès (il doit y avoir accolés dans le fichier *path.json* l'emplacement du fichier ainsi que le nom du logiciel : « Focusrite Control.exe »). Par exemple, il est courant de remplir le fichier de la façon suivante :

```
1   {
2     "focusrite_control" : "C:/Program Files/Focusrite/Focusrite Control/Focusrite Control.exe"
3 }
```

### 3.3 Accessibilité des codes sources et création de nouveaux fichiers exécutables

Le code source est fourni avec l'application dans un fichier Python sous le nom de *Merspeaker.py* (voir le Code 2 en annexes, section 6). Vous pouvez modifier ce fichier sans modifier l'application *Merspeakers.exe* (qui est indépendante), mais il faudra effectuer plusieurs démarches pour être capable d'exécuter un script Python, de le modifier, et de générer un nouveau fichier exécutable.

Il est toujours conseillé de copier l'ensemble du dossier du projet Merspeakers vers un autre dossier (le nom du dossier global et son emplacement ne sont pas important pour le bon fonctionnement des logiciels, seuls le bon agencement des dossiers et fichiers à l'intérieur de celui-ci importent).

#### 3.3.1 Installation de Python

Python est un langage de programmation en constante évolution, impliquant parfois des changements d'une versions à une autre, et dans de très rares cas (lors de mises à jour majeures), des problèmes de compatibilité entre les versions. En 2020, l'application *Merspeakers.exe* fut générée à partir de *Merspeakers.py*, développé avec la version de Python 3.8.5. Il est conseillé d'utiliser une version récente de Python, mais il se peut que cela ne permette pas de lire un code obsolète. Ainsi, si vous rencontrez des erreurs à l'exécution du script une fois toutes les étapes décrites ci-dessous effectuées, essayer de réinstaller Python sous la version 3.8.5. Vous pouvez par ailleurs installer plusieurs versions de Python sur un même machine : il existe de nombreux tutoriaux sur internet pour cela, notamment grâce à l'outil Virtualenv. Dans tous les cas, veuillez vérifier que la version de Python que vous souhaitez installer soit bien compatible avec Cx\_Freeze (documentation : (« [cx-freeze.readthedocs.io/en/latest/index.html](https://cx-freeze.readthedocs.io/en/latest/index.html) »)

1. Rendez-vous sur [python.org](https://www.python.org) pour télécharger Python (la dernière version ou non, comme spécifié précédemment).
2. Veillez à télécharger la version 64 bits (qui ne se télécharge pas toujours par défaut, il se peut que cela soit la version 32 bits (x86) qui se télécharge). Par exemple, vous pouvez trouver pour Python 3.8.5 l'installateur suivant : « Windows x86-64 executable installer ».
3. Lancer l'installation (possibilité de l'installer sur l'ensemble des utilisateurs d'un ordinateur, vérifiez ce que vous cochez dans l'installateur).
4. Une fois Python installé, ouvrez l'invite de commande (**cmd + R** ⇒ “cmd”) pour installer les bibliothèques utiles. Pour les ordinateurs HUG, il peut être nécessaire de passer par le proxy ou de contourner cela en utilisant un VPN :

- Avec un VPN actif ou sur un ordinateur personnel :

```
py -m pip install pygame cx_Freeze
```

- Avec les identifiants du proxy :

```
py -m pip install --proxy=https://[username:password@]proxyserver:port pygame cx_Freeze
```

5. Si tout s'est bien passé et dans la grande majorité des cas, Python est installé avec les bibliothèques nécessaires. Il se peut toutefois que l'installation des bibliothèques ne se soit pas correctement effectué. Cependant, une simple recherche sur internet du message d'erreur permet normalement de comprendre l'origine du problème (de nombreux internautes auront déjà rencontré votre problème).

Afin de programmer en Python, il est recommandable d'utiliser un IDE (pour *integrated development environment*) mais ce n'est en aucun cas obligatoire, sachant que les IDE sont parfois difficiles à prendre en main. Je peux cependant conseiller d'utiliser Visual Studio Code avec le plug-in Python.

### 3.3.2 Du script Python au fichier exécutable

Pour permettre de distribuer facilement un logiciel écrit en Python, il est intéressant de le transformer en fichier exécutable. C'est à cela que sert la bibliothèque Cx\_Freeze et le fichier *setup.py*, donné en Code 1.

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  import sys
5  import os
6  from cx_Freeze import setup, Executable
7
8  script_dir = os.path.dirname(__file__)
9
10 base = None
11 if sys.platform == "win32":
12     base = "Win32GUI"
13
14 build_exe_options = {"packages": ["os", "subprocess", "json", "PIL", "tkinter", "pygame", "sys", "ctypes"],
15                      "excludes": ["asyncio", "cffi", "concurrent", "curses", "distutils", "email", "html",
16                                   "http", "lib2to3", "multiprocessing", "numpy", "pkg_ressources", "psutil",
17                                   "pycparser", "pydoc_data", "setuptools", "unittest", "win32com", "xmlrpc"]}
18
19 setup(
20     name = "Merspeakers",
21     version = "0.1",
22     options = {"build_exe": build_exe_options},
23     description = "Génération de bruits ambients avec accès à Focusrite Control",
24     executables = [Executable("Merspeakers.py", base=base, icon=os.path.join(script_dir, "logo", "icon.ico"))]
25 )

```

Code 1 – *setup.py*

Dans *setup.py*, pouvez rajouter l'ensemble des bibliothèques utilisées dans votre script dans la liste relative à la clé “packages” du dictionnaire *build\_exe\_options*. Cela permettra à Cx\_Freeze de créer les fichiers nécessaires au fonctionnement du fichier exécutable. Comme on fait référence dans *setup.py* à différents fichiers, il est important de garder telle quelle l'arborescence des fichiers dans le dossier du projet. En fonction de ce que vous avez modifié dans le script *Merspeakers.py*, il se peut que vous soyez obligé de modifier aussi des paramètres dans le script *setup.py*.

Quoi qu'il en soit, il vous faudra pour créer un nouveau fichier *Merspeakers.exe* :

1. S'assurer qu'il n'y ait plus l'ancien fichier exécutable *Merspeakers.exe* à l'intérieur du (nouveau) dossier dans lequel vous travaillez : c'est pour cela qu'il était conseillé de créer une copie du dossier d'origine !
2. Ouvrir l'invite de commande (**Shift + R** ⇒ “cmd”) et accéder au dossier dans lequel vous travaillez. Par exemple, si celui-ci s'appelle « *Merspeakers\_v2* » et s'il se situe dans le bureau, tapez :

```
cd Desktop/Merspeakers_v2
```

3. Lancer le fichier *setup.py* en lui passant le paramètre « build » :

