

MASTER 1 INFORMATIQUE

Faculté des Sciences
et Sciences de l'Ingénieur
Rue André Lwoff
BP 573
56017 VANNES Cedex

Rapport de projet



Responsable d'UE : RAIMBAULT Frederic

Tutrice : GIBET Sylvie

Remerciements

Nous tenons tout d'abord à remercier notre tutrice, Sylvie Gibet, pour la proposition et le suivi de notre projet. Merci à Frederic Rimbault pour ses retours pertinents sur notre avancement. Nous sommes reconnaissants envers Tiago Brizolara, doctorant dans l'équipe expression d'Irisa, pour nous avoir donné un cours sur le logiciel PureData. Nous tenons à remercier Axel Jacomme pour avoir été présent au Virtual Lab. Merci à Charlotte Pelletier pour son aide sur les algorithmes de reconnaissance des gestes. Pour finir, nous remercions Mansour Tchenegon, actuellement étudiant en M2 Informatique, pour nous avoir aidé à démarrer le projet, en s'appuyant sur celui de l'an passé (SONIC 2).

Résumé

Notre projet s'intitule CONDUCT. Son but est de permettre de contrôler un son ou une mélodie avec les gestes des mains. Les différentes caractéristiques contrôlables sont le volume, le tempo, la fréquence et le vibrato. Le suivi des gestes est possible grâce à l'appareil Leap Motion, ainsi que le logiciel Unity pour le retour visuel et la calibration. Pour finir, la synthèse de son est effectué avec le logiciel PureData et l'instrumentalisation se fait avec le langage Faust.

MOT CLÉS : Reconnaissance de geste, synthèse de son, réalité virtuelle.

Summary

Our project is called CONDUCT. It aims to control a sound or a melody with hands gesture. The features we can control are the volume, the tempo, the frequency and the vibrato. The tracking of movements is possible thanks to a device named Leap Motion, and the software Unity for the visual and calibration feedback. Finally, the sound synthesis is made with the software PureData and the instrumentalization is made with the Faust language.

KEYWORDS: Gesture recognition, sound synthesis, virtual reality.

Table des matières

1.	Contexte	1
1.1.	Équipe de recherche	1
1.2.	Thématique.....	1
1.2.1.	Analyse et reconnaissance de gestes expressifs	1
1.2.2.	Synthèse de parole expressive	2
1.2.3.	Expressivité dans les textes, fouilles de textes	2
2.	Travail demandé.....	3
2.1.	Cahier des charges.....	3
2.2.	Spécification du sujet.....	3
3.	Travail réalisé.....	4
3.1.	Problèmes rencontrés et solutions effectués	4
3.1.1.	Reconnaissance de gestes	4
3.1.1.1.	Gestes statiques	5
3.1.1.2.	Gestes dynamiques	7
3.1.2.	Synthèse sonore	8
3.1.3.	Interface visuelle	11
3.2.	Produit réalisé.....	16
3.2.1.	Page d'accueil.....	17
3.2.2.	Enrichissement de la base de données de gestes.....	17
3.2.3.	Écran de jeu	18
4.	Glossaire	21
5.	Table des figures.....	22
6.	Bibliographie	23

1. Contexte

1.1. Équipe de recherche

L'Équipe de recherche *Expression* est basée en Bretagne, située dans les villes de Vannes, Lannion et Lorient. Elle est composée d'une cinquantaine de membres dont **Sylvie Gibet**, notre tutrice de projet.



Figure 1 - Logo équipe Expression, laboratoire Irisa

Cette équipe a pour grande objectif d'étudier le langage humain, que ça soit le geste, la parole ou le texte. Les grandes thématiques que l'équipe aborde sont « l'analyse synthèse et reconnaissance de gestes expressifs », la « synthèse de parole expressive » et l'Expressivité dans les textes, fouilles de textes », nous allons maintenant détailler ces thématiques.

1.2. Thématique

1.2.1. Analyse et reconnaissance de gestes expressifs

Cette thématique vise à tirer des caractéristiques de hauts niveaux des gestes, à partir de caractéristiques de bas niveau. On cherche à déterminer la signification d'un geste à partir de

données brutes capturées grâce à des capteurs utilisant des technologies de pointes. On pourra utiliser cette technologie dans les domaines de l'art, du jeu ou encore de la santé.

1.2.2. Synthèse de parole expressive

Dans ce cas, nous cherchons à régler le problème de la synthèse de parole classique, le manque d'expression de ces synthèses. L'équipe cherche donc à extraire d'un texte « l'expression » afin de l'utiliser pour synthétiser une voix expressive. On va ensuite chercher à évaluer ces synthèses. Les champs d'applications de la synthèse de parole expressive sont les médias, l'assistance ou encore des cas de la vie de tous les jours, tel l'interaction avec des objets connectés.

1.2.3. Expressivité dans les textes, fouilles de textes

On va ici chercher à extraire le sens dans les textes, afin de pouvoir mieux les comprendre, les appréhender, rendre l'implicite explicite. L'objectif de tout cela est de rendre la compréhension des textes plus aisées, de prendre des décisions plus rapidement.

Trois parties sont à distinguer dans cet objectif, le premier est d'acquérir les textes qui vont nous servir de sources, tout en le filtrant afin d'éliminer le bruit, ensuite nous allons extraire les caractéristiques de ces textes, extraire les informations dont nous avons besoin, puis pour finir, nous voulons représenter les résultats afin de les mettre en lumière, et les rendre utilisables.

2. Travail demandé

2.1. Cahier des charges

Ce projet s'inscrit dans les axes de recherche l'équipe Expression (<https://www.expression.irisa.fr/fr/>) qui concernent l'analyse et la synthèse des données expressives produites par l'humain (geste, son, image). Plus précisément, on s'intéresse ici au contrôle gestuel de processus de simulation sonore. Des travaux réalisés préalablement dans différentes équipes du laboratoire et dans le cadre de projets de master 1 (SONIC1 et SONIC2) ont permis de d'analyser et de synthétiser des gestes instrumentaux (percussion, violon) et des gestes de chef d'orchestre.

