

# RAPPORT DE PROJET

Cokila ILANGOVANE

Laura KOUADIO

Marie BOONE

Ismail JORHOUMI

Abderrahim ERRAMI

Octobre 2020 - Avril 2021

## SYNTH'EASY QUEST

---

### SOMMAIRE

<b>1 INTRODUCTION</b>	<b>3</b>
<b>2 Démarche DESIGN THINKING : Le Vintage c'est pour le grand âge</b>	<b>4</b>
2.1 Brainstorming . . . . .	4
2.2 Quelles idées sont apparues ? . . . . .	5
2.3 Approfondir les paradoxes . . . . .	5
<b>3 Notre projet</b>	<b>7</b>
3.1 Principe du jeu . . . . .	7
3.2 Implantation du projet dans le marché . . . . .	8
3.3 Le concept du jeu . . . . .	8
3.3.1 En quoi ce jeu est-il une innovation ? . . . . .	9
3.4 Scénario initial . . . . .	9
3.5 Scénario développé et plus détaillé (MARIE) . . . . .	10
<b>4 Développement de l'environnement de jeu sur Unity3D (ISMAIL)</b>	<b>15</b>
4.1 Introduction . . . . .	15
4.2 Caméra . . . . .	15
4.3 Personnages . . . . .	15
4.4 Décor . . . . .	15
4.5 Mise en place de la Réalité Virtuelle . . . . .	16
4.6 Difficultés rencontrées . . . . .	17
4.7 Améliorations et prochaines démarches . . . . .	18
<b>5 Création du clavier réel (LAURA)</b>	<b>19</b>
5.1 Introduction . . . . .	19
5.2 Assemblage hardware . . . . .	19
5.3 Fonctionnement du capteur . . . . .	20
5.4 Activation du capteur . . . . .	20
5.5 Réglage des valeurs seuils pour deux canaux . . . . .	21
5.6 Utilisation des valeurs recueillies par les capteurs . . . . .	22
5.7 Le serveur UDP . . . . .	23

5.8	Difficultés rencontrées . . . . .	23
5.9	Améliorations et prochaines démarches . . . . .	23
5.10	Instructions pour la mise en route du serveur . . . . .	23
<b>6</b>	<b>Établissement de la connexion entre le clavier réel et le clavier virtuel (AB-DERRAHIM)</b>	<b>25</b>
6.1	Introduction . . . . .	25
6.2	Création du serveur client . . . . .	25
6.3	Récupération de variable sur Unity3D . . . . .	25
6.4	Difficultés rencontrées . . . . .	25
6.5	Améliorations et prochaines démarches . . . . .	25
<b>7</b>	<b>Création du synthétiseur virtuel sur Unity3D</b>	<b>26</b>
7.1	Introduction . . . . .	26
7.1.1	L'oscillateur . . . . .	26
7.1.2	L'enveloppe . . . . .	28
7.1.3	Le filtre . . . . .	30
7.2	Design du clavier (MARIE) . . . . .	32
7.2.1	Création du clavier à l'aide Game Objects . . . . .	32
7.2.2	Changement de couleur lors d'un appui sur une touche . . . . .	33
7.3	Implémentation des notes, des paramètres et des oscillateurs (COKILA) . . . . .	35
7.4	Difficultés rencontrées . . . . .	38
7.5	Améliorations et prochaines démarches . . . . .	38
<b>8</b>	<b>ANNEXES</b>	<b>39</b>
8.1	Annexe 1 : Bibliographie des tutoriels et guides suivis . . . . .	39
8.2	Lien GitHub . . . . .	40
<b>9</b>	<b>Remerciements</b>	<b>41</b>

# 1 INTRODUCTION

N'avez vous jamais eu envie d'apprendre à manier un instrument de pointe tel que le théremine, le dualo-du-touch ou encore un synthétiseur ?

Et ne vous souvenez vous pas d'avoir abandonné cette idée par manque de temps, ou par pur manque de motivation ?

Le fait est qu'il ne devrait pas être si compliqué d'apprendre à jouer d'un instrument. Nous entendons par là non pas le fait de savoir jouer, mais simplement le fait de se donner les moyens et la motivation d'apprendre.

La technicité freine les novices, l'implémentation de fonctionnalités high tech freine les joueurs d'instruments classiques, les guides d'utilisation de 400 pages freinent les plus pressés.

La prise en main de nouvelles technologies nécessite en effet un guide simple et adapté à tout profil de personne. Les guides d'utilisation vintage c'est pour le grand âge, en d'autres termes, il faut concevoir une nouvelle façon d'apprendre à jouer d'un instrument.

Donc pour résumer, il nous faudrait un tutoriel, interactif, qui ne devient pas barbant, qui se suit étapes par étapes, et qui permet de comprendre les bases du traitement de signal jusqu'à la création d'une mélodie.

Nous nous souvenons tous des jeux ludiques de notre enfance, ils marquent encore les esprits. Dans cette même perspective, quoi de mieux que d'apprendre en s'amusant ?

C'est ainsi que nous avons pensé à la création d'un jeu, un jeu qui permettrait de se familiariser avec l'instrument, en découvrant les paramètres au fur et à mesure et non tout en même temps. Apprendre en s'amusant, telle est la devise que l'on veut suivre car c'est une devise qui marche.

Afin de donner l'opportunité à toutes les générations, à tous les profils, à tous les curieux de prendre en main un instrument de musique de pointe, nous avons conceptualisé un jeu d'aventure en réalité virtuelle, axé sur la prise en main d'un synthétiseur, qui se nomme SYNTH'EASY QUEST. Dans un premier temps, afin de garder le côté sensoriel de la prise en main d'un instrument, nous avons choisi de fabriquer un clavier avec quelques notes et boutons qui permettra au joueur de communiquer avec le clavier du synthétiseur virtuel.

Dans un deuxième temps, le but étant de voir les effets de certains paramètres essentiels en traitement de signal (comme la fréquence, l'enveloppe, ou le choix d'oscillateur sur un signal), nous avons voulu mettre en forme le signal en 3D.

Afin de vous présenter plus en détails la démarche dans laquelle s'inscrit notre innovation et notre jeu, ainsi que les travaux menés jusqu'à présent, nous invitons à prendre connaissance de la suite.

## 2 Démarche DESIGN THINKING : Le Vintage c'est pour le grand âge

### 2.1 Brainstorming

A partir du slogan qui nous a été donné, chacun des membres du groupe a écrit sur des post-it les mots auxquels il pensait. Pour nous aider dans notre réflexion, nous nous sommes basés sur les thèmes suivants:

- Les lieux
- Les Hommes
- Les objets
- Les temps
- Les interactions
- Les mouvements
- Social, écologique, légal, économique

Nous avons ensuite mis en commun l'ensemble de nos post it et les avons regroupé par thème:

- Le vintage comme argument marketing
- Grand âge
- Intérêt de la musique
- Nouvel âge, avancées technologiques
- La musique dans la société
- Instruments et écologie

A partir de ces thèmes, nous avons dégagé les 4 paradoxes suivants :

**Comment la production de masse peut être l'opportunité de recycler?** Nous pensons qu'il serait intéressant que notre projet puisse s'inscrire dans une démarche de développement durable.