```
py setup.py build
```

4. Quand la commande est terminée, ouvrir le dossier dans lequel vous travailler, et constater l'existence d'un dossier *build*.
5. Ouvrir ce dossier *build*, puis un second dossier dont le nom peut varier mais qui devrait ressembler à « *exe.win32-3.x* ».
6. Déplacer l'ensemble de ce que ce dernier dossier contient vers le dossier principal (que nous avions ici appelé « *Merspeakers\_v2* »). Vous devriez avoir un dossier « *lib* », un fichier *.dll*, et le fameux exécutable fraîchement créé, qui porte le nom spécifié dans *setup.py*. Le déplacement est indispensable dans la mesure où le fichier exécutable utilise les dossiers *sounds*, *logo*, *cursor* et le fichier *path.json*, selon une arborescence définie dans le code. Libre à vous de changer le code pour agencer différemment les fichiers.
7. Vous pouvez lancer le logiciel et recréer des raccourcis vers celui !

## 4 Ajout de sons

### 4.1 Banque de sons

Pour constituer une banque sonore, nous avons essayé de rassembler tous les sons plus ou moins gênant dans la vie courante, qu'ils soient aiguës ou graves, naturels ou artificiels. Nous avons ainsi pu constituer un ensemble de plus de 50 environnements sonores différents, auxquels s'ajoutent quelques sons non-choisis dans la liste principale d'utilisation mais qui sont conservés dans le dossier *Other sounds*. Comme le principe était d'être capable de superposer des sons, nous avons cherché à isoler au maximums ceux-ci. Par exemple, au lieu de choisir un son de ville avec énormément de trafic, il est plus judicieux de choisir un son d'ambiance de ville, et un son de trafic à côté !

Les sons choisis ont tous été téléchargés dans la mesure du possible en formats audios non-compressés, et avec une sonie perçue relativement stable dans le temps (afin de pouvoir par la suite estimer le niveau sonore au point où se situe le patient). La préférence vient aussi aux échantillons stéréos avec un enregistrement droit distinct de la gauche, mais il n'a pas toujours été facile de se constituer une banque sonore sur des sites gratuits en ligne : il serait peut-être nécessaire dans le futur de capter des sons par nous-même. L'avantage d'une piste droite différente de la gauche réside avant tout dans l'utilisation de l'échelle de balance gauche-droite de l'application *Merspeakers* : cela permet de mieux profiter de la spatialisation du son.

Quelques bruits sortent du lot. Premièrement, nous avons un trio de sons nommés respectivement « *cafet\_hug\_centre* », « *cafet\_hug\_gd* » et « *cafet\_hug\_ar* ». Ce sont des bruits issus d'une captation multicanale (5 canaux : un carré avec un point central) dans la cafétéria du 10<sup>e</sup> étage des H.U.G. effectuée auparavant. Le premier son correspond à la piste mono centrale : changer la balance gauche-droite sur *Merspeakers* déplacera donc le son sortant des enceintes sans le modifier pour autant. Le deuxième est différent : les pistes avant-gauche et arrière-gauche ont été attribuées au canal gauche de la stéréo, tandis que les pistes avant-droite et arrière-droite ont été attribuées au canal droit de la stéréo. C'est un échantillon à utiliser en configuration « *gauche\_droite* » dans *Focusrite Control* (voir la section 3.1.2). Le troisième est l'*alter ego* relatif à l'avant-arrière du deuxième : les pistes avant ont été disposées sur le canal gauche de la stéréo et les pistes arrières ont été disposées sur le canal droit de la stéréo. C'est cette fois-ci un échantillon à utiliser en configuration « *avant\_arriere* ». Ces mixages ont été effectués grâce au logiciel *Audacity*.

Nous avons en second temps récupéré l'ensemble des bruits colorés décrit par Wikipédia qui permettront de générer des environnements sonores probablement très difficiles à appréhender pour des personnes possédant un ou deux implant(s) cochléaire(s), mais qui seront très intéressants à étudier : nous pourrons tester les systèmes de contrôle de bruit actif pour différents spectres sonores. ([wikipedia.org/wiki/Colors\\_of\\_noise](https://wikipedia.org/wiki/Colors_of_noise))

Pour ajouter un son à la liste utilisée, il suffit de le déplacer vers le fichier *sounds*. Il faut veiller à ce que le nom de celui-ci ne soit pas trop long pour un bon affichage de l'application.

## 4.2 Calibration et conversion des échantillons sonores

Afin de permettre à l'ingénieur d'estimer le niveau sonore auquel est joué un son, il a fallu tout d'abord normaliser les niveaux des échantillons, puis fournir des références. Ainsi, nous sommes passé par le logiciel *Audacity* afin de normaliser l'amplitude des différents sons (voir la figure 13) : il s'agit d'un des effets disponibles dans le logiciel (« Effets » ⇒ « Normalisation de l'amplitude »). Il faut le paramétrier selon la sonie perçue, pour atteindre -23,0 LUFS (Loudness Unit Full Scale), correspondant à une norme de référence par rapport au maximum de niveau qu'un ordinateur peut gérer sans saturation. -23 LUFS est la référence suivie internationalement pour la télédiffusion. Sans prendre en compte les valeurs RMS de pression acoustique, cela permet d'estimer globalement que le niveau sera le même d'un échantillon à l'autre pour un même réglage du volume d'un système audio (ordinateur/carte-son/enceintes). Nous pouvons, une fois les modifications voulues sur le son effectuées (éventuellement par exemple un « Fade in/out » peut être ajouté), exporter l'échantillon vers un fichier WAV. Il faut cependant à tout pris veiller lors de l'exportation à ce que l'encodage soit en « 16 bits PCM signé », sans quoi *Merspeakers* ne parviendra pas à lire le son. L'exportation en un fichier WAV est indispensable.