L'objectif final du projet CONDUCT est de contrôler et synthétiser des sons au moyen de la reconnaissance et du tracking de gestes en temps réel. On s'intéresse à un ensemble de gestes d'interprétation d'une mélodie donnée qui permettent de moduler de manière expressive les sons constituants de cette mélodie (par exemple variation de l'intensité, de la radiosité, de l'attaque, arrêt, variation du tempo, etc.).

2.2. Spécification du sujet

Les gestes seront d'abord capturés au moyen du système Leap Motion. La reconnaissance des gestes statiques (postures clés prédéfinies) sera réalisée par un algorithme d'apprentissage simple (K-NN ou SVM). La détection des gestes dynamiques saura réaliser un tracking simple ou une reconnaissance en temps réel du tracé du geste (aussi grâce à K-NN ou SVM). La synthèse sera réalisée grâce au framework Faust et au système de synthèse PureData. Le projet pourra être développé dans l'environnement Unity.

3. Travail réalisé

3.1. Problèmes rencontrés et solutions effectués

3.1.1. Reconnaissance de gestes

Pour contrôler l'application, deux types de gestes devaient être disponible :

- Des gestes statiques, ces gestes sont des configurations manuelles (main ouverte, fermée...)
- Des gestes dynamiques, ce sont des mouvements (mouvement de main en huit par exemple)

La reconnaissance de gestes nécessite un dispositif de capture, nous avons opté pour le Leap Motion. Ce périphérique est composé de deux caméras infrarouges ainsi que de trois LEDs infrarouges. Grâce aux images des caméras, le Leap Motion peut déterminer la position des objets qu'ils observent dans un espace 3D, ici les mains.



Figure 2 - Leap Motion

Pour mettre en œuvre les algorithmes de reconnaissance de gestes, nous avons besoin d'avoir accès aux positions des différentes parties de la main (les différentes phalanges, la paume...) nous utilisons le SDK fournies par Leap Motion pour cela.



Figure 3 - SDK Leap Motion

Grâce à cet asset, nous avons accès à chaque os de chaque doigt mais aussi, pour chaque partie de la main, aux caractéristiques qui la définissent, tel que sa position, sa rotation... etc.

Nous allons maintenant détailler les deux types de gestes.

3.1.1.1. Gestes statiques

Dans cette application, les gestes statiques vont être interprétés par la main gauche pour choisir un mode, tandis que la main droite va permettre de jouer avec de mode. Par exemple, on ferme le poing gauche pour contrôler le tempo de la musique puis on va contrôler le tempo

en bougeant notre main droite horizontalement (vers la gauche pour un tempo lent, vers la droite pour un tempo rapide).

Actuellement, l'application de reconnaissance de configurations manuelles reconnaît 5 gestes statiques différents :






Main ouverte	Poing fermé	Un doigt	Deux doigts	Lettre J
				

Figure 4 - Les différents gestes possibles

La main ouverte est la position « de base », elle permet de ne rien déclencher, le poing fermé permet de contrôler le tempo de la musique, un doigt le vibrato, deux doigts le volume et « lettre J » (dans la langue des signes) le pitch de la musique.

Pour reconnaître les différents gestes statiques nous utilisons l'algorithme des k plus proches voisins ([K-PPV](#)).

Pour déterminer la position d'une main, nous allons envoyer à l'algorithme K-PPV un point en 5 dimensions, chaque coordonnée étant la distance entre le bout d'un doigt de la main avec la paume de la main respective. Nous devons également lui envoyer le jeu de données sur lequel il doit se baser pour trouver les voisins. La distance utilisée entre la donnée dont on veut déterminer la nature et ses voisins est une distance euclidienne dans ce cas.

//TODO tests de fiabilité

3.1.1.2. Gestes dynamiques

Les gestes dynamiques seront effectués de la main droite. Chaque geste correspondra à une action spécifique.

Reconnaître des gestes dynamiques n'est pas aussi simple, en effet, quand un geste statique peut être détecté à un temps t , un geste dynamique s'effectue dans la durée, et sera toujours effectué sur une durée différente, et à une vitesse différente. Nous devons nous affranchir de la vitesse et de la durée du geste.

Nous allons donc envoyer à notre algorithme K-PP une série temporelle de points dans l'espace 3D (x,y,z) de la paume de la main, ce qui va former une trajectoire, celle de la paume de la main droite en l'occurrence. Mais au lieu d'utiliser une distance euclidienne pour trouver la distance entre le geste à déterminer et ces voisins dans l'espace, nous allons utiliser l'algorithme DTW (Dynamic Time Warping).

En effet l'algorithme DTW est un algorithme qui va mesurer la similarité entre deux séries temporelles qui ont une vitesse différente. Ce qui est parfait dans notre cas, car les gestes que nous exécutons ne seront jamais exécutés de la même manière, et cela varie encore plus d'une personne à l'autre, certains feront des gestes vifs, tandis que d'autres vont plutôt faire des gestes lents.

3 gestes dynamiques sont disponibles :

- Le « finish » qui permet de mettre le volume à 0 d'un seul geste, pour arrêter la musique.
- Le « huit » qui permet de gérer le tempo selon la vitesse du geste.
- Le « high » qui permet une augmentation subite du pitch.