**Comment intégrer un objet moderne dans une tendance vintage ?** Nous souhaiterions que notre projet fasse appel à des technologies modernes sans pour autant perdre le côté vintage, pur et authentique de la musique.

**Comment la recherche du parfait peut être l'opportunité de faire un son authentique?** L'utilisation de nouvelles technologies supposerait que l'on cherche à créer des sons parfaits, que l'on peut facilement conserver au cours du temps. Cependant nous souhaiterions que l'utilisation de la technologie dans notre projet ne dénature pas les sons produits par les instruments.

**Comment une approche simple et ludique peut être opportunité pour s'approprier une technologie de pointe?** L'une des faiblesses du synthétiseur que nous avons eu l'occasion de tester est son manque de simplicité. Il est extrêmement difficile, voire impossible pour un débutant de le prendre en main (sauf lire les centaines de pages qui composent le manuel d'utilisation). Nous souhaiterions que notre projet puisse aider les utilisateurs à comprendre les différentes fonctionnalités du synthétiseur.



Figure 1: Récapitulatif des paradoxes

## 2.2 Quelles idées sont apparues ?

- JEU ?
- Capteurs de mouvements
- Instruments
- Casque VR
- Quête avec plusieurs niveaux
- En ligne
- Possibilité de faire des quêtes à plusieurs
- Qui, quand, où
- Compétition ou pas - Que nécessite le jeu ? Une performance

## 2.3 Approfondir les paradoxes

- Comment intégrer un objet moderne dans une tendance VINTAGE ?

Objet moderne = qui est récent, qui a été réalisé depuis peu de temps et souvent d'une manière différente de ce qui avait été fait précédemment

Vintage = définition de base : Vin de Porto, récolté les années exceptionnelles, et qu'on laisse vieillir pendant au moins dix ans. Ce que nos parents écoutaient, ancien, vieux, rétro, retour dans le passé, old school

- Comment la recherche du parfait peut être une opportunité pour faire un son authentique ?  
Son parfait = sans bruit, note pure  
Son authentique = son avec bruits parasites

- Comment la production de masse peut être l'opportunité de recycler ?  
Production de masse = instruments stockés, instruments d'occasion, le marché des instruments  
Recycler = Réutiliser, donner une nouvelle vie, revisiter un concept

- Comment une approche simple et ludique peut être l'opportunité pour s'approprier une technologie de pointe ?

Simple = qui ne nécessite pas de bagage musical, de prérequis Ludique = qui donne envie, intuitif, doté d'une logique, avec lequel on se peut se familiariser S'approprier = se familiariser, maîtriser, prendre en main, finalité : devenir expert Technologie de pointe = objet avec des fonctionnalités poussées, qui est difficilement utilisable sans connaissances préalables

### **3 Notre projet**

Essentiellement à partir du dernier paradoxe, nous avons décidé de créer un jeu.

#### **3.1 Principe du jeu**

Pour expliquer le principe de notre jeu, nous avons réalisé une vidéo accessible via ce lien :  
<https://drive.google.com/file/d/1UvNAmiQ2hOeBqITV3AOyOzCVMj76FwOa/view?usp=sharing>

Le synthétiseur est à première vue un instrument qui ressemble à un clavier. Jusque-là tout va bien. Mais si on s'y intéresse de plus près, on y découvre maintes fonctionnalités... Filter, voice, enveloppe, oscillator ?

Clairement, sans aide impossible de s'y retrouver. Concernant le guide d'utilisation...

400 pages ?? C'est trop, et il n'y a même pas de schémas explicatifs ! De quoi abandonner.

Finalement ça nous a tout l'air d'être un outil pour les professionnels, les experts. Seuls des ingénieurs et des professionnels du son pourraient l'utiliser...

**De toute évidence, un problème se pose : finalement, c'est quoi un instrument de musique intelligent ?**

Nous, étudiants de l'ENSEA, qui est une école d'ingénieur spécialisée en électronique avons tenté de répondre à cette question.

**Notre défi ?**

Fournir une opportunité de s'approprier une technologie de pointe de manière ludique et simple, qui rassemble les générations et qui ne nécessite pas de prérequis. Parce que l'accessibilité est notre priorité ;)

Et suite à la phase de réflexion et de brainstorming expliquée précédemment, notre idée est née ! Aujourd'hui on vous propose Synth'Easy ! Synth'easy est LE jeu qui va vous permettre d'apprendre à manier un synthétiseur à votre rythme, de manière drôle, seul ou à plusieurs, pour un maximum d'ambiance !