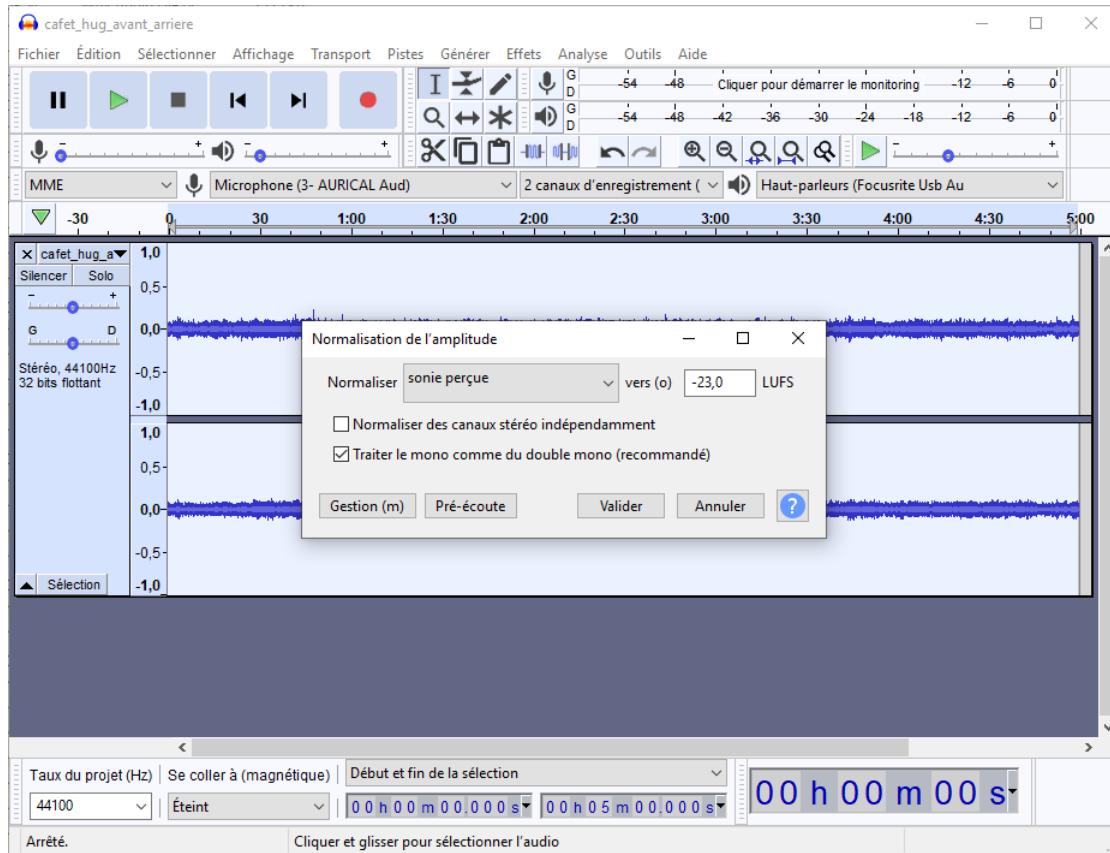


FIGURE 13 – Normalisation d'amplitude d'un échantillon sonore

Une fois tous les échantillons calibrés, il est possible de mesurer le niveau sonore au niveau de l'emplacement où se trouvera l'oreille du patient lorsqu'un échantillon est joué. En réglant un sonomètre (modèle Brüel &

Kjaer 2250) sur  $L_{eq}Z$ , on peut mesurer ce niveau. La lettre (Z) est relative à la pondération telle que la réponse en fréquence est plate entre 10Hz et 20kHz  $\pm 1.5$  dB.  $L_{eq}$  est un niveau sonore continu équivalent en décibels, équivalent à l'énergie sonore totale mesurée sur une période de temps donnée. La mesure au sonomètre n'a pas besoin d'être très précise dans notre cas ( sachant que le bruit ambiant dans la salle varie beaucoup d'un jour à l'autre), mais le mieux est d'éviter de gêner l'axe des enceintes et de veillez à légèrement incliner le sonomètre. Normalement, tous les niveaux doivent se situer autour de 65dB SPL  $\pm 2$ dB lorsque la carte son a son potentiomètre « monitor » au maximum, lorsque l'ordinateur a son volume global réglé sur 13, et lorsque l'on joue un seul son à 100% sur l'application *Merspeakers*. Des modifications peuvent alors être faites sur les échantillons, pour permettre d'observer un niveau sonore plus proche de 65 dB SPL : il est possible d'utiliser l'outil « Amplification » pour ajuster positivement ou négativement le niveau d'un son, sur *Audacity* (« Effets »  $\Rightarrow$  « Amplification »).

Nous avons effectué des mesures de niveaux en fonction du volume de l'ordinateur, pour différents réglages du volume d'un bruit blanc dans l'application *Merspeakers*. Vous pouvez vous référer à la figure 16 en annexes (section 6).

## 5 Conclusion et perspectives

Nous avons finalement pu en un mois atteindre un avancement bien satisfaisant pour ce projet *Merspeakers*. Nous avons développé un logiciel fonctionnant en parallèle de celui fourni avec la carte son choisie, ce qui offre de nombreuses alternatives d'utilisation.

L'une des améliorations qui serait envisageable dans un autre temps serait le développement d'un outil de pilotage en 2D des 4 enceintes : on pourrait imaginer déplacer un point dans un rectangle (dont les sommets représenteraient les enceintes) et il serait alors possible de ne plus utiliser *Focusrite Control* pour associer des couples d'enceintes. En effet, on utiliserait un playback par enceinte au lieu d'un pour deux enceintes et on demanderait à python de jouer du son par quatre sorties au lieu des deux de la stéréo. Cependant, cela nécessiterait un bien plus long développement, car nous ne pourrions pas utiliser le module Pygame de Python - qui peut seulement jouer des pistes en stéréo - et il faudrait soit manipuler des modules bien plus complexes, soit créer son propre module : il serait nécessaire tout d'abord d'effectuer un état de l'art sur les solutions existantes pour gérer des formes simples d'« ambisonie » (technique à utiliser). De plus, nous devrions probablement utiliser d'autres drivers ainsi que d'autres protocoles permettant de faibles latences pour les équipements audio numériques (le protocole ASIO pourrait probablement être utilisé), que nous devrions être capable de contrôler intégralement avec Python.

Une seconde amélioration est d'ordre matérielle : il serait intéressant de proposer des solutions plus durables pour faire passer correctement les câbles dans la salle de réglage où est implanté le projet. Cela ne pouvait malheureusement pas être pris en compte dans le budget accordé cette année à *Merspeakers*.

Qu'allons-nous alors faire de ce projet ? Premièrement, il va falloir tester la cohérence du système dans la reproduction d'environnements sonores de la vie courante. Pour cela, il sera intéressant de demander le ressenti à un échantillon représentatif de patients. La difficulté de qualification réside en l'hétérogénéité des profils des patients (implants binauraux ou d'un seul côté, âge, aisance à exprimer le ressenti,...). On pourrait alors capter des sons dans différents environnements sonores (comme pour ceux de la cafétéria principale des H.U.G.) qui seraient joués avec *Merspeakers*, et emmener le patient directement au lieu de captation pour qu'il nous fasse part des divergences de ressentis.

Il n'est pas impossible qu'il faille changer des agencements dans la pièce comme la façon dont sont tournées les enceintes ou sont placées chaises et tables, si les résultats des tests sont nuancés. Il serait même envisageable de procéder à un traitement acoustique (à base d'absorbeurs et de diffuseurs) pour mieux reproduire certaines ambiances potentiellement gâchées par les mauvaises réflexions/absorptions du son dans

la salle. Une étude devrait en amont être réalisée, dans la continuité d'une étude non-aboutie sur le sujet en partenariat avec la Haute école du paysage, d'ingénierie et d'architecture de Genève (HEPIA).