//TODO test fiabilité

3.1.2. Synthèse sonore

Le premier obstacle rencontré était de pouvoir jouer une mélodie et de générer du son à l'aide de [PureData](#). La première option était de jouer d'un instrument synthétique et d'envoyer les notes jouées à PureData. Bien qu'il aurait été possible de le faire, cela aurait nécessité d'avoir deux utilisateurs : un pour jouer de l'instrument, et l'autre pour modifier les paramètres à l'aide du Leap Motion. De plus, il aurait été nécessaire de disposer d'un synthétiseur ou d'un logiciel permettant de jouer d'un instrument. La seconde option était de faire lire un fichier à PureData afin qu'il joue la mélodie, solution que nous avons retenue. Le format que nous avons retenu était le format [MIDI](#). D'une part parce que ce format est prévu pour être utilisé dans le domaine musical, et d'autre part car la librairie « Cyclone » nous mettait à disposition tous les outils dont nous avons besoin pour pouvoir utiliser les fichiers MIDI au sein de PureData.

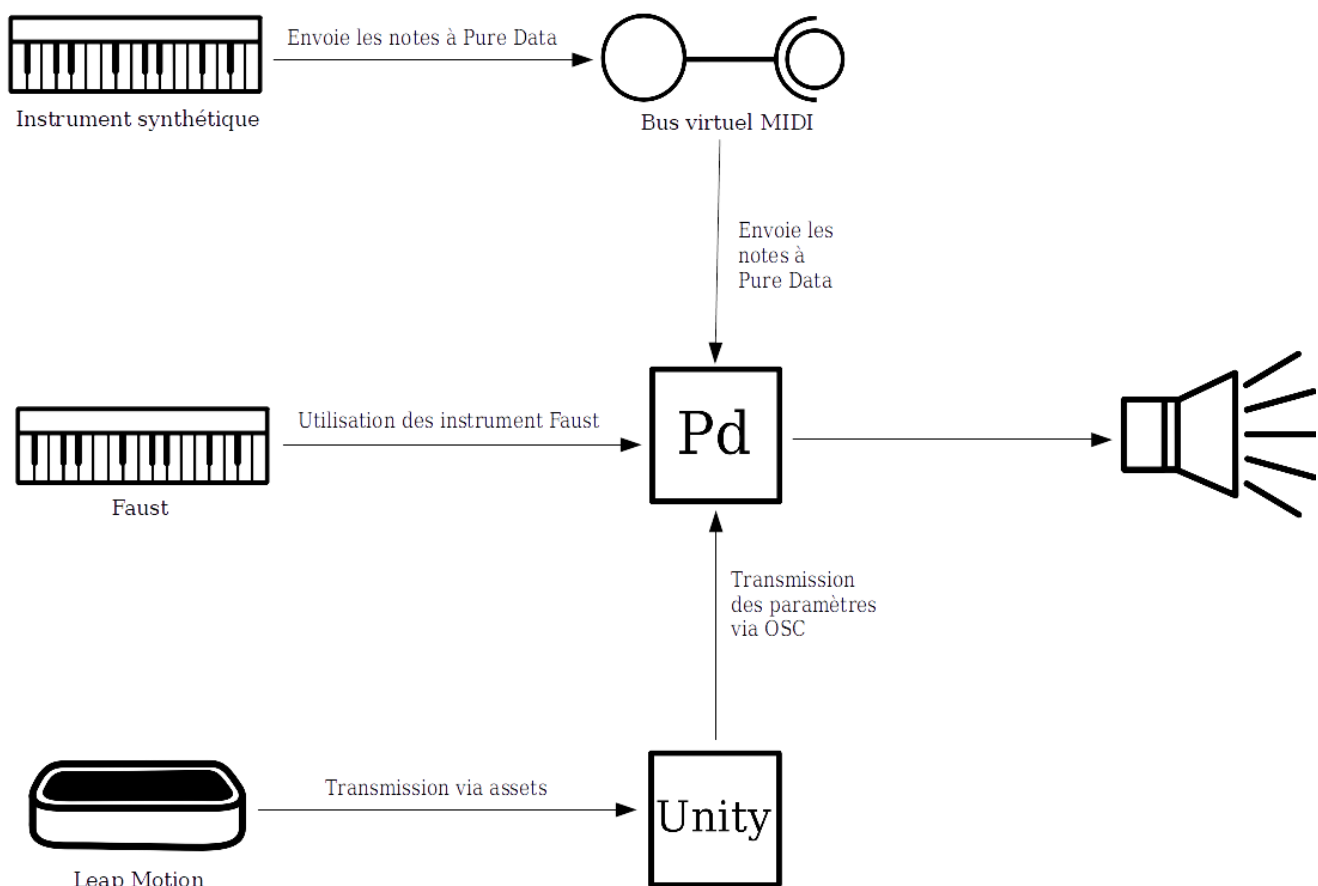


Figure 5 - Première méthode pour jouer une mélodie avec PureData

La seconde difficulté rencontrée était de modifier le tempo. Ce paramètre était probablement celui qui était le plus difficile à modifier. Nous avons 3 approches à notre disposition. La première était de relancer la lecture du fichier MIDI à chaque changement de tempo. Bien qu'il s'agît de la solution la plus simple, c'était aussi celle qui correspondait le moins à nos attentes. En effet, nous cherchions à changer le tempo en cours de lecture, et non de redémarrer la lecture à chaque changement. La seconde solution était de stocker toutes les notes dans un tableau, et d'ensuite lire ce tableau à la vitesse voulu à l'aide d'un métronome. Bien que cela nous aurait permis de changer le tempo en cours de lecture, cette approche était loin d'être satisfaisante. La première lecture du fichier ne permettait pas d'émettre du son. Bien que ce problème ne soit que peu dérangeant pour des mélodies courtes, il devient

bien plus handicapant pour des mélodies plus longue, vu que l'utilisateur n'aurait aucun retour lors de cette première lecture. De plus, cette méthode aurait drastiquement changé la mélodie, vu qu'il nous aurait été impossible de conserver, par exemple, les accords, ou le temps entre deux notes. La troisième approche, qui est celle que nous avons retenu, était de prétraiter l'information reçu puis d'envoyer cette information à l'objet « Seq » de la librairie « Cyclone ». En effet, cet objet, que nous utilisons pour la lecture de fichier MIDI, permet de changer le tempo en cours d'exécution. Bien que cette méthode fût la plus complexe à mettre en place, dû à la façon dont cet objet permet de modifier le tempo, elle était aussi la plus satisfaisante car elle permettait de changer le tempo en cours d'exécution tout en ne modifiant pas la mélodie.