### 3.2 Implantation du projet dans le marché

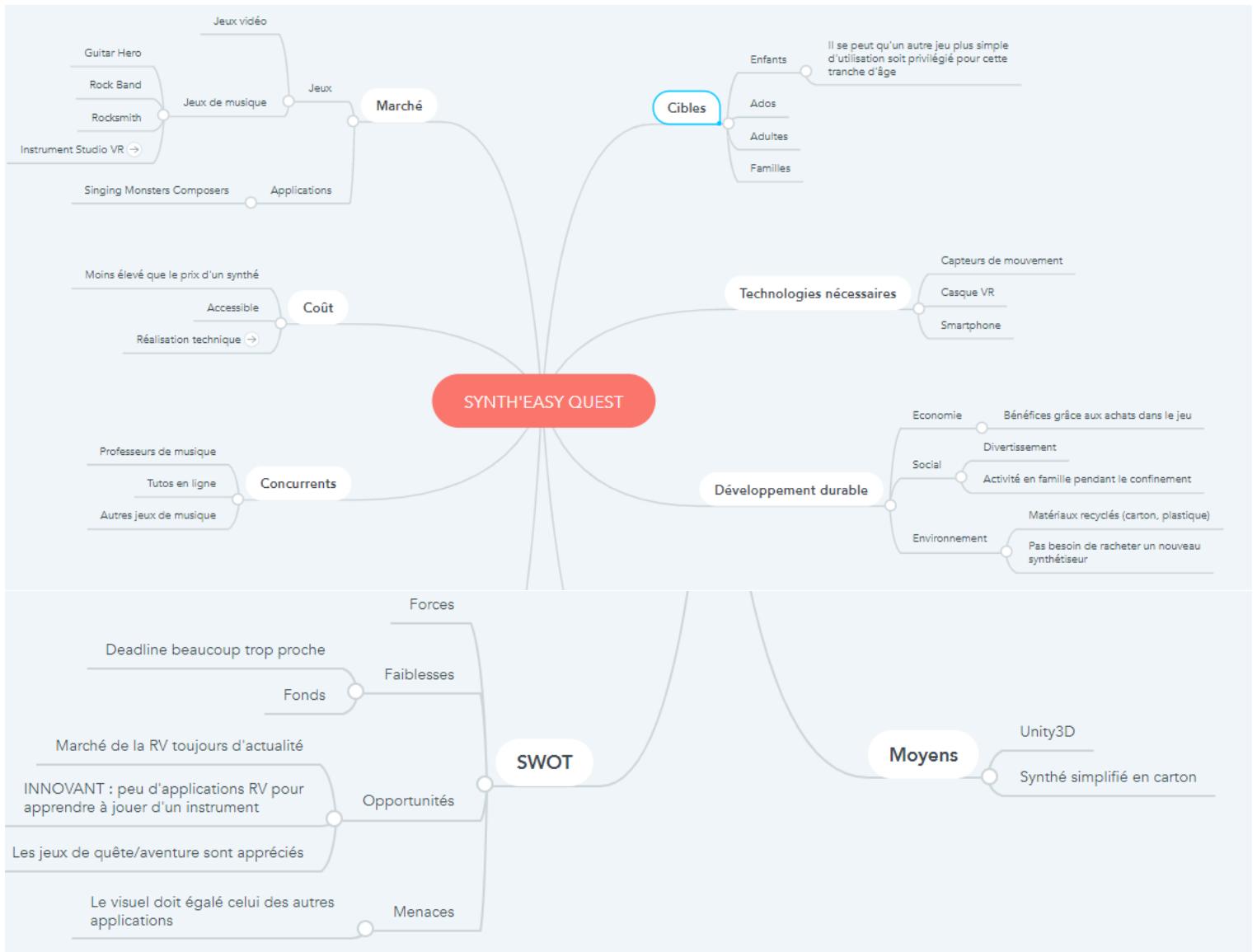


Figure 2: MindMap du marché

### 3.3 Le concept du jeu

Pour nous, le principal problème ayant été la prise en main du synthétiseur, la priorité était de réfléchir à un moyen de faciliter son utilisation.

1er point sur lequel nous sommes tombés d'accord: il faut procéder par étape afin de ne pas noyer l'utilisateur sous une tonne d'informations.

2 ème point: On ne veut pas que la prise en main du synthétiseur soit aussi ennuyante que la lecture d'un mode d'emploi.

Nous avons alors eu l'idée de créer un jeu.  
Ce jeu aurait pour but d'aider l'utilisateur à apprivoiser le synthétiseur. Il s'agirait de compléter des quêtes afin de débloquer et comprendre les différentes fonctionnalités du synthétiseur.

Cependant, nous ne voulons pas que l'apprentissage se fasse avec une simple manette de jeu. Nous avons donc eu l'idée d'intégrer des capteurs de mouvements et peut-être même un casque VR.

Grâce au casque VR, l'utilisateur n'utilisera non plus seulement ses mains et son ouïe, mais il aurait un visuel direct sur le signal créé. De cette manière, il serait plus facile pour lui de comprendre par exemple les notions d'enveloppe et de filtre.

Les capteurs disposés sur les mains de l'utilisateur lui permettraient d'avoir la main sur les différentes propriétés du signal. Par exemple, avec un geste franc de haut en bas, il pourrait diminuer l'amplitude de celui-ci, ou bien modifier l'enveloppe du signal.

L'objectif serait pour l'utilisateur de créer des signaux à l'aide du synthétiseur, qui l'aideront à franchir les obstacles et abattre les ennemis qui se dresseront sur sa route.

En complétant différentes quêtes, l'utilisateur débloque petit à petit de nouvelles fonctionnalités jusqu'à devenir le meilleur des Maestro.

Pour conserver l'esprit convivial de la musique, nous souhaiterions que les utilisateurs puissent réaliser les quêtes à plusieurs, ou même les uns contre les autres.

Nous souhaiterions que le jeu soit disponible en ligne et accessible à des personnes de tout âge.

### 3.3.1 En quoi ce jeu est-il une innovation ?

Voci un schéma qui répond à cette volonté de se démarquer et d'apprter un regard neuf sur l'apprentissage des bases du traitement de signal jusqu'à la prise en main du synthétiseur :

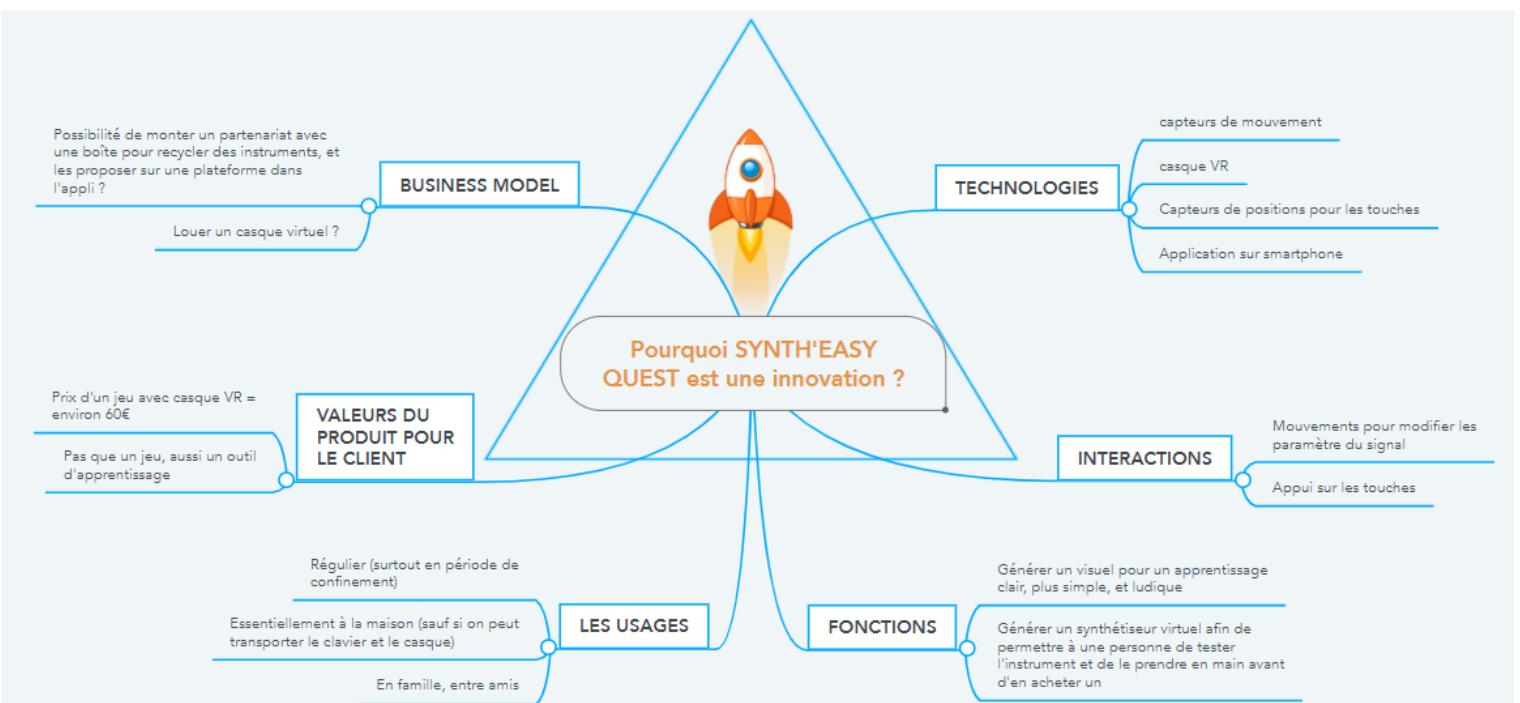


Figure 3: Innovations portées par le jeu

### 3.4 Scénario initial

Pour ce jeu, nous partons sur une application réalité virtuelle disponible sur téléphone. Il faudra ainsi se procurer le casque de réalité virtuelle, l'application accompagnée de son kit "synthé à monter soi-même" qui sera connecté à l'application.

