

Rapport de Stage

Projet ScenaBox

Stagiaire : Théo DHUIEGE

Tutrice entreprise : Céline Jost

Tuteur enseignant : Pascal Hussaud

Période du 10 mars 2025 au 13 juin 2025

Résumé

Blabla

Abstract

Blabla

Remerciements

Je tiens à remercier :

- Céline Jost pour m'avoir fait confiance pour la reprise de l'un de ses projets ainsi que les aides informatiques et administratives.
- Gérard Uzan pour ses retours envers la disposition des composants.
- Dominique Archambault et Justin Debloos pour leurs enseignements de la modélisation et de l'impression 3D.
- Esrom Tebebu, Salvatore Anzalone et des étudiants du master Technologie et Handicap, avec qui j'ai participé au Challenge Handicap et Technologie.
- Johana Bodard, Subha et le reste de l'équipe pour leur hospitalité et l'entraide.

Table des matières

Résumé	2
Abstract	2
Remerciements	2
Table des figures	4
Table des tableaux	5
I. Introduction.....	6
a. Présentation du laboratoire CHArt	6
b. Présentation du stage	6
c. Présentation de l'existant : MulseBox	6
II. Etape 1 : analyse de l'existant et tests de composants	9
a. Recherche de défauts et d'améliorations.....	9
b. Test de composants et choix des nouveaux composants	10
1. Test des boutons.....	10
2. Test des cartes programmables	11
3. Test des ventilateurs	12
4. Test des lecteurs audio.....	12
5. Test des écrans	13
6. Conclusion des tests.....	15
III. Etape 2 : conception d'un nouveau boîtier	16
a. Réduction de la taille du boîtier	16
b. Fixation des composants au nouveau boîtier	17
c. Amélioration du flux d'air.....	19
d. Impression de la première version.....	21
e. Première programmation pour tester les composants	23
f. Amélioration du nouveau boîtier	24
1. Optimisation de la position des composants	24
2. Modification du système de fermeture.....	25
IV. Valorisation.....	28
a. Participation au treizième challenge handicap et technologie	28
b. Soumission d'un article de démonstration pour la conférence IHM.....	29
V. Annexe 1 : la modélisation 3D via des programmes.....	30
VI. Annexe 2 : quelques programmes de modélisation 3D	39
VII. Annexe 3 : les programmes de tests des composants et du Simon pour ScenaBox.....	40

Table des figures

Figure 1 : MulseBox vu de face.....	7
Figure 2 : présentation des capteurs et actionneurs de MulseBox	7
Figure 3 : un des boutons actuels Figure 4 : un des nouveaux boutons.....	10
Figure 5 : jeu de Simon pour tester les boutons.....	10
Figure 6 : un des schémas de câblage pour visualiser les broches utilisées	11
Figure 7 : platine de son Adafruit	12
Figure 8 : module DF Player Mini.....	13
Figure 9 : écran Adafruit 2260	14
Figure 10 : écran Adafruit 1770	14
Figure 11 : écran Waveshare	15
Figure 12 : disposition des composants pour ScenaBox	16
Figure 13 : modélisation de la platine pour tester l'agencement des composants.....	17
Figure 14 : pièce de test des fixations avec certains composants dessus.....	18
Figure 15 : une LED dans la pièce de test des fixations	18
Figure 16 : support de la Raspberry. A gauche, le cylindre troué sans insert et à droite, le cylindre avec l'insert soudé dans le plastique.....	18
Figure 17 : fixation de l'écran	19
Figure 18 : schéma de démonstration de l'effet Venturi	19
Figure 19 : les six versions de la pièce de compression d'air. Les versions sont disposées dans le sens de lecture.	20
Figure 20 : modélisation du boîtier.....	21
Figure 21 : impression ratée du boîtier	22
Figure 22 : impression en cours du boîtier	22
Figure 23 : système partiellement monté hors de son boîtier	23
Figure 24 : angles arrondis du boîtier	24
Figure 25 : platine de la première disposition des composants	25
Figure 26 : modélisation de la deuxième version du boîtier, avec le nouvel agencement des composants	25
Figure 27 : système de rails de la première version du boîtier	26
Figure 28 : petites pièces de tests des rails.....	26
Figure 29 : longue pièce de test des rails	26
Figure 30 : système de rails de la deuxième version du boîtier	27
Figure 31: préparation du stand pour le Challenge Handicap et Technologie	28
Figure 32 : un cylindre modélisé via programme	31
Figure 33 : en changeant la valeur des variables, la forme change de propriétés	31
Figure 34 : cube creusé.....	32
Figure 35 : cube sans argument "center"	32
Figure 36 : cube creusé en volume mais pas en surface	33
Figure 37 : cube correctement creusé.....	33
Figure 38 : vue d'ensemble du logiciel OpenSCAD, avec le bouton "Calculer le rendu" survolé	34
Figure 39 : barre d'actions de OpenSCAD, avec le bouton "exporter en STL" survolé	34
Figure 40 : vue d'ensemble du logiciel Ultimaker Cura.....	35
Figure 41 : exemple d'impression avec un matériau d'impression et un support	35
Figure 42 : exemple de choix d'impression d'une pièce avec deux couleurs différentes.....	35
Figure 43 : exemple de choix d'impression d'une pièce avec deux matériaux différents.....	36
Figure 44 : les paramètres d'impression	36
Figure 45 : estimation des coûts temporel et matériel de l'impression	37
Figure 46 : aperçu d'une pièce entière avant impression	37
Figure 47 : aperçu de la couche 39 sur 87 de cette même pièce	38

Table des tableaux

Tableau 1 : Comparaison des écrans testés	13
-------------------------------------------------	----

I. Introduction

Mon stage s'est déroulé au laboratoire CHArt, basé à l'université Paris 8. J'ai eu l'occasion de travailler sur le projet ScenaBox, dans divers domaines tels que l'analyse des composants du système, la conception du boîtier, le montage ou encore la programmation de programmes de test. Dans ce document, vous trouverez un résumé de mes travaux réalisé pour le projet.

a. Présentation du laboratoire CHArt

Le laboratoire CHArt, pour Cognitions Humaine et Artificielle est un laboratoire créé en 2006 regroupant des chercheurs en sciences cognitives. Ce laboratoire effectue ses recherches dans le domaine cognitif sous tous ses aspects : l'acquisition des stimulations naturelles et artificielles et la réaction des patients à ces stimulations.

Les interventions du laboratoire sont à la fois sur le plan d'apprentissage que sur la prise de décisions et permettent aux patients d'améliorer et de réapprendre leurs capacités de réagir aux stimulus, de prendre des décisions et de réagir face à ces stimulations.

Le laboratoire effectue également ses recherches sur les nouvelles formes d'interactions que permettent les nouvelles technologies et met en place de nouvelles formes d'interactions, que ce soit pour faciliter la vie de personnes ou pour permettre un apprentissage de gestes et d'habitudes.

Parmi les projets orchestrés par les chercheuses et chercheurs du laboratoire,

b. Présentation du stage

Durant ce stage, j'ai dû analyser et améliorer le système électronique d'un projet. J'ai également eu l'occasion de découvrir la modélisation et l'impression 3D, ce qui m'a permis de modéliser et d'imprimer un nouveau boîtier pour ce projet. Enfin, je me suis occupé de la programmation des composants afin de permettre leur fonctionnement dans un contexte global.

c. Présentation de l'existant : MulseBox

ScenaBox consiste en un boîtier comprenant divers capteurs et actionneurs dont l'objectif est la stimulation cognitive et l'utilisation des membres et des sens via des activités diverses. Ces activités permettent également aux patients de réapprendre à faire des gestes du quotidien.

ScenaBox est la nouvelle version d'un projet nommé MulseBox, que l'on peut voir ci-dessous :

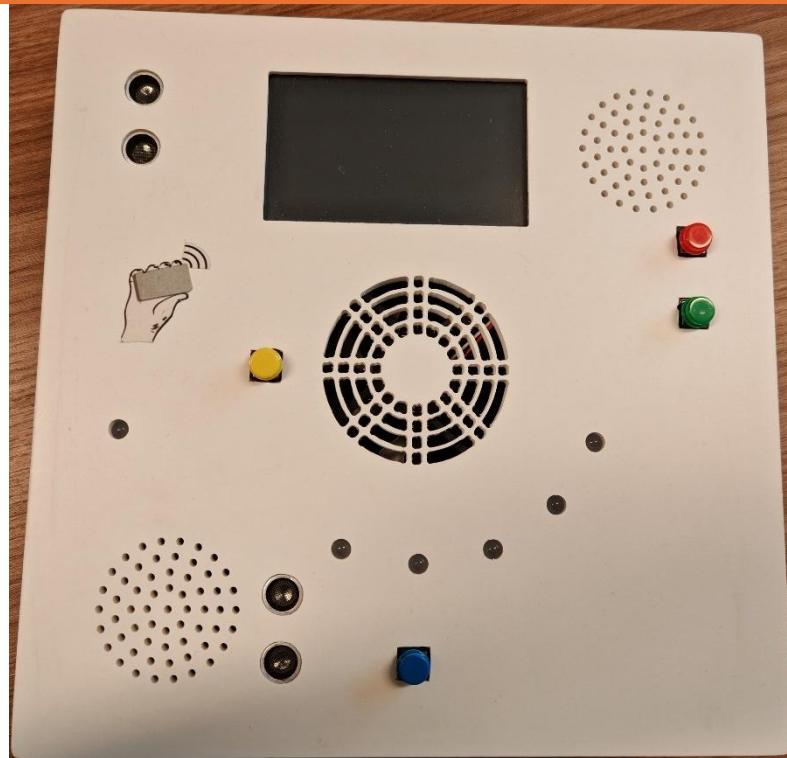


Figure 1 : MulseBox vu de face

La disposition des capteurs et des actionneurs est réfléchie avec des ergonomes afin de convenir au plus de situations possibles tout en proposant une disposition la plus agréable et ludique pour le patient :

- Aucune symétrie n'est présente pour éviter le phénomène d'habituation face au système.
- Le ventilateur est placé au centre du boîtier et visera la tête du patient.
- Un coin est laissé volontairement vide pour que le patient puisse se concentrer sur cette zone pour se détendre durant les exercices.
- Deux des quatre boutons sont mis ensemble afin de permettre de représenter un choix binaire. Les deux autres sont isolés des autres pour permettre des actions simples.
- Les haut-parleurs sont placés sur deux coins opposés pour permettre d'avoir une disposition stéréophonique gauche-droite et haut-bas quel que soit l'orientation du boîtier. Les vibrateurs et les télémètres à ultrasons sont également placés dans la même idée :

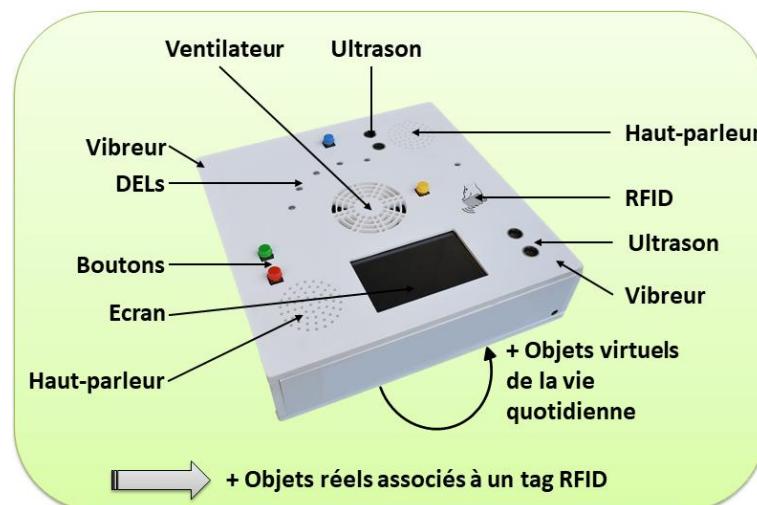


Figure 2 : présentation des capteurs et actionneurs de MulseBox

- Les LED sont placées en file de cinq pour permettre de faire des effets entre eux tels qu'une barre de chargement ou un chenillard. Le nombre impair de LED est aussi choisi pour pouvoir exprimer un avis neutre. Enfin, une sixième LED est isolée afin de pouvoir l'utiliser dans une interaction seule.
- Le ventilateur, les haut-parleurs, le lecteur de cartes et les vibreurs sont cachés afin de ne pas trop surcharger visuellement la surface.
- Dans l'orientation par défaut, l'écran est placé en haut au milieu.
- Dans cette même orientation, un des capteurs à ultrason est placé en bas pour qu'il soit aisément amener la main. Il est possible, par exemple, de simuler le rebond d'une balle et en même temps afficher la position de cette balle sur l'écran.
- Plusieurs composants sont volontairement approchés afin de permettre une interaction sous deux formes mais localisée en un point : par exemple, tourner le ventilateur et allumer les LED autour d'elle. Elle permet aussi à ce que l'interaction soit de cause à effet : interagir sur un des deux composants provoque une réaction sur le second. Parmi ces duos de composants, on peut citer le bouton jaune et le ventilateur ou la LED seule et le lecteur de cartes.

II. Etape 1 : analyse de l'existant et tests de composants

Mettre une petite intro ici pour dire ce qu'on trouve dans cette partie.

a. Recherche de défauts et d'améliorations

Dans un premier temps, j'ai commencé à analyser la conception de MulseBox afin de trouver des axes d'améliorations potentielles d'un point de vue électronique.

La version actuelle du projet contient deux cartes programmables : une Arduino Uno et une Raspberry Pi 3B. La Raspberry est la carte principale où les scénarios sont stockés et où l'écran et le son sont connectés. L'Arduino permet de faire la liaison avec les autres capteurs et actionneurs. Les informations sur ces composants sont communiquées au Raspberry de manière continue via une liaison série en USB.