Une autre difficulté était de choisir les paramètres à modifier. Tout d'abord, nous devions limiter leur nombre. Sachant que nous devions assigner un geste à chaque paramètre, un nombre trop important de paramètre aurait forcé l'utilisateur à se rappeler de trop de geste. Nous nous sommes donc arrêtés à 4 paramètres, car cela offre à l'utilisateur un contrôle suffisant sur la musique tout en limitant le nombre de geste qu'il doit retenir.

Après avoir décidé du nombre de paramètres, nous devions encore choisir les paramètres en eux-mêmes. Avec un nombre aussi limité, il était nécessaire de sélectionner les paramètres les plus pertinents, donc ceux ayant le plus d'effets sur le son. Les paramètres sont les suivants :

- Le volume
- Le tempo
- L'attaque
- Le vibrato (ce paramètre n'a un impact que sur le violon)

Au cours du développement, nous avons remarqués que l'attaque (qui était un paramètre des instruments généré avec [Faust](#)) n'avait aucun effet sur le son, peu importe sa valeur. Nous avons par conséquent décidé de le remplacer par le pitch (ou, fréquence).

3.1.3. Interface visuelle

Dès la première réunion avec Sylvie Gibet, nous avons fait le point sur les fonctionnalités manquantes dans le projet. Nous avons constaté qu'un point essentiel manquait : une réponse visuelle pour l'utilisateur.

Il fallait donc dans un premier temps, comprendre les mécaniques qui motivent l'utilisateur et qui semble le plus intuitif et cohérent à ce qu'il entend. Après plusieurs recherches, nous avons trouvé le phénomène neurologique appelé [synesthésie](#). Ce dernier, présent chez peu d'individus, leur fait associer des fréquences sonores à des couleurs. Nous avons donc décidé de baser notre interface visuelle sur cela.

Ainsi, les fréquences sonores graves sont représentées par des couleurs foncées, car souvent associées à des émotions tristes ou stressantes, les médiums sont de couleurs légèrement colorées et neutres, les aigus sont de couleurs vives. Le choix de ces couleurs n'est pas arbitraire et s'est effectué à partir du site [colors.co](#). Celui-ci permet de générer des palettes de couleurs harmonieuses, évitant ainsi d'avoir un rendu final peu esthétique.

Au total, il y a 7 couleurs, nous avons une gamme équilibrée avec 2 sombres, 3 neutres et 2 vives. Chaque couleur représente un son du plus grave au plus aigu, respectivement de gauche à droite.



Figure 6 - Palette de couleurs utilisées

Pour mettre en application cette palette, nous avons d'abord pensé à un mur en face de l'utilisateur qui affiche une couleur sur toute sa surface, cette couleur change en fonction de la fréquence du son joué. Le problème de cette solution est qu'elle est très peu stimulante, car assez monotone. Nous avons donc eu l'idée de faire des tâches de peintures, aussi nommées « splat ». Ces tâches apparaissent sur le mur, et s'effacent au bout d'un certain temps. Cela permet alors d'avoir quelque chose de colorés et dynamique.

Pour rendre ce modèle encore plus dynamique, nous avons choisi d'influencer les tâches avec les paramètres suivants :

- L'attaque : elle est retranscrite via un fade-in et fade-out des splats, plus l'attaque est faible, plus la durée de fade-in/fade-out est courte et inversement.
- Le volume : il influence la grosseur des tâches, plus le volume est bas, moins la tâche est grande et inversement.

- Le tempo : pour une petite valeur, les splats sont peu nombreux et vice-versa. Il influence également la durée d’affichage du splat, toujours sur le principe, plus la valeur est grande, plus ils s’affichent pendant longtemps.

Cependant plus tard dans le développement, il est apparu que l’attaque ne fonctionnait pas correctement. Ce paramètre a donc été éliminé du projet et remplacé par vibrato. La représentation visuelle de ce paramètre n’a malheureusement pas pu être intégrée par manque de temps.

L’implémentation de ce système sous Unity a nécessité l’utilisation de « [Decals](#) », qui sont « *des matériaux projetés sur des surfaces existantes* » [4]. Nativement, Unity n’a pas de système de decals, il a donc fallu utiliser un système gratuit disponible dans l’Asset Store nommé « Simple Decals System ». Ce système ne possède pas de documentation et très peu d’informations sur Internet, il a fallu un assez grand temps d’adaptation pour comprendre son fonctionnement. Il n’y avait pas d’alternative moins coûteuse en temps étant donné que c’est le seul asset gratuit permettant d’appliquer des decals. L’auteur de l’asset a rendu celle-ci utilisable uniquement dans l’éditeur, ce qui ne permet pas de voir les decals en testant le logiciel et empêche de build avec. Après avoir vérifié que modifier un asset n’enfreint aucune règle ou loi, nous avons modifié le code de l’asset afin de rendre le test et la compilation possible. Toutes les éditions ont été marquées par un « // Added by Alexy » ou « // Edited by Alexy » afin de rendre le traçage des modifications plus facile pour les équipes à venir.

Les splats ont été récupérés à partir d’images libre de droits sans attribution requise et dont la modification et l’utilisation ne sont pas restreintes. Avec un outil d’édition photo/vidéo, ici

AfterEffect, nous avons modifié les couleurs de chaque splat afin d'y appliquer celles de la palette.