Phase 1 : Montage du synthé en carton (comportant touches de clavier, joystick, boutons)

Phase 2 : Lancer l'application et installer le téléphone dans le casque de réalité virtuelle

Phase 3 : Arrivée dans le jeu

## Scénario global

**But du jeu :** Réanimer la ville en jouant des partitions et des morceaux afin de lui redonner ses couleurs !

Le personnage : Crée ton avatar, à ton image (ou pas !)

C'est le seul rescapé de la ville, il a échappé à cette vague mystérieuse qui a fait sombrer la ville dans une profonde tristesse.

Pourquoi doit-il accomplir cette mission ? Il veut à tout prix sauver sa ville et la délivrer de cette tristesse afin d'y faire régner la joie et le bonheur d'antan.

Il dispose d'un synthétiseur qui lui permettra de jouer des notes et des morceaux afin de délivrer petit à petit un recoin de la ville.

En progressant niveau par niveau, de nouvelles fonctionnalités seront débloquées.

Pour délivrer un recoin, il faudra gagner plusieurs niveaux, et à la fin reproduire un morceau qui fait appel aux compétences développées précédemment.

Afin de rendre accessible le jeu à toutes les générations, différents niveaux et types de musiques seront proposés, ciblés selon l'âge :

### Scénario pour enfants 5-13 ans :

Possibilité d'enlever le casque ?

Pas de fonctionnalités supplémentaires, juste le clavier

Reproduire quelques notes seulement afin de découvrir la musique et développer une oreille musicale. A la fin de la quête, l'enfant serait capable de jouer une mélodie seul.

### Scénario pour adolescents 14-18 ans :

Ce serait de l'initiation au traitement de signal.

Début : Reproduction d'une mélodie

Fin : Découverte du traitement du signal et des fonctionnalités / avancées.

### Scénario pour Adultes 19-60 ans :

Début : découvrir le clavier, reproduire des musiques

Milieu : Rentrer vivement dans l'utilisation du synthétiseur et des fonctionnalités

Fin : Pouvoir reproduire un morceau sans guide/sans aide en utilisant les fonctionnalités à bon escient

### Scénario pour Adultes de + de 60 ans :

Insister sur le visuel

Morceaux moins longs

## 3.5 Scénario développé et plus détaillé (MARIE)

Une des tâches du projet consiste à imaginer le scénario du jeu. Cette section s'attarde à décrire le scénario du jeu.

Le jeu se déroule à notre époque en 2020.

Le personnage principal est un agent secret qui doit aider un professeur de musique à reproduire à l'identique une bande son que celui-ci a perdu.

Avant de reproduire ce son, le personnage doit apprendre à maîtriser quelques fonctions du synthétiseur.

D'autres agents secrets veulent réussir avant lui.

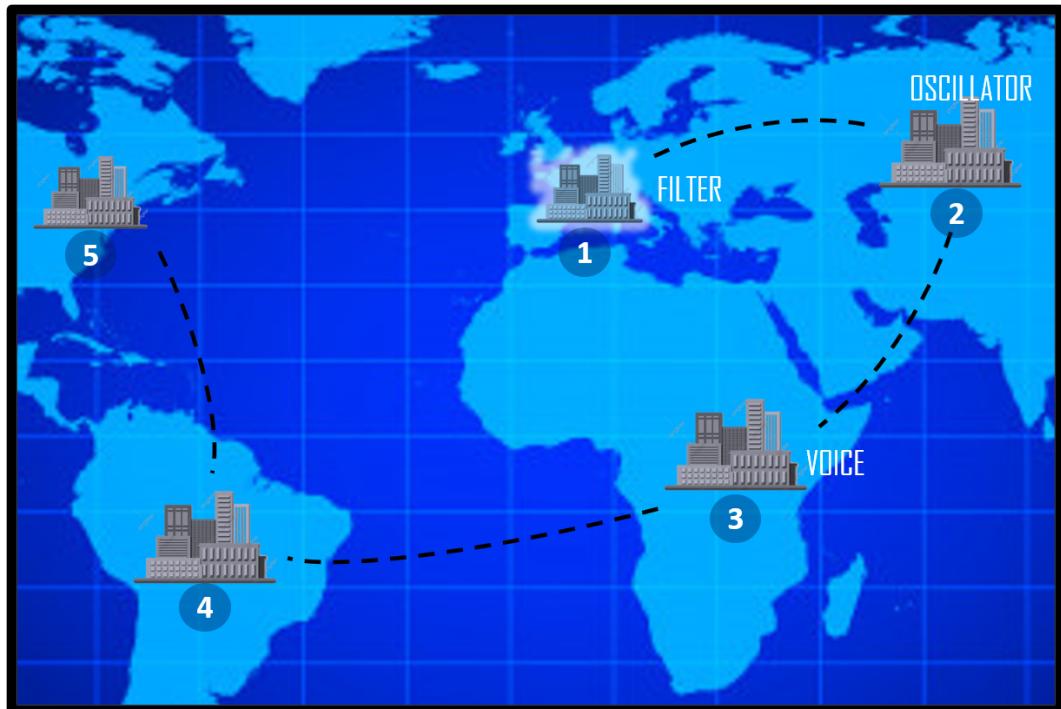


Figure 4: Carte du jeu

Il y a un territoire, composé de plusieurs villes. Dans chaque ville, le personnage peut apprendre une fonctionnalité du synthétiseur : FILTER, OSCILLATOR, VOICE

L'apprentissage de ces fonctionnalités se fait sous formes de mini missions . A chaque ville correspond une fonctionnalité.

A chaque fois que toutes les mini missions d'une même ville sont réussies, une autre ville est débloquée et le personnage apprend alors une nouvelle fonctionnalité dans la nouvelle ville débloquée.