Le but d'utiliser certains composants par deux est de permettre une spatialisation des sensations émis ou reçus, y compris pour les vibreurs. Cependant, le boîtier étant rigide et formé d'un seul bloc, quel que soit le vibreur en action, nous ne sentons pas lequel est alimenté. Nous n'en garderons alors qu'un seul.

Le télémètre à ultrason proche du bouton bleu est perturbé par une des LED RGB lorsque cette dernière est allumée.

Il faudra explorer la possibilité de n'opter que pour une seule carte programmable. Parmi celles à explorer : L'Arduino Nano 33 BLE Sense Rev2, la Raspberry Pi Zero 2W, la Raspberry Pi 5 ou encore la Raspberry Pi Pico.

Pour finir, l'aspect esthétique de MulseBox est à améliorer : le boîtier dispose d'une apparence jugée trop vieille et d'une taille et d'une masse trop grande.

b. Test de composants et choix des nouveaux composants

L'objectif de ces tests est de trouver de meilleurs composants pour ScenaBox. Pour cela, j'ai testé divers composants afin de savoir le plus approprié pour ScenaBox. Ces composants peuvent être plus complets, plus compacts ou encore plus adaptés au projet. Voici les tests effectués :

1. Test des boutons

Ci-dessous, un des boutons actuels est montré en Figure 3 et un des nouveaux boutons est montré en Figure 4 :



Figure 3 : un des boutons actuels



Figure 4 : un des nouveaux boutons

Pour essayer les nouveaux boutons, j'ai programmé un jeu de Simon. Dans ce jeu, les boutons s'illuminent dans un certain ordre. Il faut retenir puis reproduire l'ordre dans lequel les boutons s'illuminent. En cas de reproduction correcte, la séquence est allongée mais à la moindre erreur, la séquence est réinitialisée. Ce jeu est développé sur un environnement simple mais qui permet de tester ces boutons : une boîte en carton. Le système est basé sur une Arduino Nano 33 BLE Sense Rev 2 et programmé en C++.

Ci-dessous, le jeu de Simon réalisé pour tester les boutons :



Figure 5 : jeu de Simon pour tester les boutons

Les nouveaux boutons sont plus solides, un peu plus grands, et sont équipées d'une LED en interne pour les illuminer. De plus, le cliquetis a été plus apprécié par la globalité du laboratoire que celui des boutons actuels. Ces nouveaux boutons sont donc retenus.

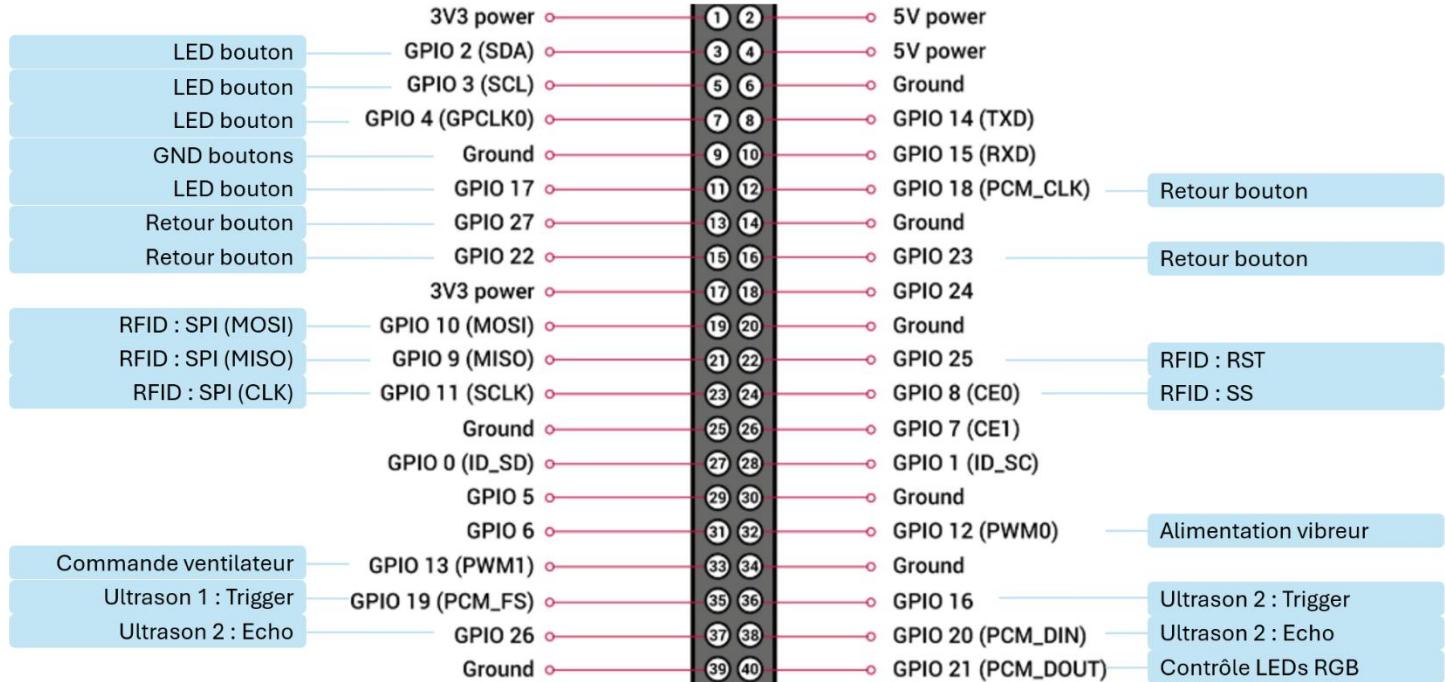
2. Test des cartes programmables

L'idée principale de cette comparaison est de réduire le nombre de carte à une seule, qui s'occuperait alors à la fois du stockage des scénarios ainsi que de la gestion de l'écran et du son mais aussi de la gestion des capteurs et des actionneurs.

Un élément important est de déterminer le câblage selon la carte et les composants choisis. Cela permet également de voir les cas non possibles, comme utiliser une Arduino ou Raspberry Pi Pico avec un écran en HDMI. J'ai donc conçu des schémas de câblage pour plusieurs cas afin de permettre à l'équipe au complet de comprendre les contraintes de chaque cas. Les cas étudiés sont :

- Cas de MulseBox : Raspberry Pi 3 + Arduino Uno
- Raspberry Pi 4b + Arduino Nano 33 BLE Sense Rev2
- Raspberry Pi 4b seule
- Raspberry Pi 5 seule
- Raspberry Pi Zero 2W seule
- Raspberry Pi Pico seule
- Arduino Nano 33 BLE Sense Rev2 seule

Utilisation Raspberry Pi 4B seule



L'écran est connecté sur HDMI. Pour le son, le port Jack du Raspberry sera utilisé.

Figure 6 : un des schémas de câblage pour visualiser les broches utilisées

Etant donné que le nombre de broches est limité, et que les Raspberry Pi Zero, 4b ou 5 permettent de connecter l'écran via les ports HDMI alors que les Arduino et que la Raspberry Pi Pico qui n'en ont pas, notre choix se limite à eux.

Pour choisir entre ces trois modèles, nous souhaitions prendre la carte la plus puissante, soit la Pi5. Cependant, des incompatibilités dans librairies des LED et du lecteur RFID ont été constatées. De plus, cette carte ne dispose pas de port Jack audio, or, nous voulons utiliser ce port pour ne pas user des GPIO ni un adaptateur USB-Jack. Nous avons finalement choisi le modèle précédent : la Raspberry Pi 4b.

3. Test des ventilateurs

Celui actuellement présent dans ScenaBox est un ventilateur de taille 80x80x25mm fonctionnant en 12V. Il produit une bonne quantité de souffle mais est trop volumineux et produit un cliquetis à l'allumage et à l'extinction, dû au relais utilisé pour l'alimenter.

Celui testé est beaucoup plus compact, mesurant 40x40x10mm. De plus, il fonctionne en 5V, ce qui lui permettra d'être directement alimenté par la Raspberry. Cependant, à cause de sa compacité, le souffle est plus faible. Il faudra alors trouver un moyen de guider l'air afin de mieux sentir le souffle.

Le ventilateur actuel a alors été remplacé par celui testé, mais pour produire un meilleur souffle, on va essayer de développer une pièce dans ce but.

4. Test des lecteurs audio

La solution actuelle, via la sortie son de la Raspberry est une solution simple, en stéréo et qui n'occupe pas de broche. Pour restituer le son, il manque un module amplificateur. Celui-ci peut être soit un module déjà tout prêt, qui s'alimente via un port USB et prend l'audio par le port Jack Audio, soit un module à monter, qui prends le son via quelques ports GPIO et alimenté via ces mêmes ports.

La platine son Adafruit est facile à utiliser. Il suffit de souder tous les composants dessus, étant vendue non montée, puis de l'installer sur la carte Arduino Uno et enfin d'insérer une carte SD contenant les sons à lire, ce qui ferait deux cartes utilisées : celle de la Raspberry et celle pour la platine Adafruit. Via une librairie fournie par Adafruit, il est possible de lire les sons stockés dans la carte SD. Cependant, il y a quelques limitations sur les sons à lire : les fichiers sonores doivent être au format .wav et obligatoirement échantillonné à 22,05 kHz. De plus, le son est restitué sur un port jack mono, ce qui signifie que si cette carte est utilisée, il n'y aura plus de stéréoscopie, ce qui ne respecte pas l'une des exigences sur les haut-parleurs.

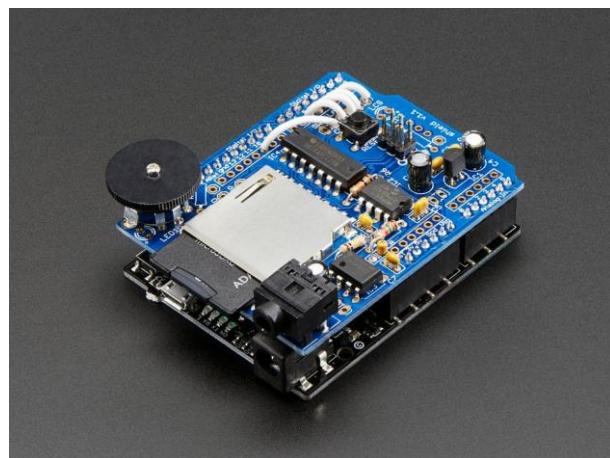


Figure 7 : platine de son Adafruit

Enfin, le module DFPlayer Mini est une autre solution plausible. Le module communique via liaison série, qui demande trois broches de moins que la carte Adafruit. De plus, ce composant dispose de plein de fonctionnalités utile : compatibilité avec fichiers au format MP3, échantillonné jusqu'à 48 kHz, lecture d'un dossier de son de manière aléatoire ou encore modification du volume sonore. Le seul défaut trouvé est le fait que la sortie directe ne supporte que le mono. Pour avoir un signal stéréo, il faudra passer par la sortie secondaire, ce qui nécessite un système d'amplification en amont des haut-parleurs.

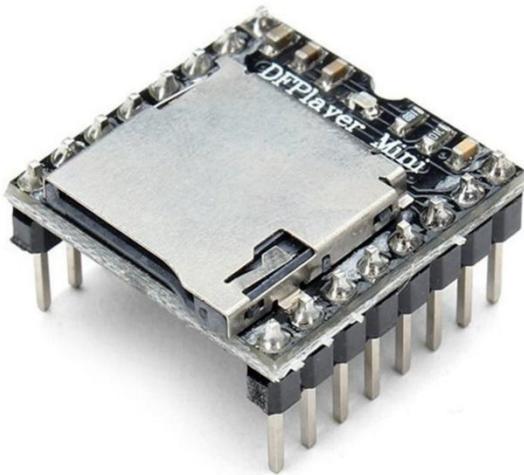


Figure 8 : module DF Player Mini

5. Test des écrans

Ci-dessous, un tableau comparatif des principales caractéristiques des écrans testés :

Ecran	Référence	Taille	Définition	Interface	Remarques
A	Adafruit 2260	4,3"	640x480	HDMI et USB	Alimentation et tactile via USB
B	Adafruit 1770	2,8"	320x240	SPI avec deux SS	Images uniquement en format BMP
C	DWIN dmg48270c043-03wtc	4,3"	480x272	Série	Images uniquement en format BMP
D	Waveshare	3,5"	640x480	HDMI et USB	Alimentation et tactile via USB

Tableau 1 : comparaison des écrans testés

L'écran A est un composant fiable et fonctionnel : il nous affiche directement l'écran de la Raspberry. Cela apporte l'avantage de pouvoir déboguer le système, simplement en utilisant cet écran comme moniteur de la Raspberry.



Figure 9 : écran Adafruit 2260

L'écran B fonctionne correctement, via une liaison SPI avec deux broches slave select : une pour le contrôle de l'écran lui-même et l'autre pour le contrôle du lecteur de carte micro-SD. Sans cette carte micro-SD, il est possible d'afficher du texte et de dessiner des formes. En y ajoutant la carte, on peut y afficher des images stockées dedans, à condition que les images soient au format BMP.

En utilisant cet écran, nous pourrions réduire la place prise car celui testé est 1,4 pouce plus petit. Cependant, cet écran dispose d'une définition plus faible et ne se connecte pas via HDMI mais se contrôle par liaison SPI. Il faudra alors revoir le stockage des images à afficher. De plus, l'affichage des images est plus rapide avec l'écran numéro 1 que le numéro 2.

Cet écran n'est pas retenu pour ces raisons : on ne peut afficher l'écran de la Raspberry ni les interfaces graphiques déjà développées.