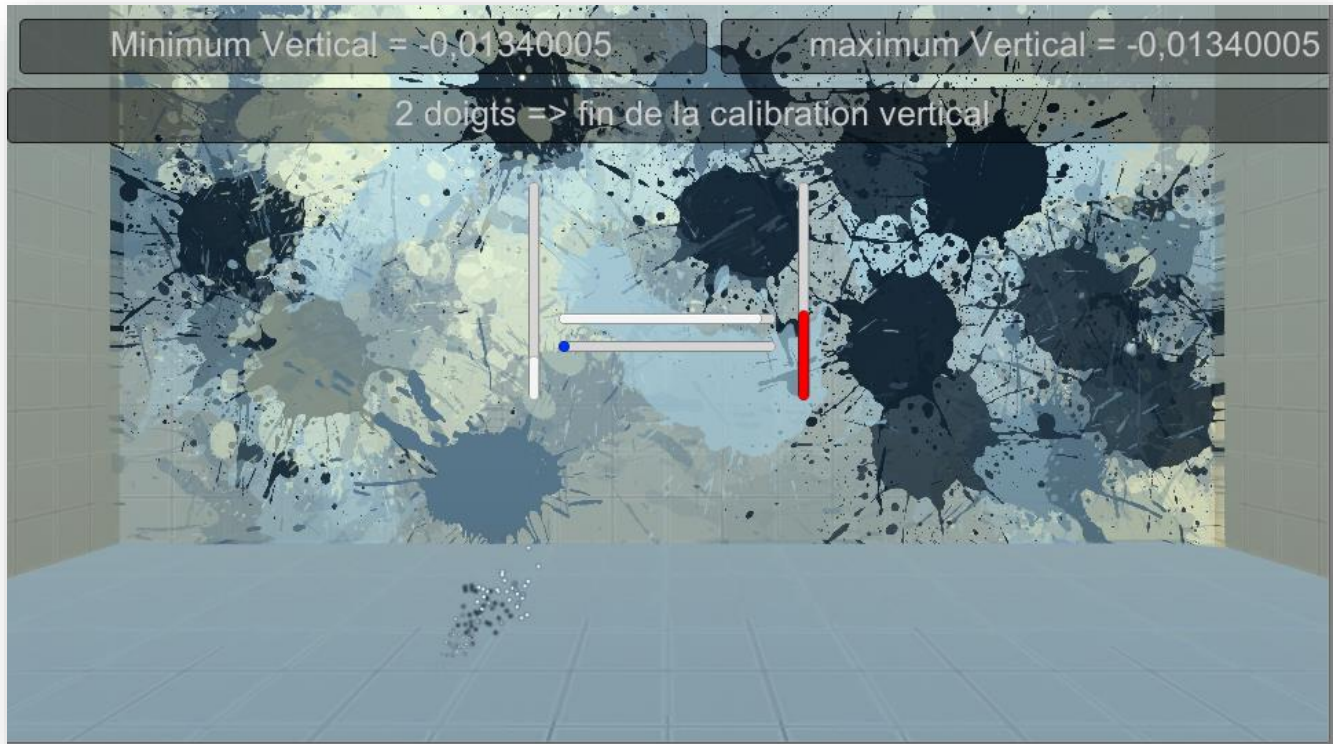


Figure 7 - Premier test d'affichage des splats

Pour ajouter une ambiance à la pièce qui semblait trop neutre, deux lumières ont été placées sur le plafond de la salle, leur couleur change en fonction de la fréquence jouée. C'est la même couleur que le splat qui est en train d'être peint. Pour plus de confort visuel et pour une future implémentation en réalité virtuelle, nous avons finalement choisi de remplacer le mur en face de l'utilisateur par un mur incurvé, ce qui est plus immersif.

Malgré ces ajouts, il manquait quelque chose pour que l'utilisateur puisse savoir quelle couleur il est en train d'utiliser, sans avoir à attendre qu'une tâche s'affiche. Nous avons donc attaché des particules aux poignets du joueur, elles permettent à la personne de voir en direct la couleur générée. Ces particules utilisent le système disponible dans Unity qui est un système physique.

Finalement, pour créer une identité propre au programme, nous avons dessiné un logo représentant deux mains entre lesquelles passent un arc de notes de musique, ce qui est représentatif de CONDUCT. L'image de note de musique est libre de droit pour toute utilisation et celle des mains nécessite une attribution à vecteezy.com.