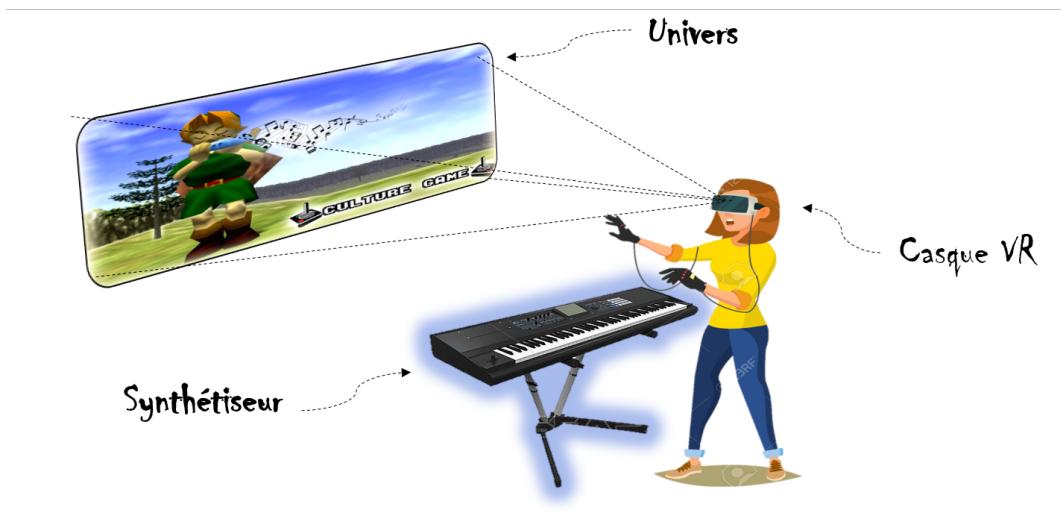


Figure 5: Le synth'easy quest

L'utilisateur dispose d'un synthétiseur en carton. Un clavier et des capteurs de position du doigt.

Le synthétiseur dans le virtuel doit correspondre au synthétiseur que l'utilisateur dispose en vrai. (Clavier, bouton on/off , modules Oscillateur, Filtre, Enveloppe,)

### Scénario détaillé

On allume le jeu : Affichage d'une Carte en 2d : La carte représente un continent. Il y a plusieurs noms de villes. Les villes non visitées sont grises. Quand elles sont visitées, elles sont en couleurs. Au début du jeu, seule 1 ville est débloquée.

Sélection d'une ville. (Sans personnage, juste une sélection). A chaque ville, un panneau s'affiche avec les différentes missions à réaliser.



Figure 6: Aperçu de la ville

1ère ville : Oscillateur. On a alors un personnage en 3d vu d'une caméra externe dans une ville en 3d. (ville que l'on peut prendre sur free3d ou alors création de bâtiments utiles : laboratoire, compétition, magasin) Sur chaque bâtiment, il y a des noms. Il y a un bâtiment où il y a écrit laboratoire . Le personnage peut tourner la tête Le personnage s'avance vers l'entrée du bâtiment. En cliquant sur la porte il rentre à l'intérieur. (Ouverture d'une porte)

Le personnage se retrouve dans le laboratoire : Devant lui, il y a un synthétiseur. Il y a également un homme immobile sur le côté. C'est le professeur qui accompagne l'utilisateur tout au long du jeu. Quand le personnage rentre dans la pièce, automatiquement, le professeur engage le dialogue et invite le personnage à prendre en main le synthétiseur.

Le personnage s'avance (et s'assoit) vers le synthétiseur. On voit juste ses mains au-dessus du clavier. Le professeur guide l'utilisateur :

- Soit par la voix
- Soit avec affichage des consignes

Il y a un bouton "on". La première consigne du professeur est d'allumer le synthétiseur.



Figure 7: Aperçu de l'intérieur du laboratoire

Click sur le bouton "on"  
Allumage du voyant "on"



Figure 8: Visualisation de l'onde 3D

Fonction oscilloscope. L'utilisateur clique sur le bouton oscilloscope. -Affichage d'une onde en 3d devant le synthétiseur en grand. -Ou affichage en 2D sur un oscilloscope à côté. Affichage d'un menu virtuel avec certaines fonctions présentes dans le module oscilloscope. Il y a toujours les consignes du professeur. Les formes d'ondes. Quand l'utilisateur sélectionne une forme d'onde, (sinusoïdale, triangulaire, carrée, en dent de scie) on a l'affichage de la nouvelle forme d'onde avec sortie du son correspondant. Le choix de la fréquence : L'utilisateur peut sélectionner une fréquence grâce au capteur de position du doigt. L'onde bouge en temps réel et sortie du son correspondant. Le professeur demande à l'utilisateur de jouer un son se rapprochant de celui de la flûte à une certaine note.

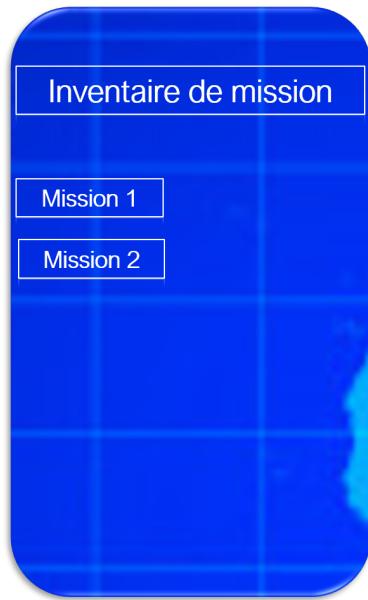


Figure 9: Affichage des missions

Une fois la mission réussie le professeur félicite l'utilisateur. L'utilisateur a un inventaire de mission auquel il a accès tout le temps. A chaque fois qu'une mission est réalisée la mission est barrée. Les autres boutons du synthétiseur sont bloqués.

L'utilisateur doit réaliser une mission : Par exemple: aller chercher un bout de partition avant qu'un ennemi le fasse avant lui dans la ville pour débloquer un bouton. Il dispose d'une carte et peut trouver une trottinette dans la rue pour aller plus vite que ses ennemis Quand toutes les missions de la ville sont réalisées, on revient sur l'interface d'entrée avec la carte en 2D. Une autre ville est dégrisée.

## 4 Développement de l'environnement de jeu sur Unity3D (ISMAIL)

### 4.1 Introduction

### 4.2 Caméra

La caméra sert à capturer et à afficher le monde au joueur, on peut avoir plusieurs caméras, cependant on a utilisé qu'une seule caméra, attachée à la tête du personnage et qui suit ses mouvements en permanence.

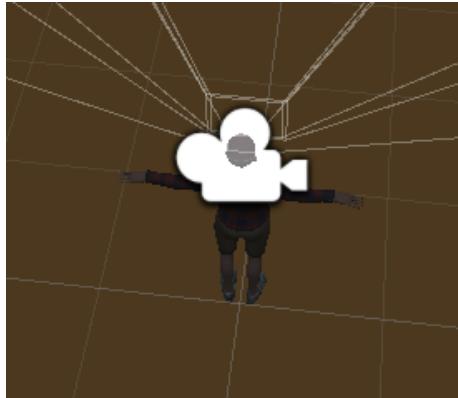


Figure 10: caméra

### 4.3 Personnages

Notre jeu contient 3 personnages, le joueur principal, le professeur ainsi que son assistant. Les trois personnages ont été créés sur la plateforme Mixamo, les deux personnages non joueurs sont fixes à l'intérieur du laboratoire et sont animées.



Figure 11: Le personnage principal.

### 4.4 Décor

L'entraînement du joueur se fera au laboratoire, c'est pour cela qu'on a ajouté et optimisé plusieurs détails à l'intérieur de notre laboratoire : des points de lumières, des shaders ainsi que des textures.



Figure 12: Les deux personnages.