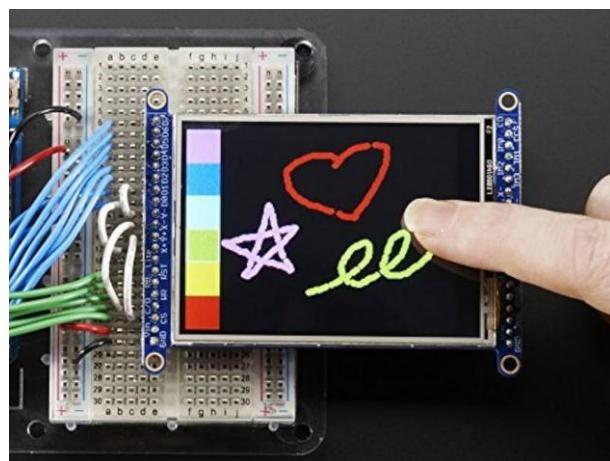


Figure 10 : écran Adafruit 1770

L'écran C est de type IHM programmable dont la programmation du comportement doit se faire via un logiciel dédié. Le contrôle logiciel de cet écran se déroule via une liaison série. Cet écran a pour avantage de fonctionner sur toutes les configurations Arduino ou Raspberry en réflexion. Cependant, la définition reste plus basse que l'écran 1 et la qualité globale de l'image est plus faible. Les images à afficher doivent obligatoirement être au format .bmp et la communication série n'est pas entièrement fonctionnelle : l'écran ne réagit pas aux commandes envoyées par un ordinateur. Son utilisation est donc trop complexe pour notre projet.

L'écran D possède un fonctionnement identique au premier et donc également ses avantages. L'image est affichée par un signal HDMI et les informations sur le tactile et l'alimentation sont fournies par le câble USB. Cet écran s'utilise donc sans nécessité de programmer, simplement à le connecter et la

Raspberry l'alimentera et communiquera directement avec lui. Il s'agit donc d'un écran s'utilisant comme celui actuellement présent mais en plus petit. Donc cet écran est le plus approprié.



Figure 11 : écran Waveshare

Comme ScenaBox sera désormais basée sur une Raspberry Pi 4b, nous choisissons la solution donnant le plus de possibilités en programmation : un écran utilisant les ports HDMI et USB. Pour savoir lequel prendre entre les écrans A et D, nous gardons le plus petit : l'écran D.

6. Conclusion des tests

Ces tests m'ont permis de choisir les composants de cette nouvelle version de ScenaBox, qui sera désormais basée sur une Raspberry Pi 4b seul et aura un mélange entre nouveaux composants et composants de la version précédente conservés. Les nouveaux composants sont l'écran D les boutons équipés de LED internes et le petit ventilateur. Les composants conservés sont le lecteur de cartes, les télémètres, le vibreur et les LED adressables.

Les composants sont donc les suivants :

- Carte : Raspberry Pi 4b avec 8Go de RAM.
- Ecran : tactile 3.5”, définition 640x480, interfaces HDMI et USB.
- Son : Sortie Jack Audio, connecté à deux haut-parleurs alimentés par USB.
- Télémètres : HC-SR04
- LED : WS2812b
- Boutons : EOZ FL13LR5, FL13LY5, FL13LG5 et FL13LB5
- Lecteur de cartes RFID : RC-522
- Ventilateur : 5V, format 40x40x10 millimètres
- Vibreur : Seeed 316040001

III. Etape 2 : conception d'un nouveau boîtier

a. Réduction de la taille du boîtier

Cette nouvelle version de ScenaBox sera également plus compacte. Pour déterminer l'objectif de taille, nous avons utilisé la taille maximale d'impression de l'imprimante 3D du laboratoire : 23 centimètres. Cette taille correspond également à la taille de deux mains, ce qui permet de facilement explorer la zone. MulseBox est trop grand, et donc plus difficile à explorer.

Pour l'objectif de hauteur, nous avions souhaité 3 centimètres afin de réduire de plus de moitié la hauteur comparée à celle de MulseBox. Cependant, 3 centimètres est trop petit, car certains composants ne rentreraient pas avec cette hauteur. Ces composants sont les haut-parleurs et la Raspberry avec ses câbles. Finalement, nous avons choisi une hauteur de 4 centimètres, ce qui correspond à un peu plus que la moitié de la hauteur de MulseBox.

Pour réduire la taille de ScenaBox pour quelle ne fasse que 23 centimètres de côté, il faut repenser l'emplacement des composants. Il faut garder les mêmes exigences que la version précédente, donc garder une disposition ressemblante. Cependant, comme certains composants seront changés par des plus petits, il est plus facile d'imaginer la disposition dans une boîte plus petite.

Après quelques recherches et réadaptations, nous sommes arrivés à cette disposition :

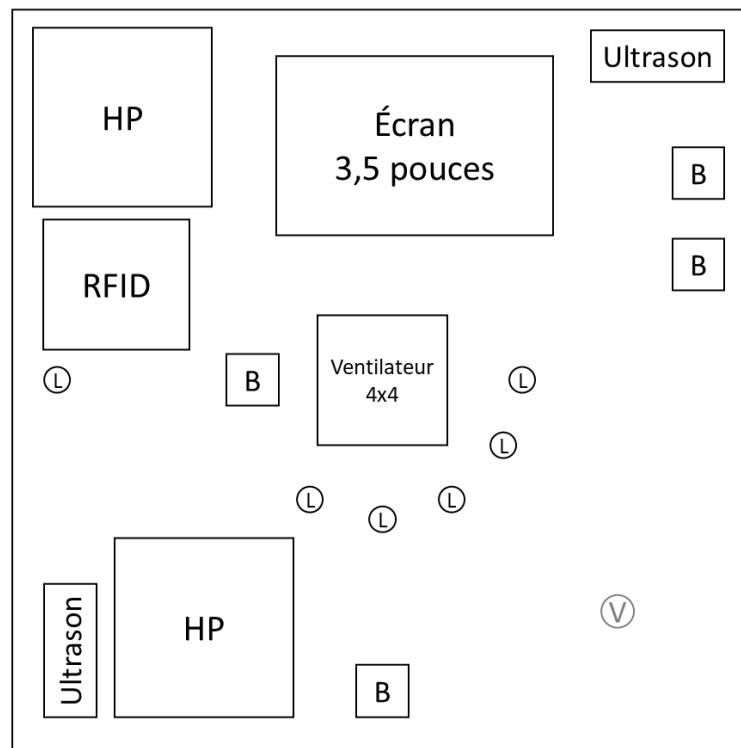


Figure 12 : disposition des composants pour ScenaBox

Nous procédons cependant à quelques changements :

- Comme exigé, le boîtier ne fait désormais plus que 23 centimètres de côté, contre 30 avant et 4 centimètres de hauteur, contre 7 auparavant. De plus, les bords seront arrondis afin d'apporter un aspect plus moderne.
- Les composants situés sur les angles en bas à gauche et en haut à droite ne sont plus les haut-parleurs mais les télémètres. Les haut-parleurs ont échangé leur position avec les télémètres.

- Le vibreur est placé sous la surface de jeu afin de le laisser caché dans la boîte, et un seul ne sera utilisé.

Une des pièces modélisées est la plaque ci-dessous. Son but est de déterminer l'emplacement des composants sur une version plus petite du boîtier et de faire valider cette configuration par toute l'équipe.

Le but est de garder les mêmes contraintes de conception que la version précédente dans un espace restreint :

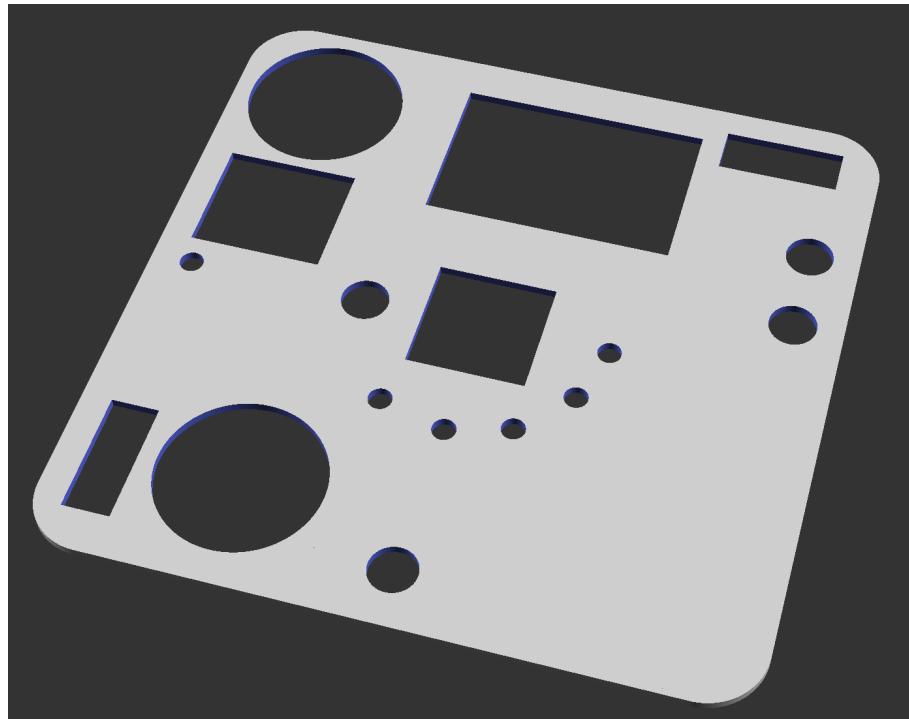


Figure 13 : modélisation de la platine pour tester l'agencement des composants

Cette plaque a permis à l'équipe de se projeter dans cette nouvelle configuration et dans l'utilisation de ScenaBox. L'agencement des composants est validé.

b. Fixation des composants au nouveau boîtier

Le principal problème étant que les composants sont fixés à l'envers, ils sont soumis à la gravité. Il faut donc trouver un moyen de permettre aux composants de tenir dans la boîte. Pour cela, je vais utiliser principalement deux méthodes : le maintien par la force et la fixation.

Pour essayer ces moyens de fixation, j'ai développé cette pièce dont le but est de réunir un logement pour chaque composant afin de tester son moyen de fixation :



Figure 14 : pièce de test des fixations avec certains composants dessus

Les LED et les télémètres sont maintenus par les trous qui les dévoile : ces trous sont juste assez grands pour soutenir leur composant, comme l'on peut voir en Figure 14 et en Figure 15.



Figure 15 : une LED dans la pièce de test des fixations

L'écran, les boutons et la Raspberry sont visés dans le boîtier. Pour cela, je vais utiliser des inserts filetés. Ces inserts prennent la place de trous volontairement faits dans la pièce et sont soudés dans le plastique.



Figure 16 : support de la Raspberry. A gauche, le cylindre troué sans insert et à droite, le cylindre avec l'insert soudé dans le plastique.

Pour l'écran, la fixation se fait également via des vis, mais ici, deux pièces viennent supporter l'écran afin de le fixer au boîtier :



Figure 17 : fixation de l'écran

c. Amélioration du flux d'air

La dernière difficulté pour le boîtier était de trouver une méthode pour concentrer le souffle du ventilateur. Pour cela, nous avons eu l'idée de s'inspirer de l'effet Venturi.

L'effet Venturi consiste en la modification de la vitesse de déplacement et de la pression de l'air en le faisant passer dans un trou de plus en plus petit : plus le trou se rétrécit en diamètre, plus l'air se comprime et se pousse : il va alors plus vite. En ressortant du trou, l'air se décomprime, provoquant un appel d'air autour du flux sous pression, augmentant alors le souffle.

Ci-dessous, l'effet Venturi illustré dans un système de spa, pour aspirer de l'air et le mélanger à de l'eau :



Figure 18 : schéma de démonstration de l'effet Venturi

Pour créer un effet Venturi, nous avons imaginé une pièce se mettant à la suite du ventilateur et se composant de quatre parties :

- Accueil de l'air, afin d'éviter qu'il ne s'échappe avant de se compresser.
- Guidage : l'air se dirige vers un des trous. Le début des trous a un diamètre fixe pour bien guider l'air.
- Compression : le diamètre des trous se rétrécit petit à petit pour comprimer l'air progressivement.

- Sortie : le diamètre a été réduit, l'air est compressé et peut désormais sortir de la pièce.