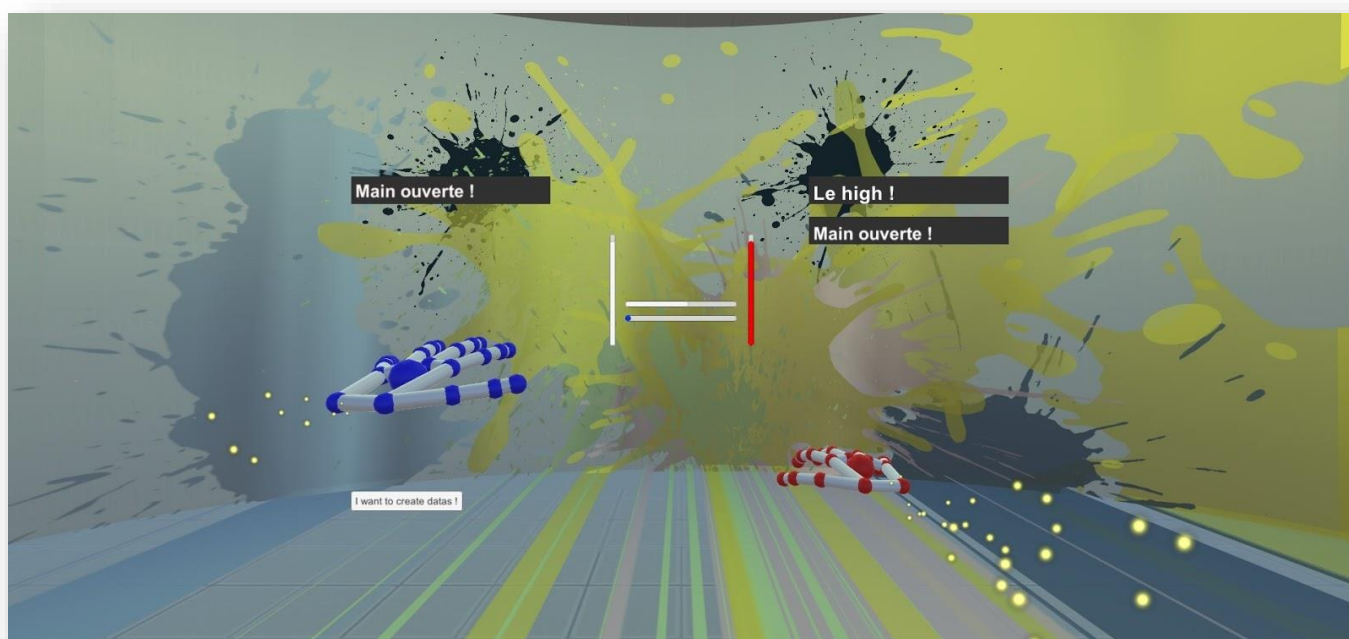


Figure 8 - Image en jeu

3.2. Produit réalisé

Le produit réalisé se divise en 3 écrans principaux.

Le schéma suivant explique le fonctionnement de l'application :

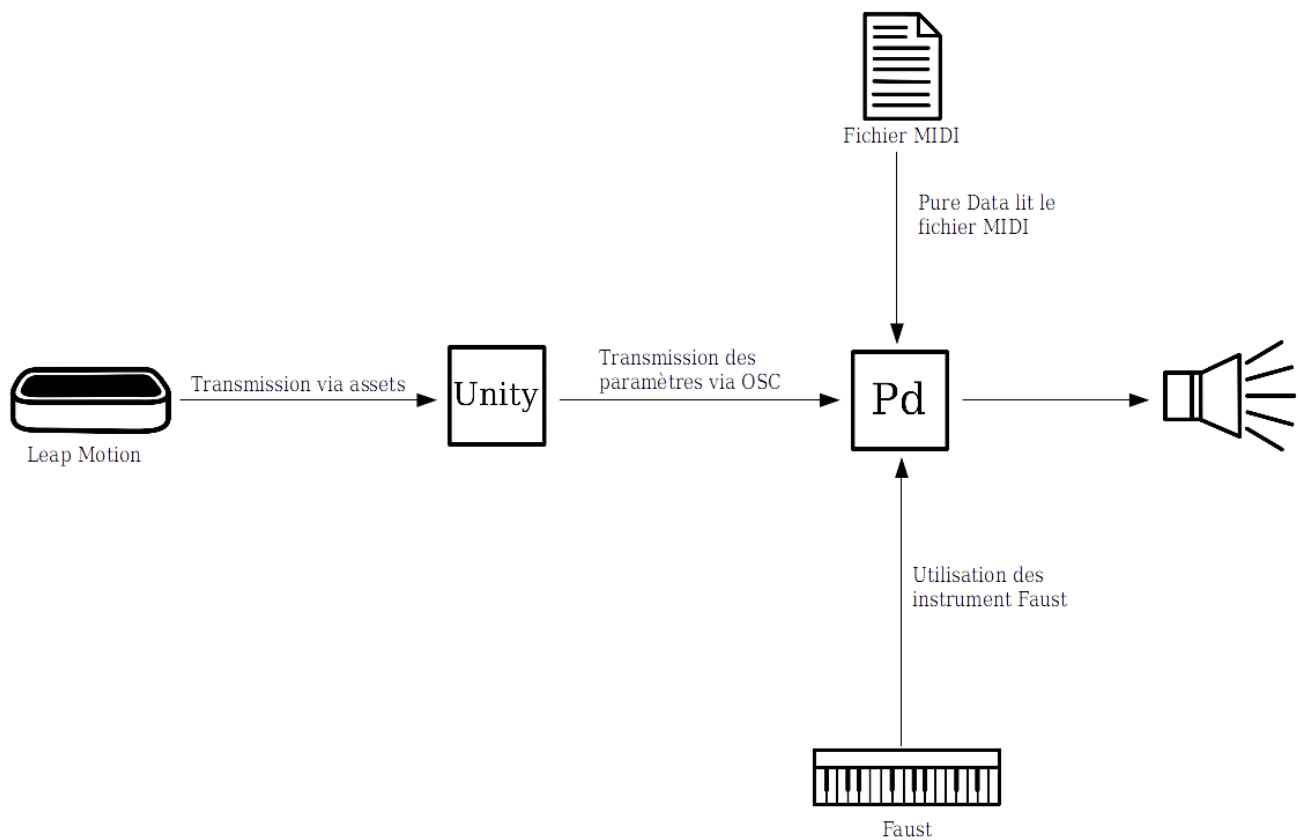


Figure 9 - Schéma de l'application

3.2.1. Page d'accueil



Figure 10 - Écran d'accueil

Quand on lance le produit, nous arrivons sur la page d'accueil, depuis cet écran, nous pouvons choisir entre enrichir la base de données de gestes, ou jouer. Il suffit donc de cliquer sur un des deux boutons pour lancer le mode correspondant, un délai de trois secondes est alors lancé, permettant à l'utilisateur de placer ses mains au-dessus du capteur Leap Motion.

3.2.2. Enrichissement de la base de données de gestes



Figure 11 - Interface d'enrichissement de la base de données

Pour enregistrer de nouvelles données dans la base de données de gestes, sélectionner le geste voulu, placer votre main (gauche pour les gestes statiques, droite pour les dynamiques), cliquer sur le bouton “record” tout en faisant le geste choisi, cliquez sur “stop record” quand vous avez fini le geste.