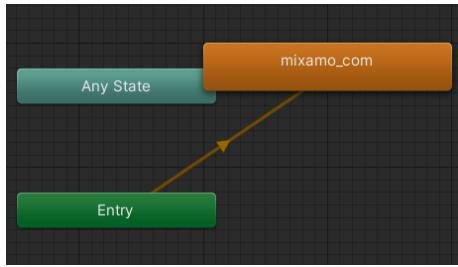


Figure 13: graphe d'états de l'animation, Ces deux personnages sont animées par un graphe d'état sans sortie.

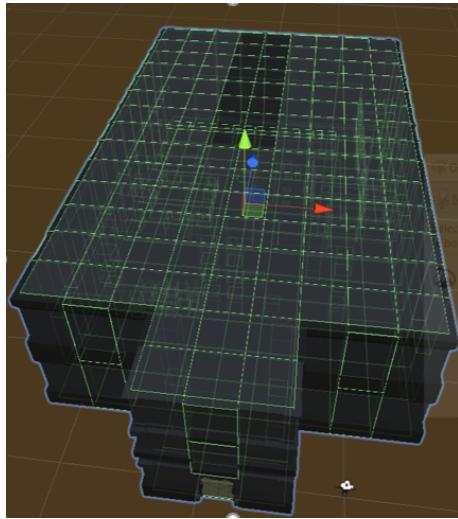


Figure 14: Le laboratoire vu de l'extérieur.

On a également ajouté un menu qui contient trois bouttons : Play, Quit et Options  
Pour créer un menu dynamique, nous avons commencé par créer une nouvelle scène. Nous y avons ajouté des "gameObjects\*" de type GUIText

#### 4.5 Mise en place de la Réalité Virtuelle

La réalité virtuelle est un procédé visant à immerger une personne dans un univers virtuel. Pour ce faire nous avons utiliser le casque Oculus rift S : L'Oculus Rift est un masque de réalité virtuelle. Il fonctionne grâce un détecteur de mouvement intégré au casque et connecté à l'ordinateur via



Figure 15: Le laboratoire vu de l'intérieur.

ècmècm

Figure 16: Menu du jeu.

USB. Ce capteur permet d'adapter la camera d'une application 3D en fonction de l'orientation de la tête du joueur. L'Oculus possède également un écran plat sur lequel est affichée une image stéréoscopique récupérée par deux caméras présentes dans l'environnement.

L'implémentation de l'Oculus est principalement réalisée à l'aide des plug-ins inclus par le constructeur : une caméra OVR qui contient des scripts qui adaptent les fonctionnalités programmés en Réalité virtuelle, l'Asset Oculus et le plugin Oculus XR

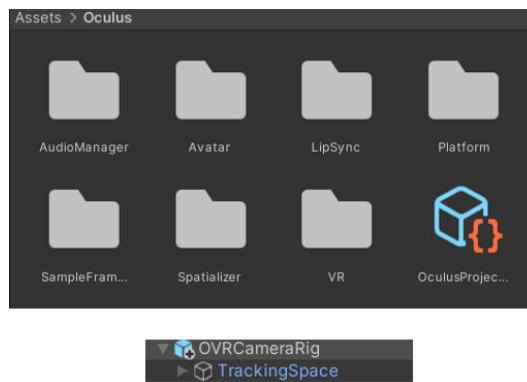


Figure 17: L'asset Oculus ainsi que la Caméra OVR.

## 4.6 Difficultés rencontrées

Lors du développement de l'environnement on a rencontré plusieurs difficultés, tout d'abord on a du s'adapter au langage C sharp qu'on avait pas utilisé précédemment et qui est le langage des scripts utilisés dans Unity.

On a également eu des problèmes de gravité, notre personnage ne détecte pas son contact avec le sol donc du à la gravité s'enfonçait au sol, c'est pour cela qu'on a ajouté le "box collider" qui définit la forme d'un objet pour les collisions physiques avec le sol ainsi que les autres objets du jeu.



Figure 18: Oculus Rift S.

#### 4.7 Améliorations et prochaines démarches

Nous envisageons de développer le jeu en ajoutant un choix de plusieurs villes lors du lancement du jeu, ainsi l'utilisateur pourra changer de scène.

Nous voulions également créer une conversation entre le personnage principal et les personnages du laboratoire afin de débuter l'entraînement.

## 5 Création du clavier réel (LAURA)

### 5.1 Introduction

Comme évoqué précédemment, nous tenions à garder le côté sensoriel intervenant lors de la pratique d'un instrument.

Ainsi, nous avons décidé de fabriquer un clavier, assez basique, constitué des touches du clavier et de boutons auxquels seraient associés les différents paramètres utiles au traitement de signal.

Ce clavier serait relié au jeu via Unity3D. Comment faire cela ? Et bien le cheminement est le suivant :

- 1) L'utilisateur touche le clavier. La donnée est captée par un capteur.
- 2) Le capteur est connecté à une carte Raspberry qui va récolter les données sur un serveur.
- 3) Les données récoltées vont être envoyées à la suite à un serveur client.
- 4) Ce serveur client va écrire les données dans un répertoire afin qu'elles soient lisibles par le jeu.
- 5) Enfin, un script C# dans Unity va lire le répertoire et accéder aux données !

Cette partie et la suivante vont donc décrire les détails de la mise en place de ces étapes.

### 5.2 Assemblage hardware

Le capteur MPR121 L'idée première de notre jeu étant d'aider le joueur à prendre un synthétiseur, nous ne voulions pas que le gameplay se limite à l'appui sur des boutons. Nous avons donc décidé d'utiliser un capteur sensitive touch .

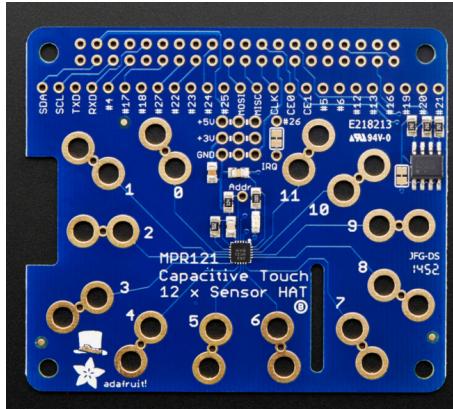


Figure 19: capteur MPR321

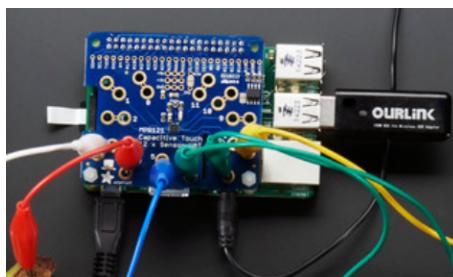


Figure 20: capteur MPR321 avec fils crocodiles

Nous avons commencé par utiliser des pinces crocodiles pour utiliser ce capteur. Par la suite, nous avons fabriquer un clavier avec des touches en aluminium pour représenter le synthétiseur.