Si le système est bien fiable pour reproduire des environnements sonores (après des modifications ou non), nous pourrons alors proposer un champ d'action assez étendu, en recherche comme en technique de réglage. En effet, nous pourrons effectuer des études comparatives sur les systèmes de contrôle de bruit actif suivant les marques et modèles d'implants, afin de mieux conseiller les patients et d'orienter potentiellement les achats. Nous pourrons aussi connaître l'influence des bruits ambients sur l'écoute du patient : il faudra par exemple comparer des réglages en conditions calmes par rapport à des réglages effectués dans des conditions plus difficiles pour le patient. Nous arriverons aussi mieux à déceler une plainte précise du patient. Par exemple, si quelqu'un n'arrive pas à expliquer à l'ingénieur quels types de bruits le gênent par dessus-tout, nous pourront directement les trouver en salle de réglage et agir en fonction : cela sera - nous l'espérons - vraiment bénéfique pour le confort des patients. Si les conditions le permettent, nous pourrions même montrer en temps réel aux patient l'influence d'un changement de réglage ou de mode de fonctionnement du processeur sur leur écoute. Cela devrait aider ceux-ci à engager la discussion, ayant plus de repères.

## 6 Annexes

1. Fiche technique des enceintes choisies (figure 14)
2. Capture de la version  $\alpha$  de *Merspeakers* (figure 15)
3. Graphique de mesures de niveaux sonores pour différents volumes d'ordinateur (figure 16)
4. Scripts pour les simulations de la figures 4 (Codes 3 et 4)
5. Ensemble du script de *Merspeakers.py* vous est fourni (Code 4)

FIGURE 14 – Fiche technique des enceintes Presonus Eris E5 XT



# ERIS E5 XT

2-Way Active Studio Monitor with Elliptical Wave Guide

## Key Features

- Elliptical Waveguide provides wide horizontal dispersion while maintaining a tightly focused vertical plane.
- Front-firing acoustic port for superior bass-frequency reproduction
- 48 Hz to 20 kHz
- 102 dB maximum continuous SPL
- 5.25-inch woven composite low-frequency transducer
- 1-inch (25 mm), ultra-low-mass, silk-dome, high-frequency transducer
- 80 watt, Class AB biamplification
- High- and Mid-range frequency controls plus Acoustic Space settings for accurate mixing contour

## Near-field Studio Monitors with a Wide Sweet Spot



The Eris E5 XT provides deeper lows and a wider, more controlled sweet spot thanks to a new EBM (Elliptical Boundary Modeled) wave guide designed by Hugh Sarvis of WorxAudio. The new waveguide offers a superior high-frequency response and wider 100-degree horizontal dispersion to create a broader sweet spot. It's ideal for group listening and collaboration—while the narrow

60° vertical dispersion reduces reflections from your desk. Silk-dome tweeters ensure snappy, accurate transient reproduction.

Set up your studio monitors for your room.

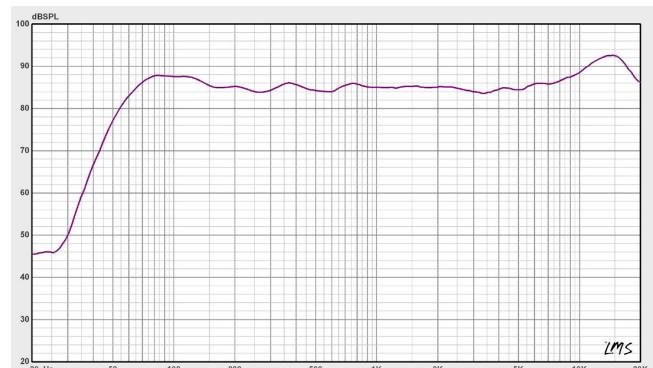
PreSonus knows modern recording studios can be anywhere from a spare bedroom to a multi-million dollar facility; that's why every studio monitor we make is equipped with controls that ensure the flattest response in your mix environment. Acoustic tuning controls allow you to configure the Eris E5 XT perfectly for your room—you get Low-cut, Mid, and High controls, as well as 3-way acoustic space tuning that allows you to easily compensate for the sonic consequences of speaker placement against a wall or in a corner.

Accuracy defined.

Suitable for both home recording studios and professional mix engineering, the Eris XT studio monitors combine sterling audio quality with the flexible tuning and connectivity you've come to expect from PreSonus. It's Eris, remastered.

## Technical Specifications

Frequency Response	48 Hz – 20 kHz
Peak SPL	102 dB (@ 1M)
Dispersion (HxV)	100° x 60°
LF Amplifier Power	45W Class AB
HF Amplifier Power	35W Class AB
LF Driver	5.25" Woven Composite
HF Driver	1" Silk Dome
Inputs (1 ea.)	Balanced XLR, Balanced 1/4" TRS, Unbalanced RCA
Controls	Level, HF, MF, Low Cut, Acoustic Space
Dimensions (WxDxH)	8" x 9" x 11" (203.2 x 228.6 x 279.4 mm)
Weight	11 lbs (5 kg)



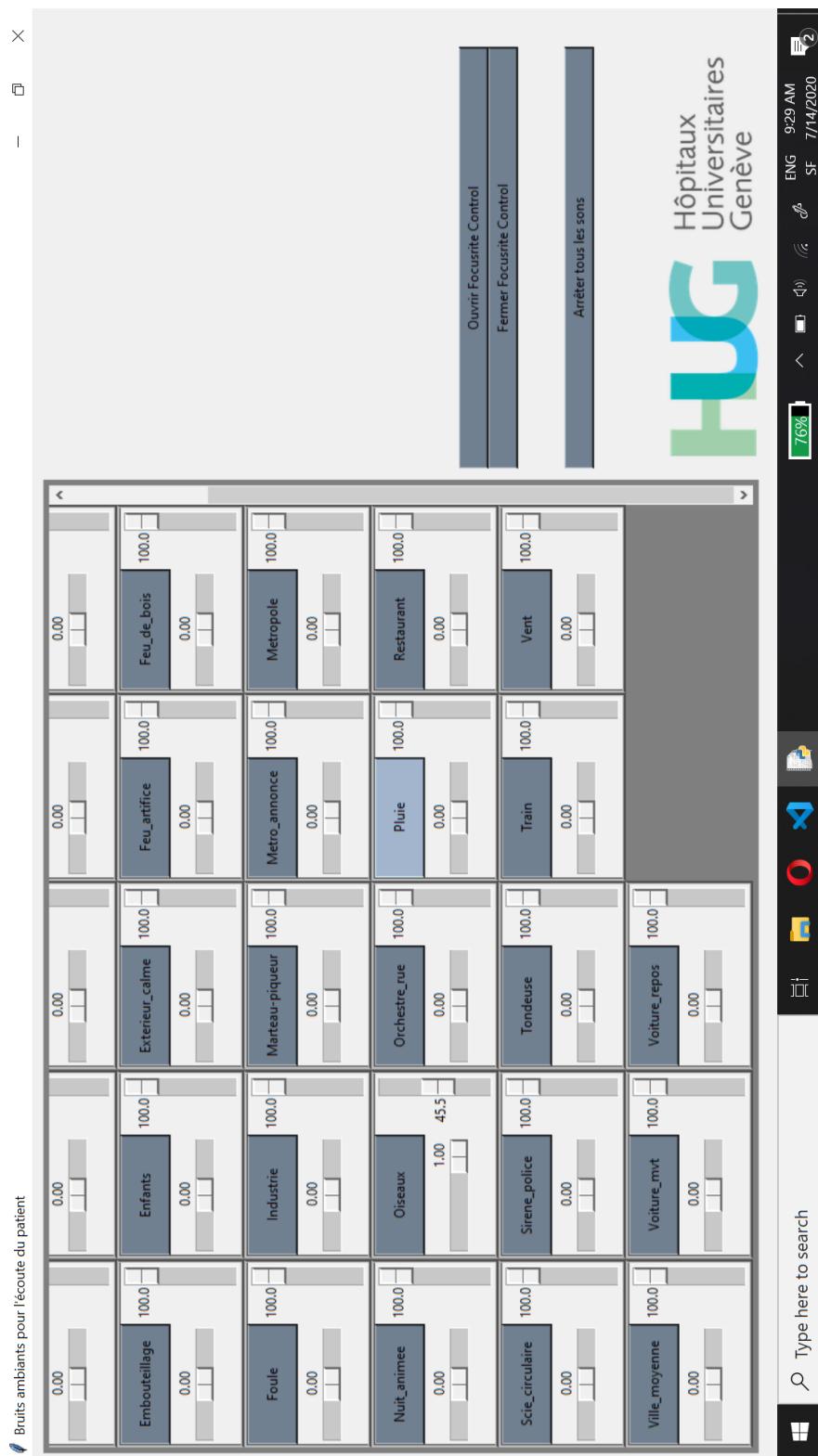


FIGURE 15 – Merspeakers (version  $\alpha$ ) avec une liste statique : on peut activer/désactiver chaque son.

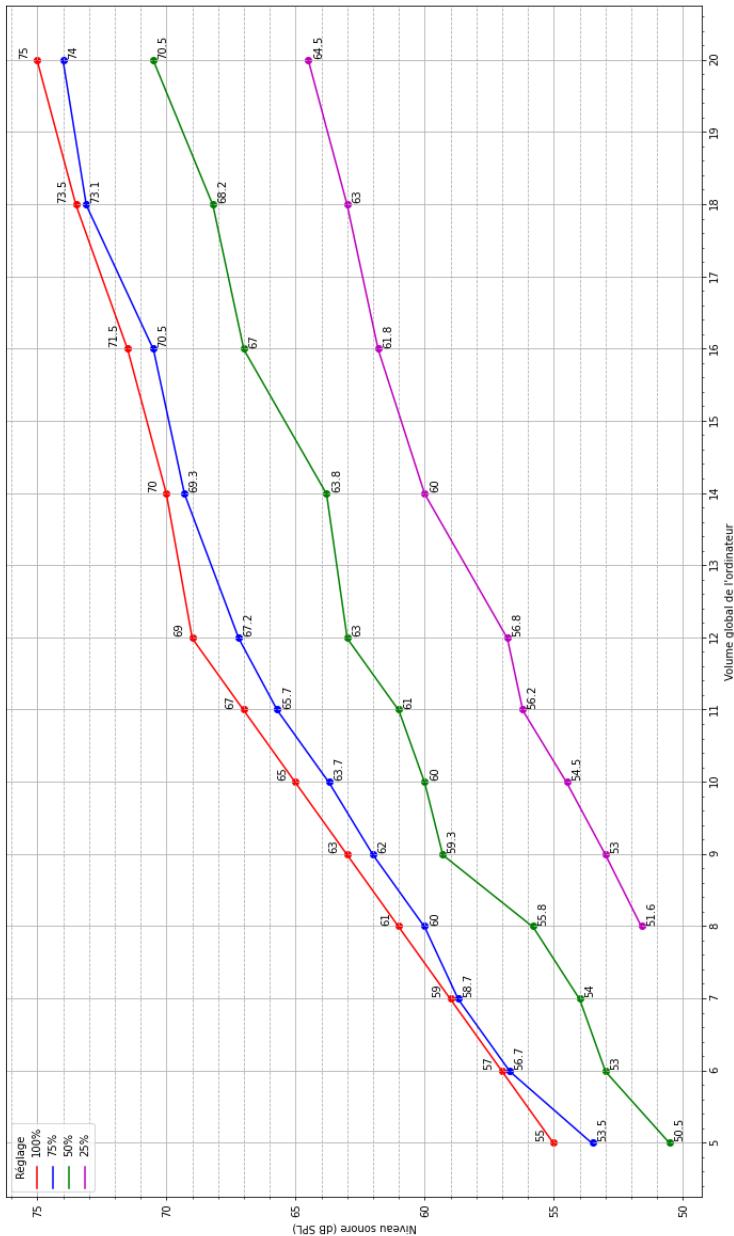


FIGURE 16 – Niveau sonore en fonction du volume de l'ordinateur, pour un bruit blanc, et différents réglages dans l'application Merspeakers

Code 2 – Angles d'inclinaisons en fonction de la distance mur-patient et de la hauteur de l'enceinte

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 Hpatient = 110 # hpatient environ entre 105 et 120
5
6 def z_function(longueur, Hmur):
7     angle = 90-(np.arctan(longueur/(Hmur - Hpatient)))*180/np.pi
8     return angle
9
10 chosen_x = 255 # respectivement 90, distance mur-patient à pointer
11 chosen_y = 173 # respectivement 203, hauteur du mur à pointer
12
13 longueur = np.linspace(60,345,200)
14 Hmur = np.linspace(140,300,300-140)
15
16 X, Y = np.meshgrid(longueur, Hmur)
17 Z = z_function(X, Y)
18
19 ax = plt.axes(projection="3d")
20 ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
21                 cmap='winter', edgecolor='none')
22 ax.set_xlabel('x = Distance mur-patient')
23 ax.set_ylabel('y = Hauteur enceinte')
24 ax.set_zlabel('z = Angle inclinaison enceinte')
25
26 chosen_z = z_function(chosen_x, chosen_y)
27 ax.scatter(chosen_x,chosen_y,chosen_z,color="red")
28 ax.text(chosen_x, chosen_y, chosen_z, s=(chosen_x, chosen_y, int(chosen_z)))
29
30 plt.show()