Cette pièce a eu plusieurs versions afin de tester le bon fonctionnement de l'effet Venturi :

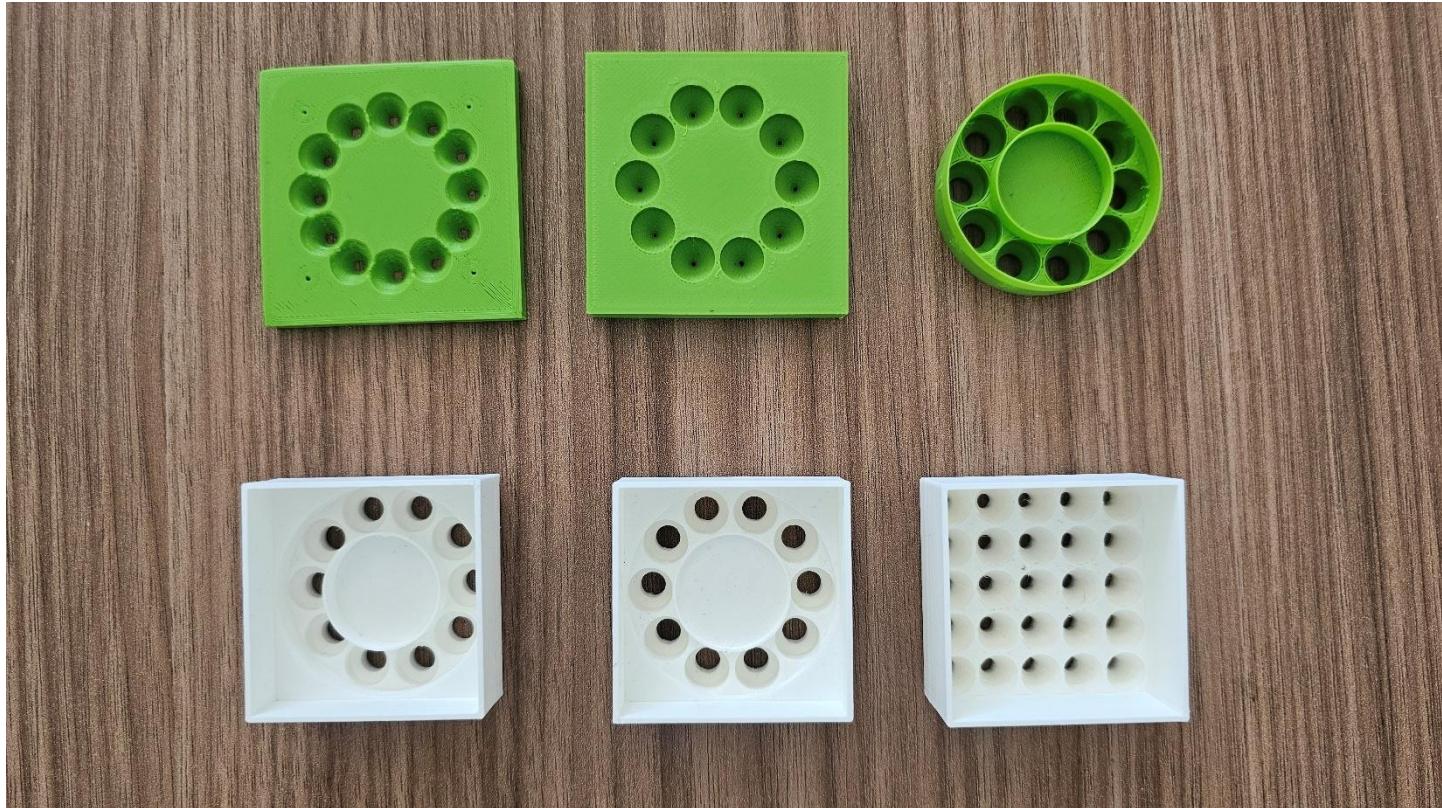


Figure 19 : les six versions de la pièce de compression d'air. Les versions sont disposées dans le sens de lecture.

Les versions ont pour but de trouver diverses solutions au problème.

La première version ne comprenait pas de zone d'accueil, ni de zone de guidage de l'air. Cette version ne produit pas d'amélioration du souffle. J'ai alors tenté avec d'autres dimensions.

La deuxième version dispose de trous plus longs, d'une sortie plus petite et d'une zone de guidage. Cependant, la sortie est trop petite, ce qui empêche l'air de sortir. De plus, j'ai remarqué qu'une partie de l'air aspiré par le ventilateur ne rentre pas dans les entonnoirs, mais ressort sur les côtés du ventilateur, voire font demi-tour et ressortent à l'arrière du ventilateur.

La troisième version essaie la perspective inverse : avoir des trous plus grands. De plus, cette pièce est la première qui dispose d'un couloir dans le but d'accueillir l'air avant de le compresser. Ici, le ressenti de l'air était meilleur, mais pas idéal.

La quatrième version contient l'ajout d'un logement pour le ventilateur, afin de bien placer les trous. Cela a réduit l'échappement de l'air par les côtés du ventilateur.

La cinquième version est analogue à la quatrième, mais la profondeur des trous est revue pour essayer de mieux faire fonctionner l'effet. C'est cette version qui fonctionne le mieux.

La sixième version est analogue à la cinquième, mais avec une disposition des trous en grille et non en anneau.

Une fois toutes les difficultés réglées, le boîtier peut être modélisé. Il sera composé de deux pièces :

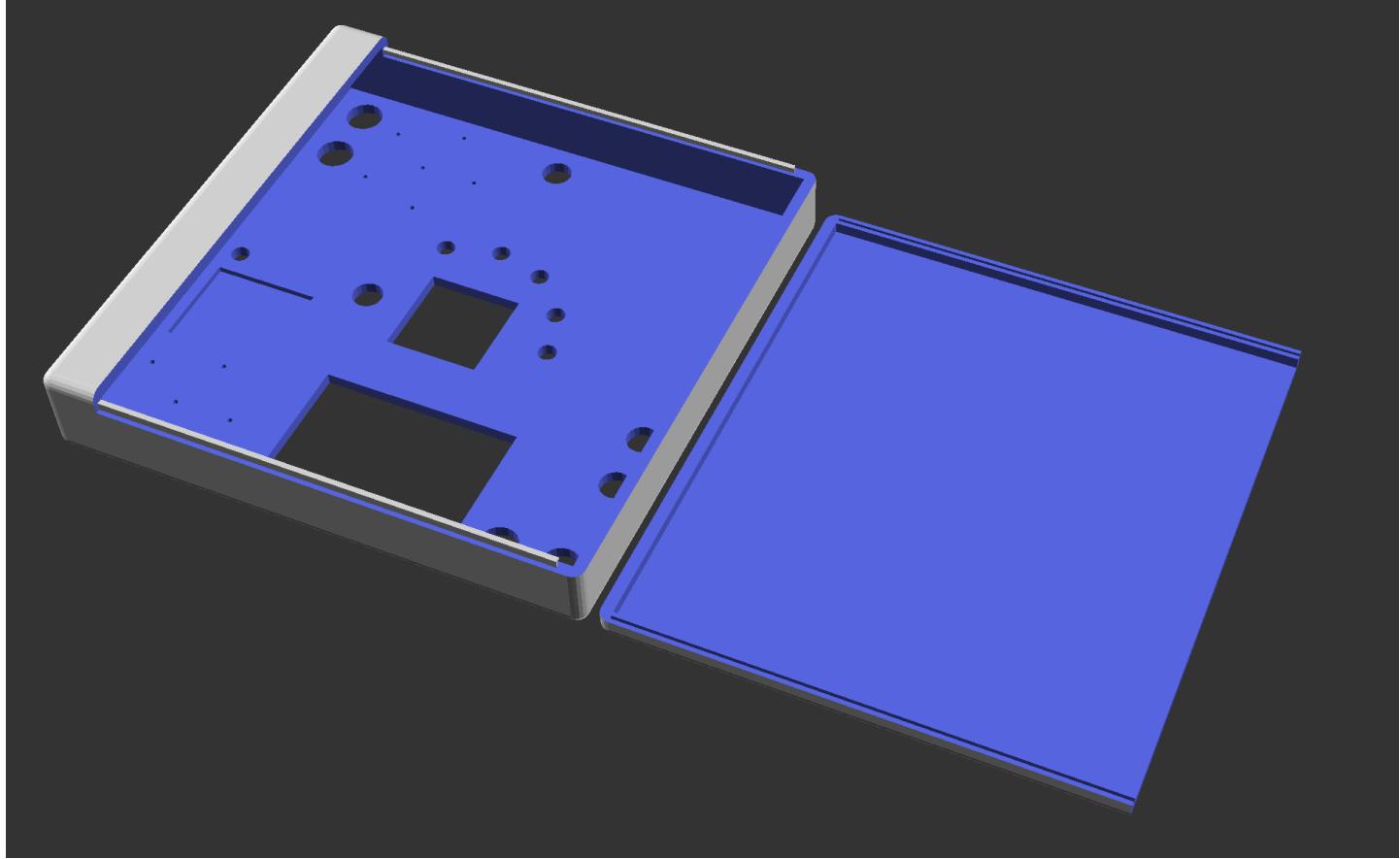


Figure 20 : modélisation du boîtier

La partie principale renferme le système au complet et est accompagné d'un couvercle pour fermer le tout.

Le couvercle se fermera via des rails pour le guider au long du boîtier. Ces rails consiste en une excroissance en forme de « L » inversé, placé au centre de deux parois, et creusée au même endroit dans le couvercle. Ainsi, le couvercle pourra se glisser dans les rails du boîtier pour la fermer.

d. Impression de la première version

L'imprimante 3D est équipée d'un plateau d'impression en verre. Il existe divers types de plateaux, chacun ayant ses avantages et ses inconvénients. Le plateau en verre possède pour principal avantage d'avoir une surface plane et lisse, permettant alors d'avoir ce même aspect sur la surface de la pièce imprimée en contact avec ce plateau. Cependant, c'est une surface possédant une faible adhérence. Pour permettre à la pièce de coller sur le plateau, il est nécessaire de mettre sur le plateau de la colle.

Un autre type de plateau existant est le plateau nano-texturé : il dispose d'une bien meilleure adhérence et donc ne nécessite pas de colle. De plus, ce plateau étant flexible, il est plus facile de retirer la pièce une fois celle-ci imprimée. Cependant, le plateau est rugueux et donc ne donnera pas un aspect lisse à la surface en contact avec le plateau.

Par exemple, sur le premier essai d'impression de la partie haute du boîtier, la colle a été mise très tôt et pas en assez bonne quantité. Lors de l'impression, la pièce n'a donc pas assez adhéré sur le plateau et s'est déformée :

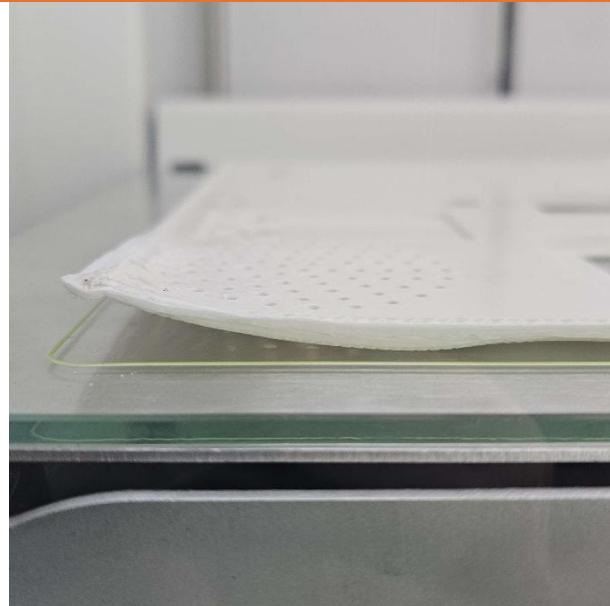


Figure 21 : impression ratée du boîtier

En mettant la colle juste avant de démarrer l'impression et en quantité suffisante, la pièce adhère et donc reste stable pour être imprimée :

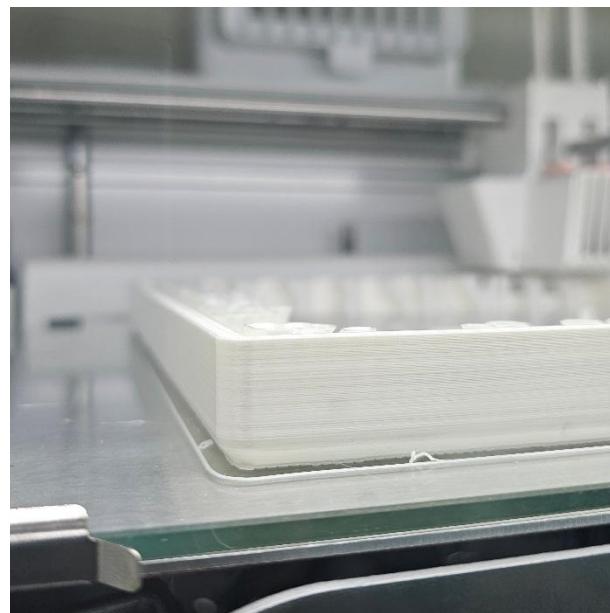


Figure 22 : impression en cours du boîtier

e. Première programmation pour tester les composants

Une étape importante du projet est de développer les programmes pour faire fonctionner les capteurs et actionneurs depuis la Raspberry et pour confirmer que ces éléments fonctionnent entre eux. Pour cela, j'ai monté le système hors d'un boîtier :

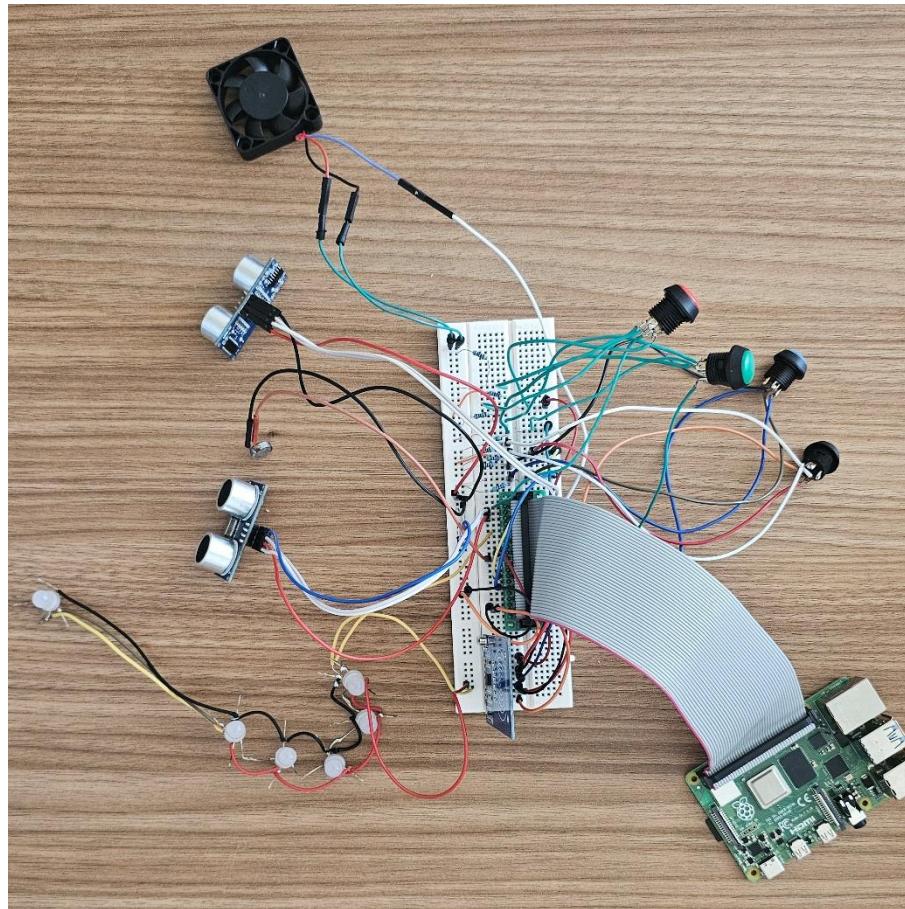


Figure 23 : système partiellement monté hors de son boîtier

Ensuite, j'ai réalisé des programmes simples, afin de tester le bon fonctionnement des composants un par un. Une fois que les composants fonctionnent seuls, j'ai développé un avant-dernier programme pour tester les composants ensembles mais étape par étape. Pour finir, j'ai réalisé un dernier programme testant tous les composants en même temps. Dans ce dernier programme, tous les capteurs fonctionnent ensemble. Ces programmes sont consultables en annexe 2.