Figure 21: clavier avec les touches en aluminium

### 5.3 Fonctionnement du capteur

Le capteur est capable de détecter le Touch et le Release de chacun des canaux. Prenons par exemple le canal 0. A intervalle de temps choisi par le programmeur, le capteur mesure la valeur de la capacité sur ce canal. Il la compare à une ligne de base. Si cette valeur dépasse un certain seuil, qui a lui aussi été fixé par le programmeur, le statut touch est reporté dans le registre correspondant. Le fonctionnement est similaire pour le statut release . Les valeurs de seuils sont à régler pour chacun des canaux.

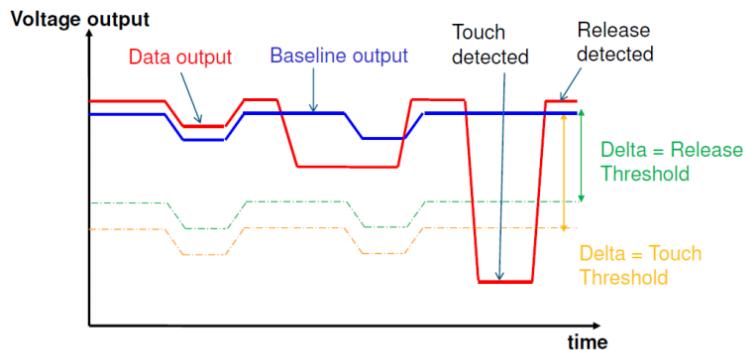


Figure 5. MPR121 Touch and Release Detection

Figure 22: Valeurs seuils du capteur MPR121

Le capteur permet également au programmeur de jouer sur la distance à laquelle les mains de l'utilisateur son détectés. Nous pourrions donc permettre au joueur de jouer sur les différentes fonctionnalités du jeu sans même avoir à toucher les capteurs. Nous avons finalement décidé de ne pas jouer sur ce paramètre parce que des bugs se présentaient déjà sans que nous l'utilisions.

### 5.4 Activation du capteur

Par défaut, le capteur est en mode sommeil . Pour activer l'un des canaux, il suffit, après reset, de mettre à 1 le bon bit dans le registre ECR dont l'adresse est 0x5E.

En écrivant 0x04 dans le registre 0x5E, nous autorisons les mesures sur les canaux 0 à 3.

```
//Préparation des capteurs
int fd= wiringPiI2CSetup(0x5A);
if(fd== -1) printf("erreur fd\n");
//soft reset :
wiringPiI2CWriteReg8(fd,0x80,0x63);
delay(1);
```

Figure 23: Préparation et reset des capteurs

```
//ECR: electrode config reg
wiringPiI2CWriteReg8(fd,0x5E,0x04);
```

Figure 24: Activation des canaux 0 et3

#### 11. Electrode Configuration Register (ECR, 0x5E)

Bit	D7	D6	D5	D4	D3	D2	D1	D0
Read			CL[1:0]		ELEPROX[1:0]			ELE[3:0]
Write								

Figure 25: Extrait de la datasheet du MPR121

0100	Run Mode with ELE0~3 measurement enabled.
------	---

Figure 26: Extrait de la datasheet du MPR121

## 5.5 Réglage des valeurs seuils pour deux canaux

Pour programmer le capteur la bibliothèque WiringPi et wiringPiI2C a été utilisée. Pour les premiers tests, nous avons décidé d'utiliser deux canaux : ELE0 et ELE2. Nous avons donc réglé les valeurs de seuils de détection pour ces deux canaux. Après plusieurs tests, nous avons choisi les valeurs de seuils suivantes :

```
#define TOUCH 0xCE
#define RELEASE 0xCA
```

Figure 27: Valeurs seuils du capteur MPR121

avec 0x5E= 206 et 0xCA= 202. La valeur maximale pour les deux paramètres est 255.

Il suffit ensuite d'écrire ces valeurs TOUCH et RELEASE , aux bonnes adresses pour les canaux que l'on souhaite utiliser et 0 pour les autres.

ELE0 Touch Threshold	E0TTH	0x41	0x00
ELE0 Release Threshold	E0RTH	0x42	0x00
ELE1 Touch Threshold	E1TTH	0x43	0x00
ELE1 Release Threshold	E1RTH	0x44	0x00
ELE2 Touch Threshold	E2TTH	0x45	0x00
ELE2 Release Threshold	E2RTH	0x46	0x00
ELE3 Touch Threshold	E3TTH	0x47	0x00
ELE3 Release Threshold	E3RTH	0x48	0x00
ELE4 Touch Threshold	E4TTH	0x49	0x00
ELE4 Release Threshold	E4RTH	0x4A	0x00
ELE5 Touch Threshold	E5TTH	0x4B	0x00
ELE5 Release Threshold	E5RTH	0x4C	0x00
ELE6 Touch Threshold	E6TTH	0x4D	0x00
ELE6 Release Threshold	E6RTH	0x4E	0x00
ELE7 Touch Threshold	E7TTH	0x4F	0x00
ELE7 Release Threshold	E7RTH	0x50	0x00
ELE8 Touch Threshold	E8TTH	0x51	0x00
ELE8 Release Threshold	E8RTH	0x52	0x00
ELE9 Touch Threshold	E9TTH	0x53	0x00
ELE9 Release Threshold	E9RTH	0x54	0x00
ELE10 Touch Threshold	E10TTH	0x55	0x00
ELE10 Release Threshold	E10RTH	0x56	0x00
ELE11 Touch Threshold	E11TTH	0x57	0x00
ELE11 Release Threshold	E11RTH	0x58	0x00

Figure 28: Adresses des électrodes

```
wiringPiI2CWriteReg8(fd,0x41,TOUCH);
wiringPiI2CWriteReg8(fd,0x42,RELEASE);
wiringPiI2CWriteReg8(fd,0x47,TOUCH);
wiringPiI2CWriteReg8(fd,0x48,RELEASE);

wiringPiI2CWriteReg8(fd,0x43,0);
wiringPiI2CWriteReg8(fd,0x44,0);
wiringPiI2CWriteReg8(fd,0x45,0);
wiringPiI2CWriteReg8(fd,0x46,0);
wiringPiI2CWriteReg8(fd,0x49,0);
wiringPiI2CWriteReg8(fd,0x50,0);
wiringPiI2CWriteReg8(fd,0x50,0);
wiringPiI2CWriteReg8(fd,0x51,0);
wiringPiI2CWriteReg8(fd,0x52,0);
wiringPiI2CWriteReg8(fd,0x53,0);
wiringPiI2CWriteReg8(fd,0x54,0);
wiringPiI2CWriteReg8(fd,0x55,0);
wiringPiI2CWriteReg8(fd,0x56,0);
wiringPiI2CWriteReg8(fd,0x57,0);
wiringPiI2CWriteReg8(fd,0x58,0);
```

Figure 29: Code: écriture des valeurs touch et release

## 5.6 Utilisation des valeurs recueillies par les capteurs

Les valeurs recueillies par le capteur vont être utilisées dans le jeu. Le serveur va devoir envoyer ces valeurs. Pour être sûr que les données envoyées sont les bonnes, nous les affichons dans la console 10 fois par seconde :

```
while(true){
    printf("Reg 0x00: %d\n", wiringPiI2CReadReg8(fd,0x00));
    message_test[0]=wiringPiI2CReadReg8(fd,0x00);
    delay(1000);
```

Figure 31: Lecture de la valeur détectée par le capteur

## 5.7 Le serveur UDP

Afin de récupérer les informations transmises par le capteur, nous avons utilisé un Raspberry Pi sur lequel on implémente un serveur.