```

Code 3 – Hauteurs d'enceintes en fonction de la distance mur-patient et de l'angle d'inclinaison

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 Hpatient = 110 # hpatient environ entre 105 et 120
5
6 def z_function(longueur, angle):
7     Hmur = longueur/np.tan(np.deg2rad(90-angle)) + Hpatient
8     return Hmur
9
10 chosen_x = 255 # respectivement 90, distance mur-patient à pointer
11 chosen_y = 15 # angle d'inclinaison à pointer
12
13 longueur = np.linspace(60,345,200)
14 angle = np.linspace(0,chosen_y,chosen_y)
15
16 X, Y = np.meshgrid(longueur, angle)
17 Z = z_function(X, Y)
18
19 ax = plt.axes(projection="3d")
20 ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
21                 cmap='winter', edgecolor='none')
22 ax.set_xlabel('x = Distance mur-patient')
23 ax.set_ylabel('y = Angle inclinaison enceinte')
24 ax.set_zlabel('z = Hauteur enceinte')
25
26 chosen_z = z_function(chosen_x, chosen_y)
27 ax.scatter(chosen_x,chosen_y,chosen_z,color="red")
28 ax.text(chosen_x, chosen_y, chosen_z, s=(chosen_x, chosen_y, int(chosen_z)))
29
30 plt.show()

```

Code 4 – *Merspeakers.py*