Suite à cette période de premier montage et premiers tests, j'ai pu remarquer quelques défauts :

- Le logement du Raspberry Pi n'est pas correctement orienté et pas correctement fait. Il a été conçu pour les ports USB et RJ45 d'une Raspberry Pi5, or, on a changé de carte pour une Pi 4b pendant la modélisation.
- Les rails de fermeture ont une structure trop fine. Il faudra les repenser pour les rendre plus solide.
- L'écran prend beaucoup de place à l'intérieur du boîtier, à cause de ses câbles. Ces derniers sont tellement volumineux qu'ils ne m'ont pas permis de placer le télémètre juste à côté. De plus, son logement est légèrement trop petit et abîme les nappes, ce qui casse l'écran.

- Voir si le lecteur de cartes ne peut pas être retourné, pour optimiser la place prise par ce composant en interne.
- Trouver une version définitive du système de concentration de l'air du ventilateur pour l'intégrer directement dans le boîtier, car jusque-là, cette pièce était indépendante du boîtier.
- Les arrondis des angles au niveau du plateau d'impression ont été moins bien imprimés :



Figure 24 : angles arrondis du boîtier

- Le rail dispose de trois principaux défauts. Le premier est que l'excroissance est très fine, et donc facilement cassante. Le deuxième est que le jeu entre le boîtier et son couvercle est trop faible, provoquant trop de frottements entre les deux pièces et empêchant la fermeture complète. Le troisième est qu'une fois fermé, le couvercle n'a pas de manière de la laisser en position fermée. Il suffit de glisser le couvercle pour ouvrir le boîtier.

f. Amélioration du nouveau boîtier

1. Optimisation de la position des composants

Afin de permettre à tous les composants de rentrer dans le boîtier, il a fallu revoir l'emplacement de certains d'entre eux :

- L'écran prend trop de place en interne à cause de ses ports USB et HDMI, ce qui fait que le télémètre à côté n'a pu être inséré dans le boîtier. Pour résoudre le problème, l'écran a été retourné : ses ports sont désormais orientés vers le lecteur de cartes.
- Le lecteur de cartes et le haut-parleur à proximité ont échangé leur place afin de laisser plus de place au port de l'écran le plus volumineux : le port HDMI.
- L'ensemble ventilateur-LED-bouton jaune a été surélevé afin que le ventilateur ne soit plus au centre du boîtier, mais que les composants à proximité restent bien autour de lui. De plus, le système de compression d'air a été intégré au boîtier et n'est plus une pièce à part entière. Ici est utilisée la cinquième version de ce système.

- Le bouton bleu et le haut-parleur du dessous ont échangé leur position afin d'éloigner les deux haut-parleurs, permettant d'améliorer l'aspect stéréoscopique du son.

A gauche, la platine d'essai de la première version du boîtier. A droite, la nouvelle version du boîtier :

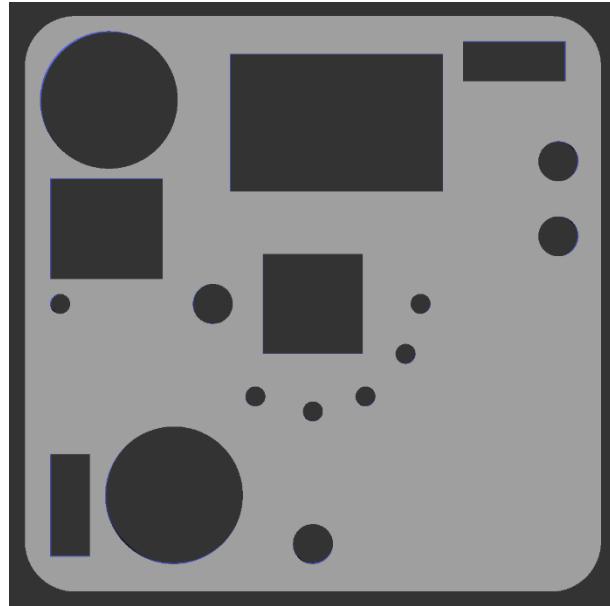


Figure 25 : platine de la première disposition des composants

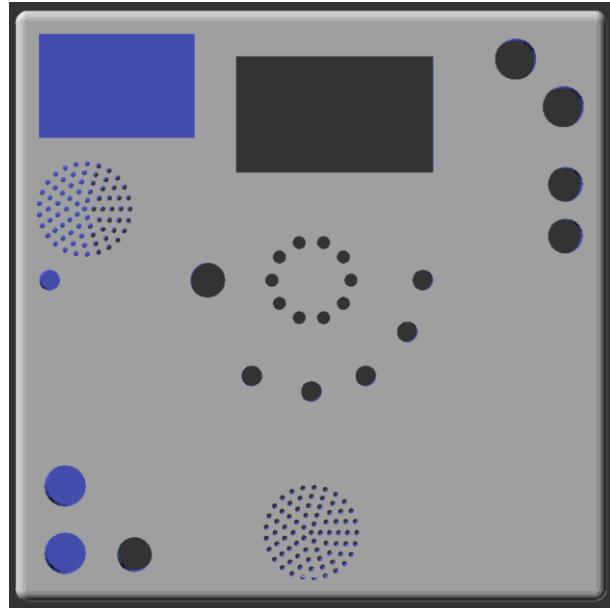


Figure 26 : modélisation de la deuxième version du boîtier, avec le nouvel agencement des composants

2. Modification du système de fermeture

Le premier rail est trop fragile à cause de sa finesse et le jeu est trop faible, ce qui fait qu'il y a trop de frottement et donc que le couvercle ne se renferme pas totalement. Ci-dessous, la conception du premier rail :

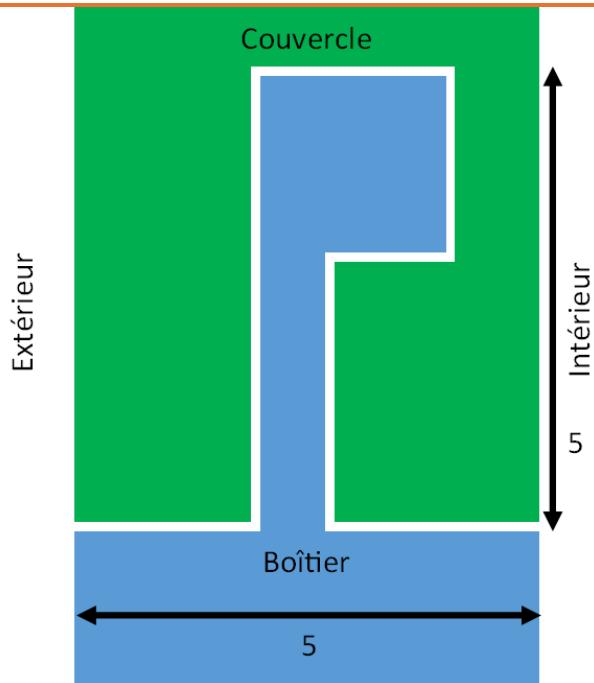


Figure 27 : système de rails de la première version du boîtier

La nouvelle configuration du rail utilise toute l'épaisseur du boîtier et de son couvercle, ce qui permet d'avoir des rails plus épais, et donc plus solides. Les deux excroissances sont identiques mais symétriques afin de simplifier la modélisation. Enfin, le jeu a été fixé à une valeur plus grande pour permettre un glissement total et facile.

Pour trouver le jeu idéal pour les rails, des pièces de ce type ont été imprimés. Il est difficile de l'observer ici, mais la forme des excroissances est différente. Le rail de gauche dispose d'un jeu de 0.4 millimètre tandis que les deux autres disposent d'un jeu de 0.3 millimètre. La différence entre les deux de gauche et celui de droite est le sens d'impression. Enfin, le logement présent sur les deux tests consiste en des logements pour essayer d'y loger le vibrEUR :

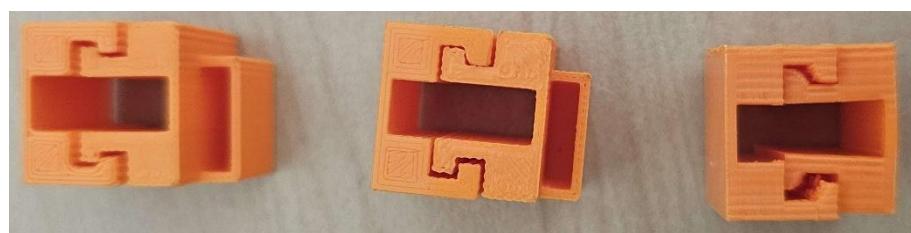


Figure 28 : petites pièces de tests des rails



Figure 29 : longue pièce de test des rails

Ci-dessous, la configuration avec les dimensions exprimées en millimètre permet de faire un système de rail occupant 5 millimètre de largeur et de hauteur tout en ayant un jeu de 0.3 millimètre :

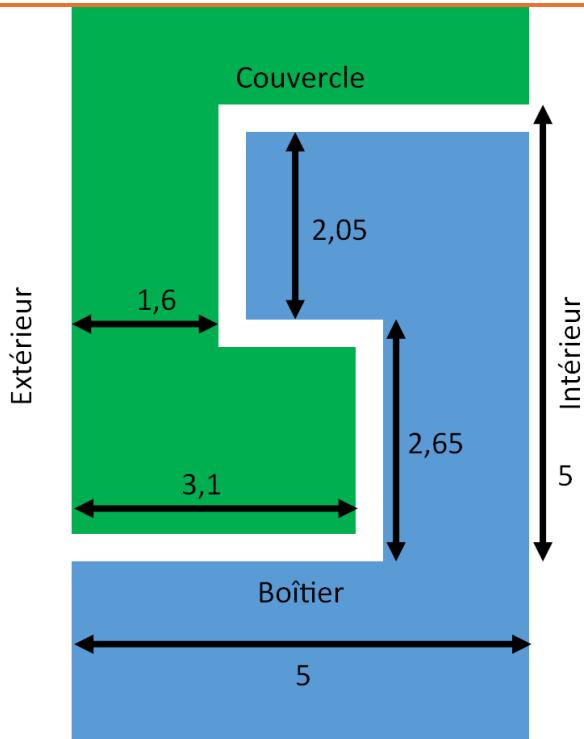


Figure 30 : système de rails de la deuxième version du boîtier

IV. Valorisation

a. Participation au treizième challenge handicap et technologie



Figure 31: préparation du stand pour le Challenge Handicap et Technologie

Lors de mon stage, j'ai pu participer au 13^e Challenge Handicap et Technologie, ayant eu lieu les 22 et 23 mai. Dans ce challenge, diverses personnes présentent leur projet pour aider le quotidien des personnes en situation de handicap. Parmi les exposants, nous trouvons des entreprises, des associations, des lycéens et des étudiants. Ces deux derniers groupes concourent pour quatre catégories. Chaque groupe étant inscrit dans une de ces catégories.

Les catégories sont :

- Communication et accessibilité numérique : permettre aux utilisateurs de mieux comprendre et de mieux se faire comprendre du monde.
- Inclusion : permettre aux utilisateurs de pouvoir faire tout ce qu'ils veulent et d'être intégré au monde.
- Autonomie : permettre aux utilisateurs de compenser la perte d'autonomie de leur situation.
- Handisport : permettre aux utilisateurs de pratiquer tous les sports possibles, sous toutes ses formes.

Pour présenter ScenaBox aux visiteurs, j'ai développé le jeu du Simon directement dans ScenaBox. Les règles du jeu ne changent pas, seulement la plateforme et le langage utilisé : ici, le jeu est programmé dans la Raspberry de ScenaBox en Python.

Ce challenge m'a permis de découvrir d'autres projets pour les personnes en situation de handicap, dont certains faits par des étudiants de l'université Paris 8. Il m'a également permis d'obtenir de l'avis de la part des visiteurs sur le projet. Parmi ces avis, beaucoup de visiteurs pensaient que ScenaBox est prédestiné à une utilisation personnelle, soit à la maison, plutôt qu'à une utilisation médicale.

b. Soumission d'un article de démonstration pour la conférence IHM

BlablaJe te donne plus d'information demain à ce sujet 😊

V. Annexe 1 : la modélisation 3D via des programmes

Lors de mon stage, j'ai suivi une formation en modélisation et impression 3D.