Lorsque vous ne créez pas de nouvelles données, il est possible de retourner vers l’écran de jeu.

3.2.3. Écran de jeu

Quand nous nous retrouvons sur l’écran de jeu, nous devons faire la calibration vertical et horizontal.

Pour faire la calibration verticale, mettre la main droite le plus haut que vous pouvez (sans sortir de l’écran), puis le plus bas possible, ensuite, faites le geste “deux doigts” avec la main gauche pour passer à la calibration horizontale.

Afin de procéder à la calibration horizontale, déplacer votre main droite le plus à droite possible (sans sortir de l'écran) puis vers la gauche (toujours sans sortir de l'écran), puis fermez le poing gauche pour terminer la calibration.

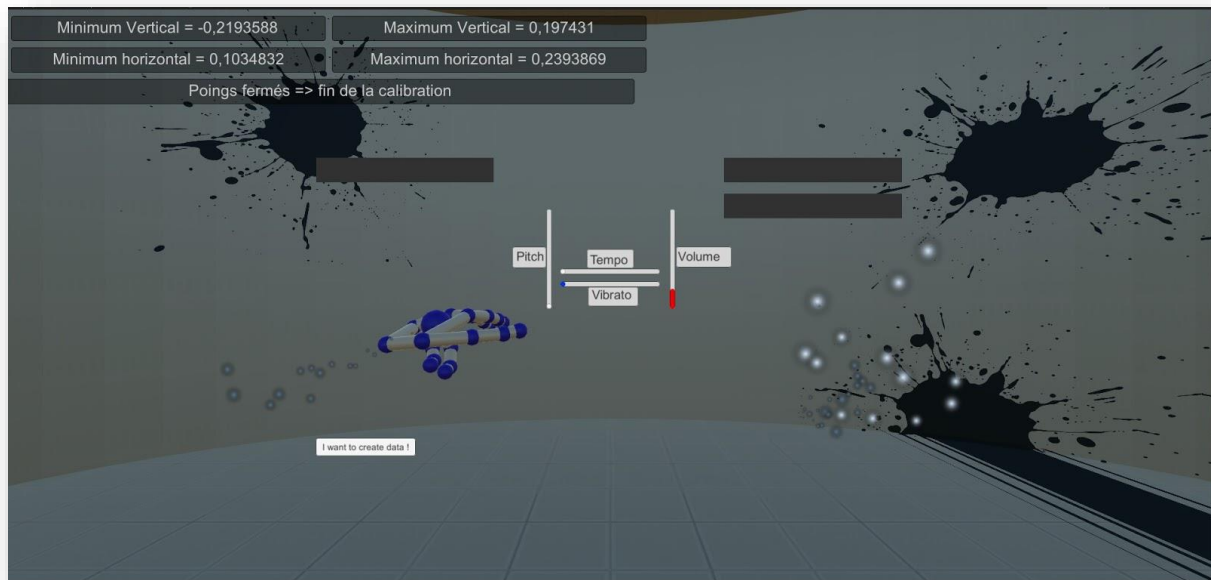


Figure 12 - Écran de calibration

Après avoir fini la calibration, nous pouvons passer à la « zone de jeu ».

C'est dans cette zone que nous pouvons interagir avec la musique grâce aux différents gestes. Les gestes statiques (de la main gauche) permettent de moduler certains paramètres tandis que les gestes dynamiques permettent des actions spécifiques sur les paramètres (par exemple couper le son brusquement).