```

1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  # ----- Importation -----
5
6  import os
7  import subprocess
8  import json
9  import sys
10 from PIL import ImageTk, Image
11 import ctypes
12
13 import tkinter as tk
14 from tkinter import messagebox
15
16 import pygame
17
18 # ----- Amélioration de la qualité graphique -----
19
20 if 'win' in sys.platform:
21     ctypes.windll.shcore.SetProcessDpiAwareness(1)
22
23 # ----- Dossiers -----
24 # permet de se référer au bon dossier dans lequel est compris l'application ou le
25 # script en fonction de ce qui lance l'application --> utile pour cx_freeze
26 if getattr(sys, 'frozen', False):
27     # frozen
28     script_dir = os.path.dirname(sys.executable)
29 else:
30     # unfrozen
31     script_dir = os.path.dirname(os.path.realpath(__file__))
32
33 # ----- Curseurs -----
34 # on nomme les curseurs pour plus de facilité
35 glassnormal = "@cursors/glassnormal.cur"
36 cursor_plus = "@cursors/7p.cur"
37 cursor_black = "@cursors/7n.cur"
38 cursor_vertical = "@cursors/7vs.cur"
39 cursor_horizontal = "@cursors/7hs.cur"
40 cursor_end = "@cursors/7u.cur"
41 cursor_wait = "@cursors/7w.ani"
42
43 # ----- Initialisation -----
44 # ouverture du fichier path.json pour lire le chemin d'accès à Focusrite Control
45 with open('path.json', encoding="utf-8") as json_file: # path to Focusrite Control.exe
46     path = json.load(json_file)
47
48 # lancement d'une grande page Tkinter, avec titre et logo, configuration du curseur
49 root = tk.Tk() #initialise l'application
50 root.title("Merspeakers")
51 root.iconbitmap(default=os.path.join(script_dir, "logo","icon.ico"))
52 root.config(cursor=glassnormal) #curseur souris de l'appli
53 root.state('zoomed')
54
55 width_screen = root.winfo_screenwidth() # récupère les information de l'écran
56 height_screen = root.winfo_screenheight()
57
58 active_buttons = [] # liste des boutons actifs
59 paused_buttons = [] # liste des boutons en pause
60 name_sounds = [] # liste des noms sans .wav des sons
61 holder_class = {} # nom sans .wav : objet de la classe Make_sound
62
63 # réglage de la taille de la police en fonction du nombre de pixels de largeur
64 if 1000 <= width_screen <= 1300 :
65     fontfamily = {'normal' : ('Helvetica', '13'), 'petit': ('Helvetica', '9')}
66 elif width_screen < 1000:
67     fontfamily = {'normal' : ('Helvetica', '9'), 'petit': ('Helvetica', '7')}
68 elif width_screen > 1300:
69     fontfamily = {'normal' : ('Helvetica', '15'), 'petit': ('Helvetica', '11')}
70
71 # ----- Creation of a list of sounds -----
72 # chemin d'accès au dossier sounds

```

```

73     soundpath = os.path.join(script_dir, "sounds")
74
75     # création d'une liste par ordre alphabétique de tous les noms de sons
76     wav_files = [f for f in os.listdir(soundpath) if os.path.isfile(os.path.join(soundpath, f))]
77     wav_files.sort()
78
79     # initialisation d'un mixage audio avec un nombre de canaux égal au nombre de sons
80     pygame.mixer.init()
81     pygame.mixer.set_num_channels(len(wav_files))
82
83     # ----- #
84     #           Hovering buttons           #
85     # ----- #
86
87     class HoverButton(tk.Button):
88         """classe de button qui change d'apparence en survolant"""
89         def __init__(self, master, enter, leave,**kw):
90             tk.Button.__init__(self,master=master,**kw)
91             self.enter = enter
92             self.leave = leave
93             self.bind("<Enter>", self.on_enter)
94             self.bind("<Leave>", self.on_leave)
95
96         def on_enter(self, e):
97             self['background'] = self.enter
98
99         def on_leave(self, e):
100            self['background'] = self.leave
101
102     class HoverCheckbutton(tk.Checkbutton):
103         """classe de button cochable qui change d'apparence en survolant"""
104
105         def __init__(self, master, enter, leave,**kw):
106             tk.Checkbutton.__init__(self,master=master,**kw)
107             self.enter = enter
108             self.leave = leave
109             self.bind("<Enter>", self.on_enter)
110             self.bind("<Leave>", self.on_leave)
111
112         def on_enter(self, e):
113             self['background'] = self.enter
114
115         def on_leave(self, e):
116             self['background'] = self.leave
117
118     # ----- #
119     #           Vertical scrolled frame           #
120     # ----- #
121
122     # classe inspirée d'un code de l'utilisateur "Gonzo" de StackOverflow
123     class VerticalScrolledFrame(tk.Frame):
124         """cadre possédant une barre de défilement vertical"""
125         def __init__(self, parent, *args, **kw):
126             tk.Frame.__init__(self, parent, *args, **kw)
127             self.parent=parent
128
129             # création d'une toile et d'une barre de défilement verticale
130             v-scrollbar = tk.Scrollbar(self, orient=tk.VERTICAL)
131             v-scrollbar.pack(fill="y", side="right",expand=True)
132             self.canvas = tk.Canvas(self, bd=0, highlightthickness=1,
133                                     yscrollcommand=v-scrollbar.set)
134             self.canvas.pack(fill="both", expand=True)
135             v-scrollbar.config(command=self.canvas.yview)
136
137             # création d'un cadre dans la toile qui va défiler avec elle
138             self.interior = interior = tk.Frame(self.canvas,bg="gray70")
139             interior_id = self.canvas.create_window(0, 0, window=interior,
140                                         anchor="nw")
141
142             # repère les changements appliqués à la toile et au cadre ; les synchronise
143             # met aussi à jour la barre de défilement
144             def _configure_interior(event):
145                 # met à jour la barre pour correspondre à la taille du cadre interne
146                 size = (interior.winfo_reqwidth(), interior.winfo_reqheight())
147                 self.canvas.config(scrollregion="0 0 %s %s" % size)
148                 if interior.winfo_reqwidth() != self.canvas.winfo_width():
149                     # met à jour la largeur de toile pour correspondre au cadre interne
150                     self.canvas.config(width=interior.winfo_reqwidth())

```