La manière la plus classique de modéliser est d'utiliser un logiciel de modélisation 3D. Cependant, l'équipe avec laquelle je travaille utilise une autre technique : la modélisation par un programme. Dans cette partie, je vais vous montrer les spécificités de la modélisation 3D par programme et de l'impression 3D en général.

En utilisant un programme codé en Python utilisant la librairie SolidPython2, nous pouvons créer des formes et les ajouter ou les supprimer afin de modéliser l'objet. Le principal avantage de cette technique est de permettre de changer rapidement les dimensions de l'objet à modéliser en utilisant des variables et aussi de pouvoir réutiliser facilement des pièces déjà modélisées simplement en copiant le code le modélisant.

Ci-dessous, la forme globale du programme :

```
from solid2 import * #Importer la librairie

def render_object(scad_object, out_scad_file=None, fa=1, fn=360):
    #Fonction pour créer le fichier scad
    if out_scad_file is None: #Si l'argument du nom du fichier scad n'est pas présent
        print(scad_render(scad_object)) #Faire un rendu de l'objet dans la console
    else:
        scad_render_to_file(scad_object, out_scad_file,
                            #Sinon, créer le fichier scad pour le voir sur un logiciel de visionneuse scad
                            file_header='$fa = ' + str(fa) + ';\n$fn = ' + str(fn) + ';')
        #En-tête du fichier contenant deux valeurs importantes pour les formes rondes

class Test:
    def __init__(self):
        #Ici, on y met les variables telles que des mesures spécifiques de l'objet
        self.ovl = 0.01
        #Cette variable permettra d'éviter des problèmes lors de la suppression de formes
        self.cylindre_hauteur = 10 #Exemple avec deux variables
        self.cylindre_diametre = 5

    def make_test(self) : #Ici, on construit l'objet
        Exemple = cylinder(d=self.cylindre_diametre,h=self.cylindre_hauteur)
        #Dans cet exemple, on met un cylindre dont le diamètre et la hauteur sont des variables
        return Exemple #Retourner la forme

if __name__ == "__main__":
    Final = Test() #Récupère l'objet Exemple créé ci-dessus
    render_object(Final.make_test(), "Exemple.scad")
    #Utilise la fonction pour créer un fichier scad où voir l'objet
```

L'exemple ci-dessus modélise un cylindre de diamètre 5 mm et de hauteur 10 mm :

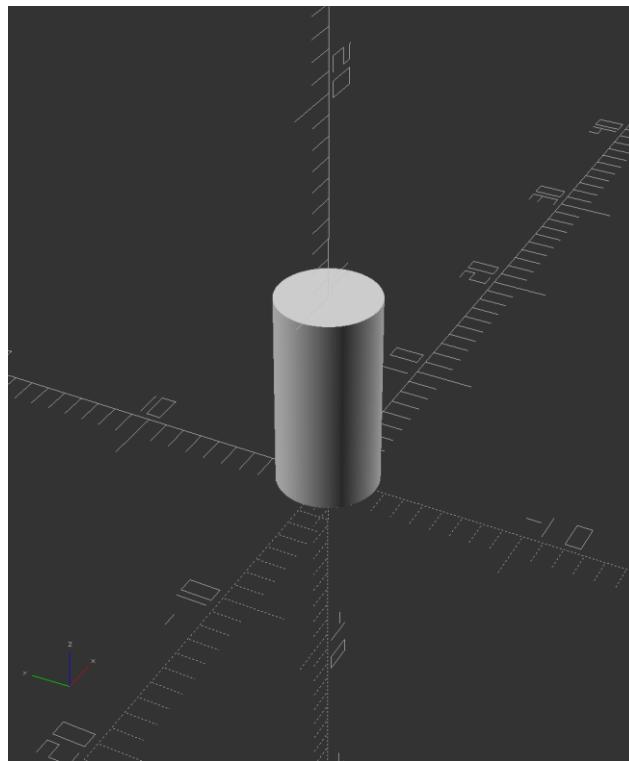


Figure 32 : un cylindre modélisé via programme

En modifiant la valeur des variables puis en exécutant à nouveau le programme, on peut alors recréer facilement l'objet avec les nouvelles dimensions. Ci-dessous, le cylindre vu du même angle mais avec une hauteur de 2 mm et un diamètre de 20mm :

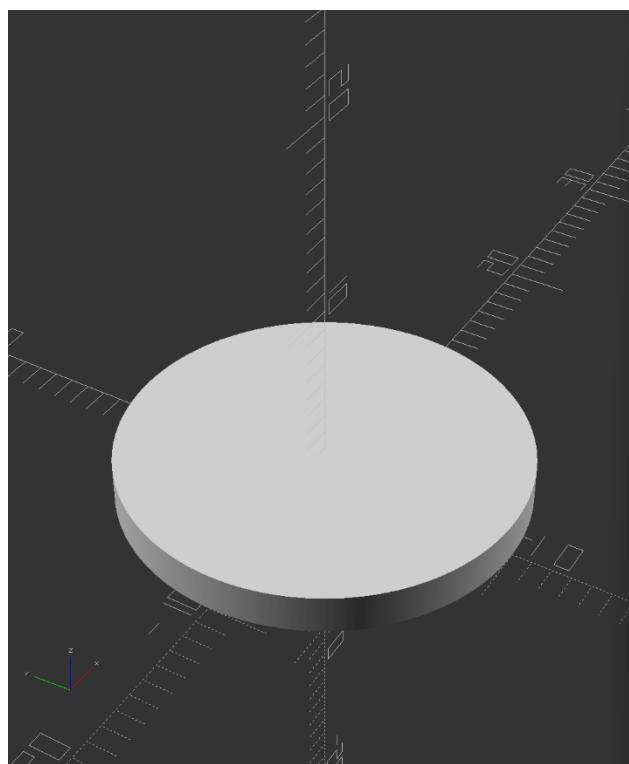


Figure 33 : en changeant la valeur des variables, la forme change de propriétés

On peut ajouter d'autres formes pour les combiner ou en supprimer pour faire des creux ou des trous.

Par exemple, on peut ajouter un cube puis en retirer un autre afin de creuser ce cube et former une esquisse de boîte :

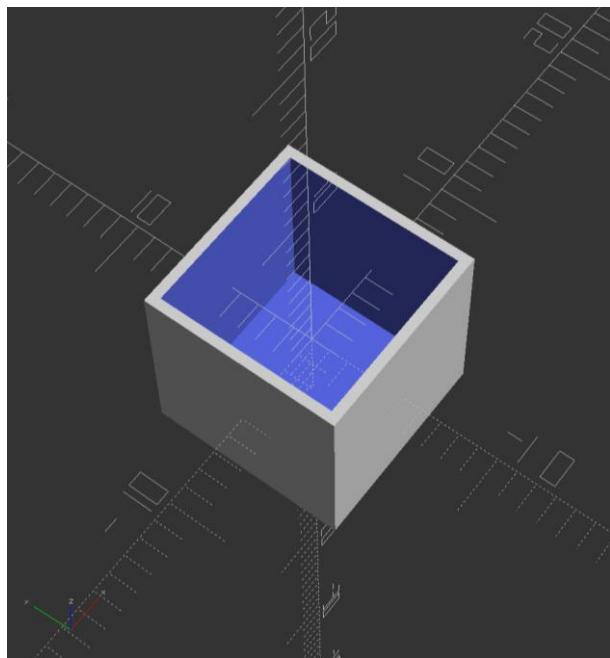


Figure 34 : cube creusé

```
Exemple = cube(self.boite_taille, center = True) #Commençons par un cube de la taille voulue
Exemple -= translate(0,0,self.boite_epaisseur)(cube(self.boite_taille-2*self.boite_epaisseur,
                                                 self.boite_taille-2*self.boite_epaisseur,self.boite_taille, center = True))
#Puis nous retirons un autre cube plus petit et translaté afin de creuser le premier.
```

Ici, nous remarquons différentes choses :

- On peut indiquer au cube soit une seule taille, soit trois tailles. Dans le premier cas, la taille indiquée sera appliquée aux trois dimensions.
- Ici, le second cube est translaté dans l'axe haut-bas afin de permettre de supprimer une face en plus de l'intérieur du premier cube.
- Les cubes possèdent l'argument « center = True », permettant de centrer les cubes sur l'origine. Si l'on met « Center = False » ou si cet argument n'apparaît pas, le cube apparaît avec la bordure à l'origine :

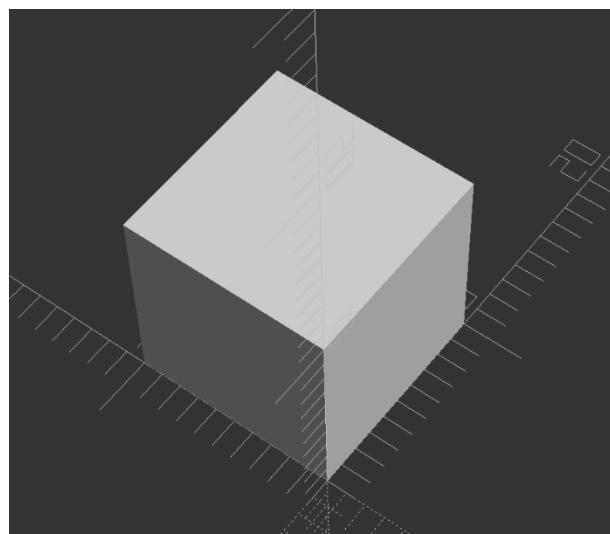
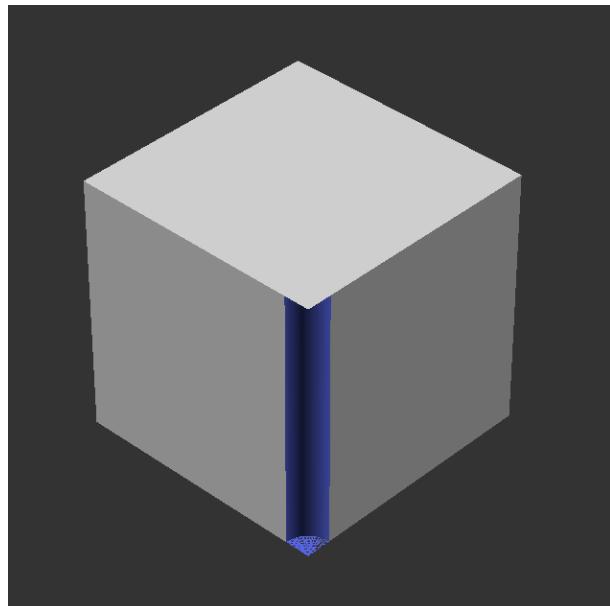


Figure 35 : cube sans argument "center"

Il y a également d'autres choses possibles : des sphères, des carrés, des cercles ou encore du texte.

Il est également possible de transformer des formes 2D en formes 3D en utilisant des extrusions :

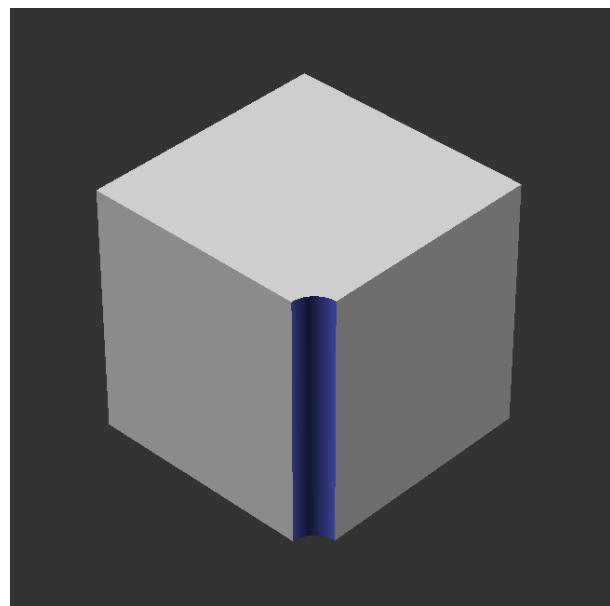
Il faut cependant faire attention lors de la suppression de formes, surtout si l'on veut retirer des formes sur le bord d'un autre. Si l'on veut retirer une forme à une autre qui se trouve à l'un des bords du premier, cela retirera le volume mais conservera la surface :



```
Exemple = cube(self.boite_taille)
Exemple -= cylinder(h= self.boite_taille
+2*self.ovl,d=2)
```

Figure 36 : cube creusé en volume mais pas en surface

Pour correctement modeler ce cube, il faut utiliser la variable « self.ovl », une valeur très petite, dans l'exemple, j'utilise une valeur de 0.001 millimètre, afin de retirer ici un cylindre un tout petit peu plus haut que le côté du cube, et déplacé un tout petit peu plus bas, pour pouvoir aussi supprimer les surfaces du cube :



```
Exemple = cube(self.boite_taille)
Exemple -= translateZ(-self.ovl)(cylinder(d=2,
h=self.boite_taille+2*self.ovl))
```

#On se décale d'une fois la variable ovl vers le bas afin de retirer la surface en trop en bas, puis on supprime un cylindre grand de la hauteur du cube + 2 fois ovl afin de retirer la surface en haut.

Figure 37 : cube correctement creusé

Une fois compilé, le programme convertit l'objet en un fichier .scad, permettant de l'observer sous tous les angles en utilisant le logiciel OpenSCAD. A chaque compilation du programme, OpenSCAD actualise l'objet sans nécessité de redémarrer le logiciel ou de rouvrir le fichier.