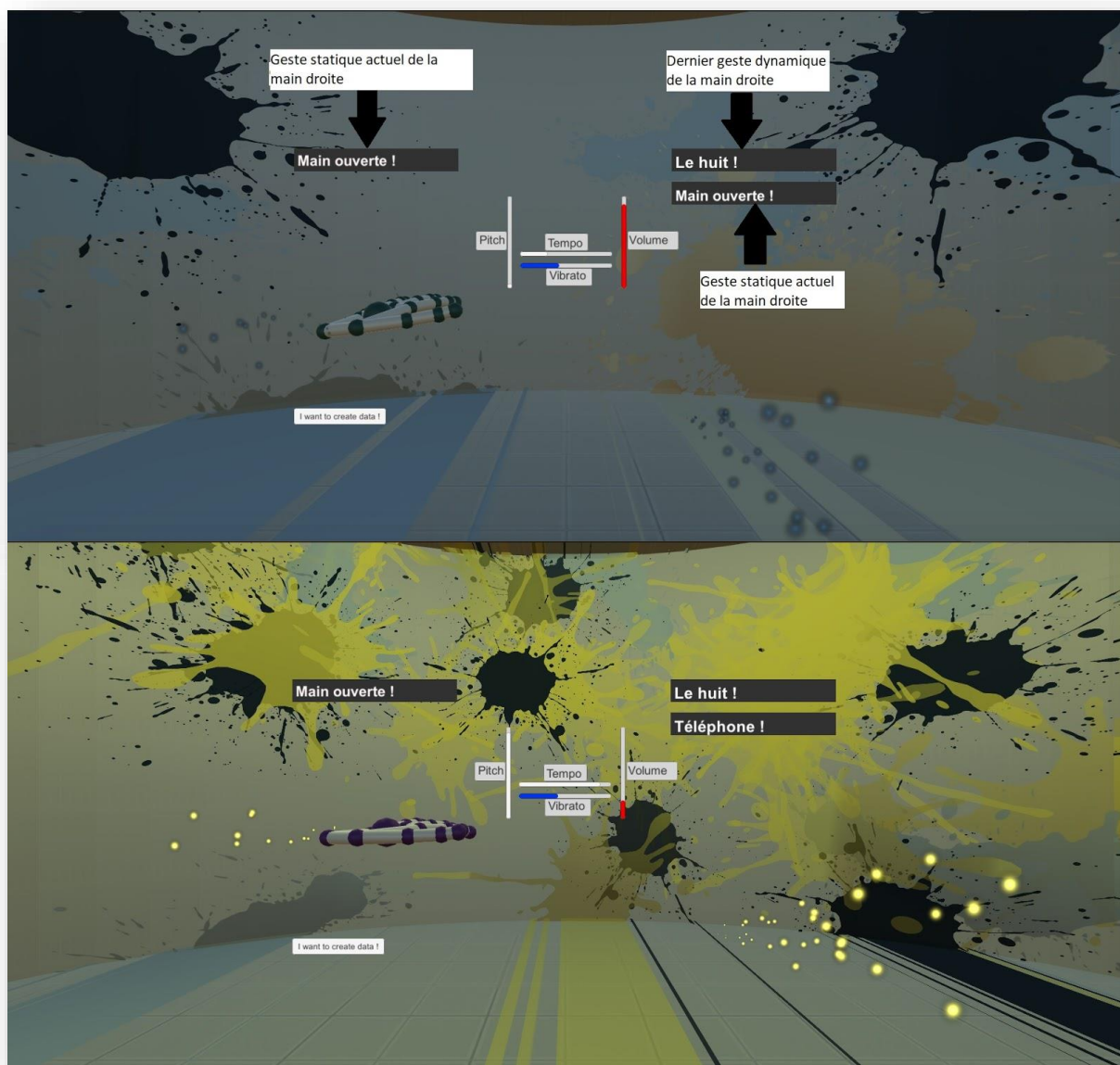


Figure 13 - Écrans de jeu

Sur l'écran de jeu, nous pouvons voir les différents gestes actuels, ainsi que des sliders indiquant le niveau de chaque paramètre. A chaque frame, les paramètres sont envoyés vers Pure Data via Open Sound Control, pour permettre de moduler le son.

4. Glossaire

[MIDI](#) : une interface de communication entre synthétiseur. Elle est principalement utilisée dans le monde de la musique, permettant l'échange d'information entre différents appareils. Les messages au format MIDI contiennent les informations sur les notes jouées, permettant ainsi à un synthétiseur de recréer la mélodie. Les fichiers MIDI, que nous utilisons dans notre projet, sont une suite de messages au format MIDI. [10]

[PureData](#) : un logiciel de programmation visuel. Il est principalement utilisé dans le domaine de la musique ou du traitement de sons. [5]

[Faust](#) : un langage de programmation fonctionnel axé sur la synthèse audio. Il permet notamment la création de synthétiseurs et d'instruments. [6]

[Algorithme K-PPV](#) : un algorithme d'apprentissage supervisé. Nous avons un jeu de données composées de m données de dimension R_d . Afin de déterminer la nature d'une nouvelle donnée (par exemple une position de main). Nous allons calculer la distance entre notre donnée et les données du jeu de données. La classe la plus présente parmi les k plus proches voisins de notre donnée dans l'espace sera la classe de notre donnée.

[Synesthésie](#) : expérience subjective dans laquelle des perceptions relevant d'une modalité sensorielle sont régulièrement accompagnées de sensations relevant d'une autre modalité, en l'absence de stimulation de cette dernière (par exemple audition colorée). [3]

[Decals](#) : Ce sont des matériaux projetés sur des surfaces existantes. [4]

5. Table des figures

Figure 1 - Logo équipe Expression, laboratoire Irisa	1
Figure 2 - Leap Motion	4
Figure 3 - SDK Leap Motion	5
Figure 4 - Les différents gestes possibles	6
Figure 5 - Première méthode pour jouer une mélodie avec PureData	9
Figure 6 - Palette de couleurs utilisées	12
Figure 7 - Premier test d'affichage des splats	14
Figure 8 - Image en jeu	15
Figure 9 - Schéma de l'application	16
Figure 10 - Écran d'accueil.....	17
Figure 11 - Interface d'enrichissement de la base de données	18
Figure 12 - Écran de calibration	19
Figure 13 - Écrans de jeu	20

6. Bibliographie

[1] A. Bouënard, S. Gibet, M. Wanderley (2012). Hybrid Inverse Motion Control for Virtual Characters Interacting with Sound Synthesis - Application to percussion motion. *The Visual Computer* 28(4): 357-370, 2012

[2] Lei Chen, Sylvie Gibet. CONDUCT: an expressive conducting gesture corpus for sound control, LREC 2018, Japon, May 2018

[3] *Larousse* [en ligne] [consulté le 8 mai 2020] :
<https://www.larousse.fr/dictionnaires/francais/synesth%C3%A9sie/76181>

[4] *Valve Developer Community* [en ligne] [consulté le 8 mai 2020] :
<https://developer.valvesoftware.com/wiki/Decals>

[5] *Site de PureData* [en ligne] [consulté le 9 mai 2020] : <https://puredata.info/>

[6] *Site de Faust* [en ligne] [consulté le 9 mai 2020] : <https://faust.grame.fr/>

[7] *Dépôt GitHub de Cyclone* [en ligne] [consulté le 9 mai 2020] :
<https://github.com/porres/pd-cyclone>

[8] *Site de Max* [en ligne] [consulté le 9 mai 2020] : <https://cycling74.com/products/max/>

[9] *Documentation de MSP* [en ligne] [consulté le 9 mai 2020] :
<https://docs.cycling74.com/max5/tutorials/msp-tut/mspaudioio.html>

[10] Jim Heckroth, Crystal Semiconductor Corp : The Complete MIDI 1.0 Detailed Specification, The MIDI Manufacturers Association, Avril 2006,
<https://www.midi.org/specifications-old/item/the-midi-1-0-specification>

Logo des mains par vecteezy.com

ANNEXES