```

150     interior.bind('<Configure>', _configure_interior)
151
152     self.canvas.config(scrollregion=self.canvas.bbox("all"))
153
154 def _configure_canvas(event):
155     if interior.winfo_reqwidth() != self.canvas.winfo_width():
156         # update the inner frame's width to fill the canvas
157         self.canvas.itemconfigure(interior_id, width=self.canvas.winfo_width())
158
159     self.canvas.bind('<Configure>', _configure_canvas)
160
161 def _on_mousewheel(self, event):
162     if len(wav_files) > 25:
163         self.canvas.yview_scroll(int(-1*(event.delta/120)), "units")
164
165 # ----- #
166 # Focusrite Control #
167 # ----- #
168 class Focusrite:
169     """Classe répertoriant callbacks pour les boutons de Focusrite Control"""
170     def openfocusritecallback(self):
171         """callback pour le bouton d'ouverture"""
172         try:
173             self.Focusrite_control = subprocess.Popen(path["focusrite_control"])
174         except:
175             messagebox.showerror(u"Erreur à l'ouverture de Focusrite Control",
176                                 "Erreur dans l'ouverture de Focusrite Control,"
177                                 " veuillez vérifier le chemin d'accès à l'application"
178                                 "dans le fichier path.json")
179     def endfocusritecallback(self):
180         """callback pour le bouton de fermeture"""
181         try:
182             self.Focusrite_control.terminate()
183         except:
184             pass
185
186 # ----- #
187 # Sound Buttons #
188 # ----- #
189 class Make_sound:
190     """Classe utile à l'initialisation de buttons en fonction des
191     sons répertoriés, et de buttons lorsque les sons sont actifs.
192     """
193     def __init__(self, name, parent, parent2, i, i2=False):
194         """Initialisation
195
196         Args:
197             name (str): nom du son
198             parent : frame parent de gauche
199             parent2 : frame parent de droite
200             i (int) : nombre colonnes de gauche
201             i2 (bool, int): nombre colonnes de droite. Defaults to False.
202         """
203         # variable de contrôle
204         self.varbutton = tk.StringVar()
205
206         self.name = name
207         self.parent = parent
208         self.parent2 = parent2
209         self.soundpath = os.path.join(script_dir, "sounds", "{}.wav".format(self.name))
210
211         self.num = i
212         # création d'images qui seront invisibles et qui régleront la taille des boutons tkinter
213         self.pixel_button = tk.PhotoImage(width=int(width_screen/8), height=int(height_screen/8))
214         self.pixel_setting = tk.PhotoImage(width=int(width_screen/11), height=int(height_screen/13))
215         # valeurs par défaut des barres (volume, fader gauche-droite)
216         self.vol = 100
217         self.fader = 0.0
218         # liste des couleurs dans le dégradé utilisé pour le volume, pas entre les couleurs
219         self.scale_color = ["forest green", "green2", "green yellow", "yellow green",
220                            "yellow3", "orange", "dark orange", "orange red", "red2", "red4"]
221         self.step_color = (len(self.scale_color)-1)/100
222
223         self.soundbuttoncreator()
224
225     def soundbuttoncreator(self):
226         """Crée les boutons de gauche"""

```

```

227     # création d'un cadre pour les boutons
228     self.frame_button = tk.Frame(self.parent,bg="gray70", bd=3, relief="flat")
229     # nombre de buttons par longeur/largeur
230     self.rows_button = self.num//4
231     self.columns_button = self.num%4
232     # création du bouton
233     self.button = HoverCheckButton(self.frame_button, text=self.name.capitalize(),
234                                     indicatoron=False, selectcolor="DeepSkyBlue3",
235                                     background="light slate gray",
236                                     activebackground="LightSteelBlue3",
237                                     variable=self.varbutton, command=self.launchsound,
238                                     cursor=cursor_plus, image=self.pixel_button,
239                                     font=fontfamily["normal"], enter='chartreuse3',
240                                     compound="c", leave='light slate gray')
241     self.button.pack()
242
243     self.frame_button.grid(row=self.rows_button, column=self.columns_button)
244
245
246     def play(self):
247         """Joue le son"""
248         self.sound = pygame.mixer.Sound(self.soundpath)
249         self.chan = pygame.mixer.find_channel() # trouve un canal libre
250         self.chan.play(self.sound,loops=-1) # boucle pour répéter le son indéfiniment
251
252     def launchsound(self):
253         """Pilote ce qu'il faut faire quand un bouton de gauche est appuyé"""
254         # si le son n'est pas en pause
255         if self.name not in paused_buttons:
256             # si le bouton est inactif
257             if self.varbutton.get() == "1":
258                 # on le lance et on le note comme actif --> création du bouton
259                 self.button["cursor"] = cursor_black
260                 self.play()
261                 active_buttons.append(self.name)
262                 sounds_setting_buttons()
263
264             else:
265                 # sinon on stop le bouton
266                 self.button["cursor"]=cursor_plus
267                 self.chan.stop()
268                 self.vol = 100
269                 self.fader = 0.0
270                 active_buttons.remove(self.name)
271                 sounds_setting_buttons()
272
273         else:
274             # sinon il va falloir gérer la pause (autre fonction)
275             self.command_button_setting()
276
277     def soundbuttonsettingcreator(self,i2,parent):
278         """Crée les boutons de droite"""
279         # création d'un cadre pour les boutons
280         self.frame_setting = tk.Frame(parent,bg="gray70", bd=3, relief="ridge")
281         # nombre de buttons par longeur/largeur
282         self.rows_setting = i2//3
283         self.columns_setting = i2%3
284         # échelle de volume
285         self.volumescycle = tk.Scale(self.frame_setting, orient='vertical', from_=100,
286                                     to=0, resolution=0.1, command=self.setvolume,
287                                     cursor=cursor_vertical,fg=self.scale_color[-1])
288         self.volumescycle.grid(row=0,column=1, rowspan=2, sticky="nsew")
289         self.volumescycle.set(self.vol)
290         # échelle de spatialisation gauche-droite
291         self.faderscale = tk.Scale(self.frame_setting, orient='horizontal', from_=-1,
292                                     to=1, resolution=0.01, command=self.setbalance,
293                                     cursor=cursor_horizontal)
294         self.faderscale.grid(row=1,column=0, sticky="nsew")
295         self.faderscale.set(self.fader)
296         # bouton qui se crée quand le son est actif
297         self.button_setting = HoverCheckButton(self.frame_setting, text=self.name.capitalize(),
298                                             indicatoron=False, selectcolor="DeepSkyBlue3",
299                                             background="light slate gray",
300                                             activebackground="LightSteelBlue3",
301                                             variable=self.varbutton, leave='light slate gray',
302                                             cursor=cursor_black, image=self.pixel_setting,
303                                             font=fontfamily["petit"], enter='chartreuse3',

```

```

304                                         compound="c", command=self.command_button_setting)
305
306     self.button_setting.grid(row=0, column=0, sticky="nsew")
307
308     self.frame_setting.grid(row=self.rows_setting, column=self.columns_setting)
309
310 def setvolume(self,event):
311     """Règle le volume en fonction de l'échelle"""
312     self.vol = self.volumescycle.get()
313     self.volumescycle["fg"] = self.scale_color[int(round(self.step_color*self.vol))]
314
315     if self.varbutton.get() == "1":
316         self.sound.set_volume(self.vol/100)
317
318
319 def setbalance(self,event):
320     """Règle la spatialisation gauche-droite en fonction de l'échelle"""
321     if self.varbutton.get() == "1":
322         self.fader = self.faderscale.get()
323         if self.fader < 0:
324             volLeft = 0.5-self.fader/2
325             volRight = 1-volLeft
326             self.chan.set_volume(volLeft,volRight)
327         elif self.fader > 0:
328             volRight = 0.5+self.fader/2
329             volLeft = 1-volRight
330             self.chan.set_volume(volLeft,volRight)
331         else:
332             self.chan.set_volume(0.5,0.5)
333
334 def command_button_setting(self):
335     """Pilote ce qu'il faut faire quand un bouton de droite est appuyé"""
336     # si le bouton est inactif
337     if self.varbutton.get() == "1":
338         # on l'enlève de la pause
339         self.button["cursor"] = cursor_black
340         self.play()
341         paused_buttons.remove(self.name)
342         sounds_setting_buttons()
343
344     else:
345         # sinon on le met sur pause
346         self.button["cursor"] = cursor_plus
347         self.chan.stop()
348         paused_buttons.append(self.name)
349         sounds_setting_buttons()
350
351
352 # ----- #
353
354 def sounds_buttons(parent, parent2):
355     """Crée une liste de sons sans le .wav et initialise des objets de la classe
356     Make_sound qui sont rangés dans holder_class"""
357     # pour i allant dans la liste des sons wav
358     for i in range(len(wav_files)):
359         # on remplit la liste de sons en enlevant le .wav
360         name_sounds.append(wav_files[i][:-4])
361
362     # pour i dans la liste de sons
363     for i in range(len(name_sounds)):
364         # on range dans une liste les objets de la classe Make_sound que l'on crée
365         holder_class[name_sounds[i]] = Make_sound(name_sounds[i],parent,parent2,i)
366
367 def sounds_setting_buttons():
368     """Crée les boutons de droite"""
369     # création d'un cadre contenant les boutons
370     parent = tk.Frame(frame_fac_buttons, bg="gray70")
371     parent.grid(row=0, rowspan=7, sticky="nsew", padx=5, pady=10)
372     # pour i dans la liste de sons actifs
373     for i in range(len(active_buttons)):
374         # on lance la méthode soundbuttonsettingcreator relative au son en question
375         holder_class[active_buttons[i]].soundbuttonsettingcreator(i2=i, parent=parent)
376
377 def end_all():
378     """Éteint tous les sons"""
379     global active_buttons, paused_buttons # redéfinition globale de ces listes
380     # pour i dans la liste de sons actifs

```