Une fois l'objet terminé, on demande à OpenSCAD de faire un rendu de l'objet afin de pouvoir générer par la suite un fichier d'impression STL :

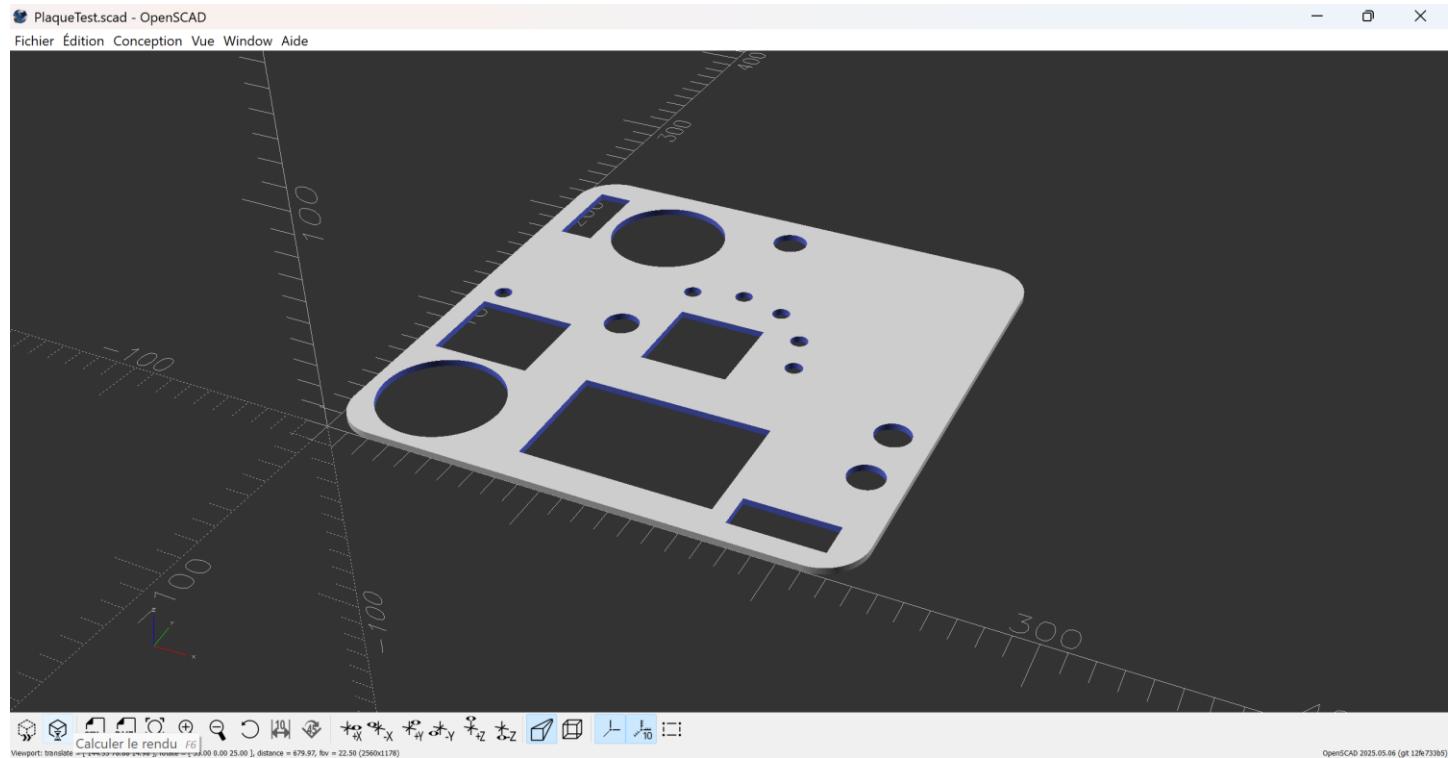


Figure 38 : vue d'ensemble du logiciel OpenSCAD, avec le bouton "Calculer le rendu" survolé

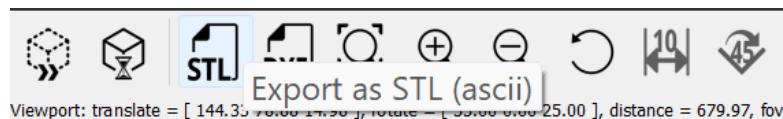


Figure 39 : barre d'actions de OpenSCAD, avec le bouton "exporter en STL" survolé

Ce fichier STL sera enfin utilisé sur un logiciel de type slicer, où l'on pourra paramétrier et lancer l'impression. En l'occurrence, j'ai utilisé le slicer officiel du constructeur de l'imprimante que j'utilise : UltiMaker Cura.

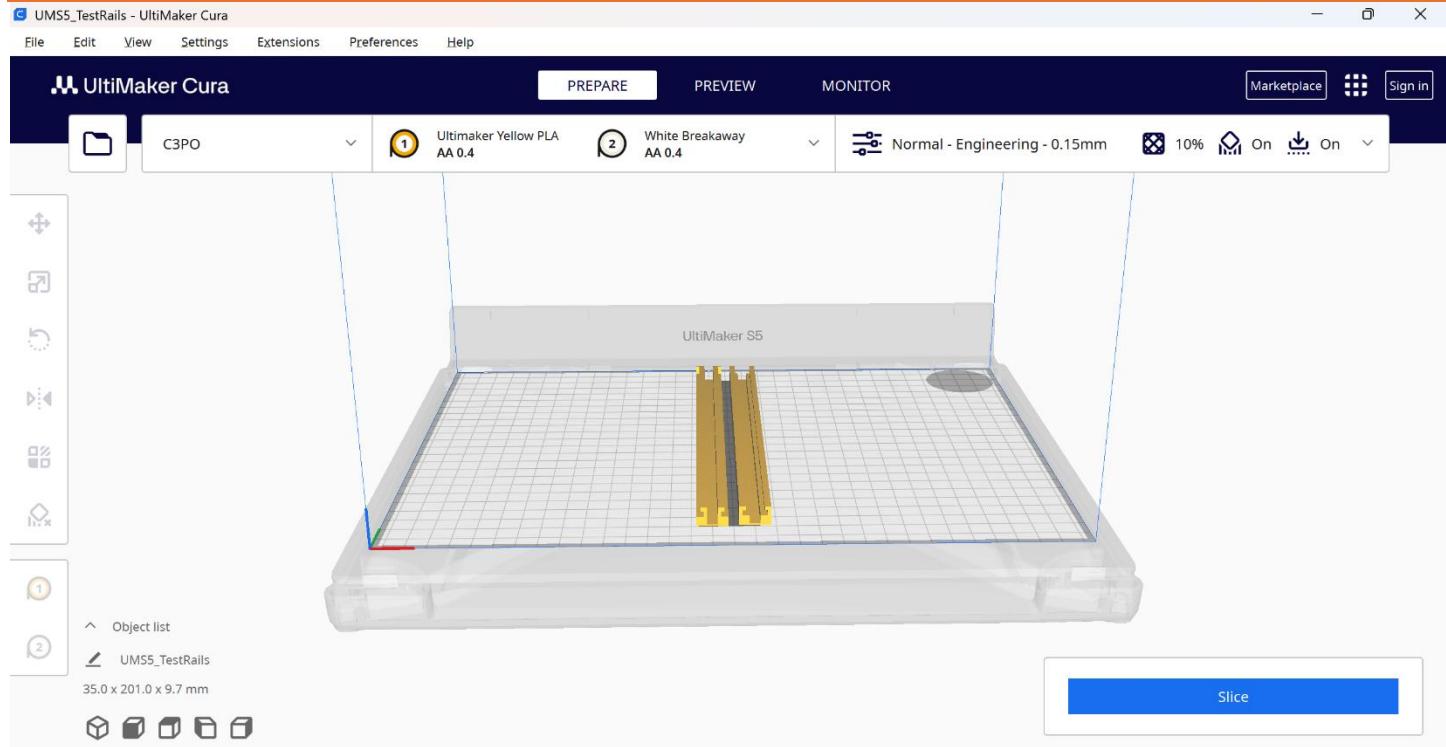


Figure 40 : vue d'ensemble du logiciel Ultimaker Cura

La capture ci-dessus montre le logiciel Cura, avec une pièce placée dans la zone imprimable de l'imprimante. On peut y observer plusieurs choses.

L'imprimante dispose de deux têtes d'impression. Cela permet, dans la plupart des impressions et dans l'exemple ci-dessous, d'imprimer avec le matériau d'impression sur la première tête et avec un matériau de support avec la seconde tête. Ce matériau de support permet d'imprimer avec une bonne qualité même si la pièce contient des parties dans le vide.



Figure 41 : exemple d'impression avec un matériau d'impression et un support

Les réglages ci-dessus indiquent que le matériau d'impression est du PLA jaune. Le PLA est le plastique le plus classique et le plus simple pour l'impression 3D. Le matériau secondaire est du Breakaway, soit un plastique spécifiquement conçu pour être utilisé comme matériau de support.

On peut aussi imprimer avec deux couleurs différentes, voire avec deux matériaux de pièce différents :



Figure 42 : exemple de choix d'impression d'une pièce avec deux couleurs différentes



Silver Metallic PLA
AA 0.4



Transparent PETG
AA 0.4



Figure 43 : exemple de choix d'impression d'une pièce avec deux matériaux différents

L'image ci-dessous correspond aux paramètres d'impression. Ces paramètres influeront sur la durée et la qualité d'impression :

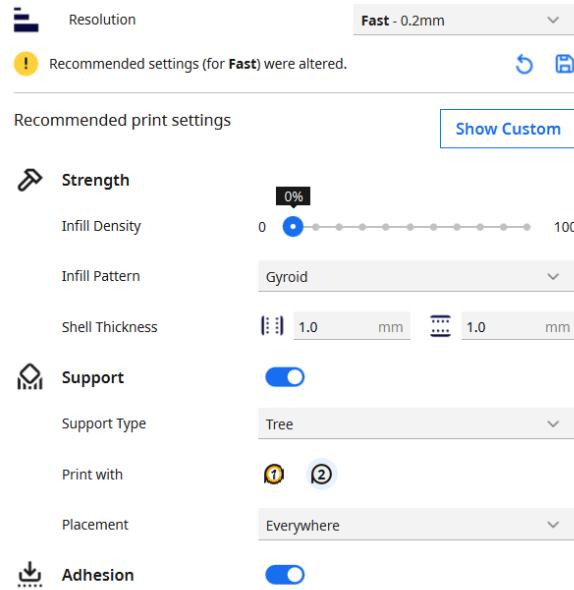


Figure 44 : les paramètres d'impression

Resolution : résolution d'impression. Une petite valeur donnera une impression plus précise mais plus lente.

Infill Density : taux de remplissage intérieur des pièces.

Infill Pattern : méthode de remplissage. Selon les volontés de solidité, il faudra changer la méthode. Gyroid est la méthode la plus polyvalente pour une solidité dans toutes les dimensions.

Shell Thickness : épaisseur des parois.

Support : utiliser ou non du support.

Support type : forme du support.

Print With : buse utilisée pour imprimer le support.

Placement : où placer le support.

Adhesion : utiliser de l'adhésion. Cette adhésion consiste en un ajout de matériau pour permettre aux premières couches de bien tenir sur le plateau.

Une fois les paramètres choisis, on peut alors demander au logiciel de calculer le protocole d'impression. On obtiendra alors une estimation de la durée d'impression et de la quantité de plastique utilisée :

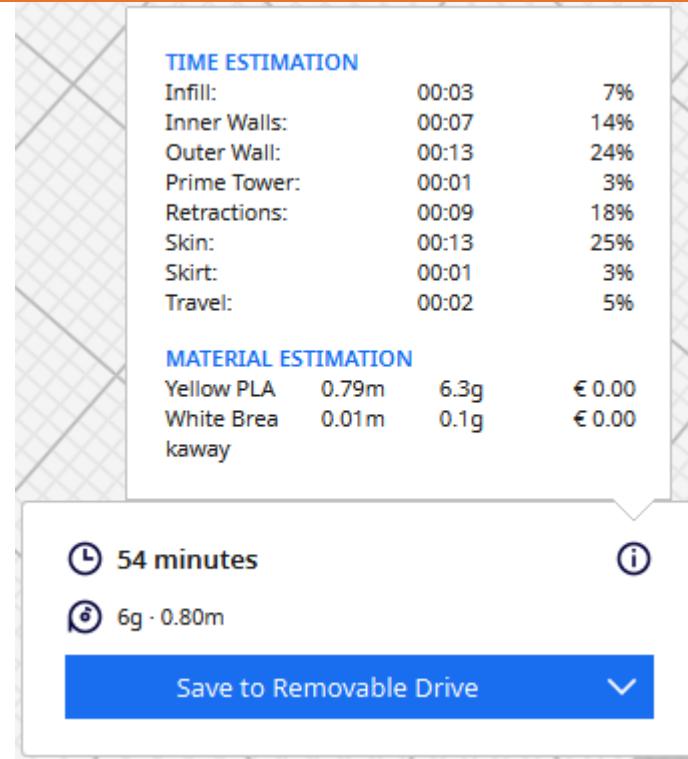


Figure 45 : estimation des coûts temporel et matériel de l'impression

Puis on peut voir un aperçu de la pièce. Les couleurs indiquent divers composants de la pièce :

- Rouge : parties en contact avec l'extérieur.
- Jaune : parois du haut et du bas.
- Vert : parois des murs.
- Bleu clair : adhésion et support.
- Orange : remplissage.
- Points blancs : départ des chemins.

L'écran « aperçu » permet également de voir la pièce couche par couche, comme montré en Figure 47.

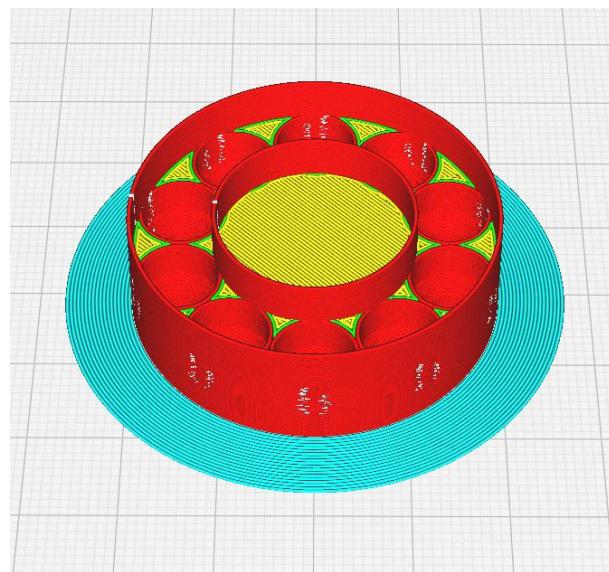


Figure 46 : aperçu d'une pièce entière avant impression

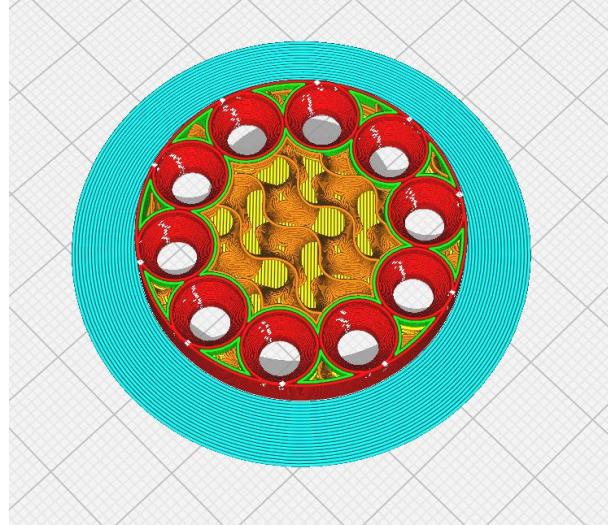


Figure 47 : aperçu de la couche 39 sur 87 de cette même pièce

Lorsque tout est prêt, on dispose de deux méthodes pour lancer l'impression :

- Enregistrer le fichier d'impression, de format UFP, dans un dispositif de stockage afin de connecter ce dernier à l'imprimante. Dans ce cas, on ne peut lancer l'impression que lorsque l'imprimante est au repos.
- Lancer l'impression à distance, si l'on peut se connecter à l'imprimante. Dans ce cas, si l'imprimante est occupée, l'impression sera envoyée en file d'attente.

Si l'on observe la Figure 45, le bouton propose la première méthode, car au moment de la capture d'écran, j'étais chez moi, donc je ne pouvais pas me connecter à l'imprimante.

VI. Annexe 2 : quelques programmes de modélisation 3D

Le programme ci-dessous est la modélisation de la plaque de test citée en partie III.a.

```
#Plaque sans rebords
PlaqueTest = cube(self.plaqueTaille-2*self.plaqueArrondi, self.plaqueTaille, self.plaqueEpaisseur, center = True)
PlaqueTest += cube(self.plaqueTaille, self.plaqueTaille-2*self.plaqueArrondi, self.plaqueEpaisseur, center = True)
#Rebords de la plaque
PlaqueTest += translate(self.plaqueTaille/2-self.plaqueArrondi, self.plaqueTaille/2-self.plaqueArrondi,
                      self.plaqueEpaisseur/2) (cylinder(r=self.plaqueArrondi, h=self.plaqueEpaisseur))
PlaqueTest += translate(self.plaqueTaille/2-self.plaqueArrondi, -self.plaqueTaille/2+self.plaqueArrondi, -
                      self.plaqueEpaisseur/2) (cylinder(r=self.plaqueArrondi, h=self.plaqueEpaisseur))
PlaqueTest += translate(-self.plaqueTaille/2+self.plaqueArrondi, self.plaqueTaille/2-self.plaqueArrondi, -
                      self.plaqueEpaisseur/2) (cylinder(r=self.plaqueArrondi, h=self.plaqueEpaisseur))
PlaqueTest += translate(-self.plaqueTaille/2+self.plaqueArrondi, -self.plaqueTaille/2+self.plaqueArrondi, -
                      self.plaqueEpaisseur/2) (cylinder(r=self.plaqueArrondi, h=self.plaqueEpaisseur))

#Déplacement de la plaque à l'origine
PlaqueTest = translate(self.plaqueTaille/2, self.plaqueTaille/2, self.plaqueEpaisseur/2) (PlaqueTest)

#Ventilateur
PlaqueTest -= translate(self.ventilateurX, self.ventilateurY, -
                      self.ovl) (cube(self.ventilateurCote, self.ventilateurCote, self.plaqueEpaisseur+2*self.ovl))

#Haut-parleurs
PlaqueTest -= translate(self.HP1X+self.HPCote/2, self.HP1Y+self.HPCote/2, -self.ovl) (cylinder(d=self.HPCote, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.HP2X+self.HPCote/2, self.HP2Y+self.HPCote/2, -self.ovl) (cylinder(d=self.HPCote, h=self.plaqueEpaisseur+2*self.ovl))

#RFID
PlaqueTest -= translate(self.RFIDX, self.RFIDY, -self.ovl) (cube(self.RFIDL, self.RFIDH, self.plaqueEpaisseur+2*self.ovl))

#Ecran
PlaqueTest -= translate(self.ecranX, self.ecranY, -self.ovl) (cube(self.ecranL, self.ecranH, self.plaqueEpaisseur+2*self.ovl))

#Ultrasons
PlaqueTest -= translate(self.US1X, self.US1Y, -self.ovl) (cube(self.USL, self.USH, self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.US2X, self.US2Y, -self.ovl) (cube(self.USH, self.USL, self.plaqueEpaisseur+2*self.ovl))

#Boutons
PlaqueTest -= translate(self.B1X+self.BD/2, self.B1Y+self.BD/2, -self.ovl) (cylinder(d=self.BD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.B2X+self.BD/2, self.B2Y+self.BD/2, -self.ovl) (cylinder(d=self.BD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.B3X+self.BD/2, self.B3Y+self.BD/2, -self.ovl) (cylinder(d=self.BD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.B4X+self.BD/2, self.B4Y+self.BD/2, -self.ovl) (cylinder(d=self.BD, h=self.plaqueEpaisseur+2*self.ovl))

#LEDs
PlaqueTest -= translate(self.L1X+self.LD/2, self.L1Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.L2X+self.LD/2, self.L2Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.L3X+self.LD/2, self.L3Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.L4X+self.LD/2, self.L4Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.L5X+self.LD/2, self.L5Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
PlaqueTest -= translate(self.L6X+self.LD/2, self.L6Y+self.LD/2, -self.ovl) (cylinder(d=self.LD, h=self.plaqueEpaisseur+2*self.ovl))
```

VII. Annexe 3 : les programmes de tests des composants et du Simon pour ScenaBox

Premier test des LED

```
import board
import neopixel
from time import sleep

pixels = neopixel.NeoPixel(board.D21, 6) #6 LED dont la broche Din de la 1ere est connectée au
#GPIO18

pixels.fill((0, 255, 0)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((255, 255, 0)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en
RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((255, 0, 0)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((255, 0, 255)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en
RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((0, 0, 255)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((0, 255, 255)) #Mettre toutes les LED en rouge pur (les LED sont en GRB, pas en
RGB)
sleep(0.5) #Attente d'une demi-seconde
pixels.fill((0, 0, 0)) #Eteindre toutes les LED
```

[Les autres seront ajoutées dans cette idée]

Programme de test global, étape par étape :

```
import RPi.GPIO as GPIO
import time
import board
import neopixel
from mfrc522 import SimpleMFRC522

reader = SimpleMFRC522() #Initialisation du lecteur de cartes

#Liste des GPIO des composants
BR = 5
BJ = 23
BV = 27
BB = 18
LEDR = 21
LEDJ = 3
LEDV = 6
LEDB = 2
trig = [19,16]
echo = [26,20]
Ventilo = 13
Vibreur = 25

#Préparation des GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setup(BR, GPIO.IN)
GPIO.setup(BJ, GPIO.IN)
GPIO.setup(BV, GPIO.IN)
GPIO.setup(BB, GPIO.IN)
GPIO.setup(LEDR, GPIO.OUT)
GPIO.setup(LEDJ, GPIO.OUT)
GPIO.setup(LEDV, GPIO.OUT)
GPIO.setup(LEDB, GPIO.OUT)
GPIO.setup(Ventilo, GPIO.OUT)
GPIO.setup(Vibreur, GPIO.OUT)
GPIO.setup(trig[0], GPIO.OUT)
GPIO.setup(trig[1], GPIO.OUT)
GPIO.setup(echo[0], GPIO.IN)
GPIO.setup(echo[1], GPIO.IN)

#Préparation des LED adressables
pixels = neopixel.NeoPixel(board.D12, 6, pixel_order=neopixel.GRB)

#Variables
EtatBouton = 0
FlagRouge = 1
FlagJaune = 1
FlagVert = 1
FlagBleu = 1
```

```
#Fonction pour mesurer une distance avec un télémètre
def measure_distance(ID):
    #Envoi d'un signal ultrasonique au télémètre voulu, pour lui faire faire la mesure
    GPIO.output(trig[ID-1], True)
    time.sleep(0.00001) # 10µs
    GPIO.output(trig[ID-1], False)
    start_time = None
    stop_time = None

    #Mesure du temps de retour. Plus le temps haut est long, plus la mesure est grande
    #Cette boucle détermine le moment où le télémètre renvoie un signal haut
    while GPIO.input(echo[ID-1]) == 0:
        start_time = time.time()
    #Cette boucle détermine le moment où le télémètre arrête d'envoyer le signal haut
    while GPIO.input(echo[ID-1]) == 1:
        stop_time = time.time()

    #Calcul de la durée et conversion en distance (vitesse du son = 34300 cm/s)
    time_elapsed = stop_time - start_time
    distance = (time_elapsed * 34300) / 2 # Diviser par 2 pour l'aller-retour

    return round(distance, 2)

#Premier test : boutons
print("test lecture broche. Appuyez sur les 4 boutons. Les boutons s'allumeront s'ils sont détectés appuyés")
while (FlagRouge or FlagJaune or FlagVert or FlagBleu) : #Appui sur les 4 boutons nécessaire
    if (GPIO.input(BR) == 1 and FlagRouge == 1) : #Si le bouton est appuyé pour la première fois
        FlagRouge = 0 #Changer la valeur du drapeau associé
        GPIO.output(LED_R, GPIO.HIGH) #Illuminer le bouton
        print("Bouton rouge validé") #Informer de la détection du bouton
    if (GPIO.input(BJ) == 1 and FlagJaune == 1) :
        FlagJaune = 0
        GPIO.output(LED_J, GPIO.HIGH)
        print("Bouton jaune validé")
    if (GPIO.input(BV) == 1 and FlagVert == 1) :
        FlagVert = 0
        GPIO.output(LED_V, GPIO.HIGH)
        print("Bouton vert validé")
    if (GPIO.input(BB) == 1 and FlagBleu == 1) :
        FlagBleu = 0
        GPIO.output(LED_B, GPIO.HIGH)
        print("Bouton bleu validé")
    print("Boutons détectés. Désormais, faites ctrl+c pour passer au test suivant.")
GPIO.output(LED_R, GPIO.LOW) #Eteindre les quatres boutons
GPIO.output(LED_J, GPIO.LOW)
GPIO.output(LED_V, GPIO.LOW)
GPIO.output(LED_B, GPIO.LOW)
try: #Attente avant prochaine étape
    while True:
        EtatBouton = 0 #Faire une action indéfiniment pour attendre
except KeyboardInterrupt: print("Test écriture broche. Vérifiez le bon fonctionnement du ventilateur et du vibreur.") #Quand l'utilisateur fait Ctrl+c
GPIO.output(Ventilo, GPIO.HIGH) #Démarrer le ventilateur et le vibreur
GPIO.output(Vibreur, GPIO.HIGH)
try:
    while True:
        EtatBouton = 0 #Attendre Ctrl+c pour continuer
except KeyboardInterrupt: GPIO.output(Ventilo, GPIO.LOW) #Ensuite, éteindre le ventilateur
GPIO.output(Vibreur, GPIO.LOW) #Et le vibreur
print("Test télémètre.")
try :
    while True:
        distance1 = measure_distance(1) #Lire constamment les télémètres
        distance2 = measure_distance(2)
        time.sleep(0.1)
        print(f"Distance Capteurs : {distance1};{distance2} cm") #Afficher les mesures
except KeyboardInterrupt : print("Test LED")
```

```
try :  
    while True :  
        for i in range(6): #Pour le test des LED, elles s'allument une à une en chenillard  
            pixels.fill((0, 0, 0)) #Eteindre toutes les LEDs  
            pixels[i] = (0, 0, 255) #Allumer une LED en bleu  
            time.sleep(0.1)  
  
        for i in range(4, 0, -1): #Retour en arrière  
            pixels.fill((0, 0, 0))  
            pixels[i] = (0, 0, 255)  
            time.sleep(0.1)  
    except KeyboardInterrupt: pixels.fill((0, 0, 0))  
    print ("Test final : RFID")  
    while True :  
        uid, text = reader.read() #Lire constamment le lecteur de cartes  
        print(f"Carte trouvée : {uid}; {text}")
```