```

381     for i in range(len(active_buttons)):
382         try: # essaie de tout éteindre et remet les paramètres à 0
383             holder_class[active_buttons[i]].varbutton.set("0")
384             holder_class[active_buttons[i]].chan.stop()
385             holder_class[active_buttons[i]].vol = 100
386             holder_class[active_buttons[i]].fader = 0.0
387         except: # il ne pourra pas y arriver si les sons sont en pause
388             pass
389         # on enlève tous les sons actifs et en pause
390         active_buttons = []
391         paused_buttons = []
392         sounds_setting_buttons() # on recrée un cadre pour contenir des boutons (vide)
393
394     def pause_all():
395         """Met en pause tous les sons"""
396         # pour i dans la liste de sons actifs
397         for i in range(len(active_buttons)):
398             # met tout en pause
399             holder_class[active_buttons[i]].varbutton.set("0")
400             holder_class[active_buttons[i]].chan.stop()
401             holder_class[active_buttons[i]].button["cursor"] = cursor_plus
402             paused_buttons.append(active_buttons[i])
403
404     # ----- # Creation #
405     # ----- #
406     # ----- # Initialisation #
407
408     # parent : création d'un cadre pour contenir celui qui défile les boutons à gauche
409     frame_buttons = tk.Frame(root,bd=5,bg="gray70")
410     frame_buttons.grid(row=1, column=0, rowspan=8, columnspan=6, padx=5, pady=0, sticky="nsew")
411     scframe_1 = VerticalScrolledFrame(frame_buttons)
412     scframe_1.pack(fill="both", expand=True)
413
414     # parent2 : création d'un cadre pour contenir celui qui range les boutons à droite
415     frame_fac_buttons = tk.Frame(root,bd=5,bg="gray70")
416     frame_fac_buttons.grid(row=1, column=6, rowspan=8, columnspan=4, padx=5, sticky="nsew")
417
418     # ----- # Focusrite #
419
420     focusrite = Focusrite() # on récupère les callbacks
421     # on crée les deux boutons de Focusrite Control
422     Button_open_focusrite = HoverButton(root, text="Ouvrir Focusrite Control",
423                                         command=focusrite.openfocusritecallback,
424                                         cursor=cursor_plus, background="light slate gray",
425                                         activebackground="LightSteelBlue3",
426                                         leave='light slate gray', enter='purple1',
427                                         font=fontfamily["petit"])
428     Button_open_focusrite.grid(row=9, column=0, columnspan=5, sticky="wens", padx=5, pady=3)
429     Button_end_focusrite = HoverButton(root, text="Fermer Focusrite Control", leave='light slate gray',
430                                         command=focusrite.endfocusritecallback, cursor=cursor_end,
431                                         background="light slate gray", activebackground="LightSteelBlue3",
432                                         font=fontfamily["petit"], enter='purple1')
433     Button_end_focusrite.grid(row=9, column=5, columnspan=5, sticky="wens", padx=5, pady=3)
434
435     # ----- # Boutons gauche #
436     # on lance la création des boutons à gauche
437     sounds_buttons(scframe_1.interior,frame_fac_buttons)
438
439     # ----- # logo #
440     # on rajoute le logo HUG
441     hug = Image.open(os.path.join(script_dir, "logo","hug.png"))
442     hug = hug.resize((int(2273/11), int(583/11)), Image.ANTIALIAS)
443     img = ImageTk.PhotoImage(hug)
444     panel = tk.Label(root, image=img)
445     panel.grid(row=0,column=0,sticky="nw")
446
447     # ----- # Boutons droite #
448     # on crée les boutons à droite
449     sounds_setting_buttons()
450
451     # ----- # Arrêter les sons #
452     # on crée les boutons de pause/arrêt
453     #pause
454     Button_pause = HoverButton(root, text="Mettre sur pause tous les sons", command=pause_all,
455                                cursor=cursor_wait, background="light slate gray", enter='yellow2',
456                                leave='light slate gray')

```

```

458                                     activebackground="LightSteelBlue3", font=fontfamily["normal"],
459                                     leave='light slate gray')
460 Button_pause.grid(row=7, column=6,padx=5,rowspan=1, columnspan=4,sticky="nsew")
461
462 #arrêt
463 Button_end_all = HoverButton(root, text="Arrêter tous les sons", command=end_all, leave='light slate gray',
464                               cursor=cursor_end, background="light slate gray", enter='red',
465                               activebackground="LightSteelBlue3", font=fontfamily["normal"])
466 Button_end_all.grid(row=8, column=6,padx=5,rowspan=1, columnspan=4,sticky="nsew")
467
468 # ----- #
469 #             Responsive Desgin          #
470 # ----- #
471 # on autorise l'augmentation de colonnes et lignes selon l'écran
472 for i in range(1, 9):
473     root.grid_rowconfigure(i, weight=1)
474 for i in range(6,10):
475     root.grid_columnconfigure(i, weight=1)
476
477 # ----- #
478 #             Volume de Windows          #
479 # ----- #
480
481 def askvolume():
482     """boîte de dialogue qui permet de s'assurer que l'utilisateur a bien réglé
483     le volume de son ordinateur"""
484     result = messagebox.askyesno("Attention au volume !",
485                                 u"Avez-vous bien correctement réglé le volume "
486                                 "principal de Windows ? Cela pourrait dans le "
487                                 "cas contraire être très dangereux pour le/la "
488                                 "patient(e) et vous-même !")
489     if result:
490         pass
491     else:
492         root.destroy() # quitte l'application
493
494 askvolume() # on ajoute la boîte de dialogue au lancement de l'application
495
496 # ----- #
497 #             ROOT                      #
498 # ----- #
499 # on lance l'application
500 root.mainloop()

```