Avant la programmation du serveur, la première étape a été la prise en main de l'environnement Raspberry (commandes de base) et l'installation de toutes les bibliothèques nécessaires. (voir guide d'installation) Pour le protocole de transport des informations nous avons choisi UDP, à cause de sa simplicité et de sa rapidité. Son utilisation implique que nous ne recevrons jamais d'acquittement pour les paquets envoyés, ce que ne devrait pas entraver le fonctionnement de notre jeu. La version actuelle du serveur ne permet qu'à un seul utilisateur de se connecter au serveur. Il suffit à ce joueur d'envoyer un message quelconque au serveur. Le serveur conserve l'adresse de l'expéditeur et lui renvoie les données récoltées par le capteur. Ces données sont ensuite traitées pour activer l'une des fonctionnalités du jeu. Cette version du jeu a été codée en C et ne nécessitait que la fonction main pour fonctionner.

## 5.8 Difficultés rencontrées

Ils arrivent que les capteurs dysfonctionnent : par exemple le capteur numéro 0 est supposé envoyer un 0 quand il n'est pas activé et un 1 quand il l'est. Il arrive que le capteur s'active seul, ce qui pourrait dégrader l'expérience utilisateur. Pour régler ce problème, il est nécessaire de relancer le serveur. Nous n'avons pas réussi à trouver la source du problème malgré nos divers essais (changement des pinces crocodiles utilisées, réglage des valeurs seuils...)

## 5.9 Améliorations et prochaines démarches

Une amélioration possible du serveur serait d'adapter le code en C++, afin de permettre à plusieurs joueurs de se connecter au serveur.

## 5.10 Instructions pour la mise en route du serveur

- 1) Télécharger les bibliothèques nécessaires: wiringPi, wiringPiI2C
  - 2) Télécharger un compilateur g++
  - 3) Activer I2C sur la raspberry: <https://projetsdiy.fr/activer-bus-i2c-raspberry-pi-3-pi-zero-w/>: :text=Activer

- 4) Exécuter le programme après compilation:  
-pour compiler: g++ main.cpp -lwiringPi -o prog  
- ./prog

## 6 Établissement de la connexion entre le clavier réel et le clavier virtuel (ABDERRAHIM)

### 6.1 Introduction

L'objectif de cette partie est de récupérer les données envoyées par le capteur via la Raspberry, les préparer pour qu'elles soient utilisables sur Unity, afin de commander un clavier virtuel sur Unity à l'aide du capteur. On procède donc en plusieurs étapes. Avant tout il va falloir créer une petite application qui va permettre de recevoir les données du capteur, ensuite il faudra convertir ces données en entiers, afin qu'elles soient interprétées par un code C# sur Unity sans problème.

### 6.2 Création du serveur client

La classe UdpClient communique avec les services réseaux à l'aide d'UDP. Les méthodes et propriétés de la classe UdpClient assurent l'abstraction des informations nécessaires pour créer un Socket permettant l'envoi et la réception de données avec le protocole UDP.

UDP (User Datagram Protocol) est un protocole simple qui essaie de fournir les données à un hôte distant le plus efficacement possible. Toutefois, comme le protocole UDP est un protocole non connecté, les datagrammes UDP envoyés au point de terminaison distant ne sont pas assurés d'arriver, ou d'arriver dans la même séquence que celle dans laquelle ils ont été envoyés. Les applications qui utilisent le protocole UDP doivent être en mesure de gérer les datagrammes manquants, en double et hors séquence.

Pour envoyer un datagramme avec une connexion UDP, vous devez connaître l'adresse de l'appareil réseau qui héberge le service dont vous avez besoin, ainsi que le numéro de port UDP utilisé par ce service pour communiquer. Dans le cadre de notre projet, nous avons créé une application qui va permettre de se connecter à la Raspberry et recevoir les données du capteur, cette application codée en C# permet de se connecter à la Raspberry, recevoir des données avant de les convertir au bon format pour qu'elles soient traitable facilement à l'aide d'un code script C# sur Unity.

### 6.3 Récupération de variable sur Unity3D

Pour récupérer les variables reçues sur Unity, on utilise un script C# qui va accéder au fichier où les variables sont stockées, et activer des boutons d'un clavier virtuel sur Unity selon la valeur de la variable.

### 6.4 Difficultés rencontrées

Quand nous avons fusionné les parties, nous avons remarqué un petit retard au niveau du traitement de la donnée. En effet les données reçues par l'application sont stockées dans un fichier texte après être converties au bon format. Or ce fichier texte est créé puis supprimé autant de fois qu'une nouvelle donnée est reçue, soit 10 fois par seconde dans notre cas. Le script Unity n'a donc pas le temps de lire la variable. Ainsi il faudrait réduire le temps d'envoi de données à ne fois par seconde par exemple, ce qui diminue la cadence d'envoi de données et donc diminue l'efficacité de la réception de donnée à partir du clavier réel.

Nous avons toutefois essayé d'avoir l'application dans le même répertoire Unity où se trouve le script C# qui permet de traiter les données du capteur, mais l'application n'était pas compatible avec Unity3D. Ainsi, il était impossible de passer en mode exécution sur Unity.

Finalement nous avons réduit le temps de réception de donnée à 1 fois par seconde.

### 6.5 Améliorations et prochaines démarches

Une des améliorations possibles est de pouvoir lire la donnée du capteur envoyée par la Raspberry directement sur Unity sans passer par un fichier texte, mais en implémentant directement le serveur client sur Unity.

## 7 Création du synthétiseur virtuel sur Unity3D

### 7.1 Introduction

La synthèse sonore est la production électronique de sons à partir de propriétés de base comme les sons sinusoïdaux et autres ondes simples. Les synthétiseurs tirent leur nom du fait qu'ils imitent, ou synthétisent, une large gamme de sons comme le son d'un autre instrument, la voix, un hélicoptère, une voiture ou encore un chien. Les synthétiseurs peuvent également produire des sons qui n'existent pas à l'état naturel. Cette capacité à générer des sons qui ne peuvent pas être créés d'une autre façon fait du synthétiseur un outil musical unique.

Dans un synthétiseur, le composant chargé de générer le son est appelé oscillateur. La plupart des oscillateurs de synthétiseur génèrent, outre des ondes sinusoïdales, des formes d'onde très riches d'un point de vue harmonique telles que les ondes en dents de scie, triangulaires, carrées ou pulsées. Ces différentes formes d'onde ont été baptisées en référence aux objets auxquels elles ressemblent (dents de scie, triangle, carré, etc.).

Pour transformer le son fondamental et ses harmoniques en un son différent, conforme à nos souhaits, il faut acheminer le signal provenant de l'oscillateur à des modules de traitement de signal qui vont permettre de modeler le son tels que le filtre, l'amplificateur, ou l'enveloppe.

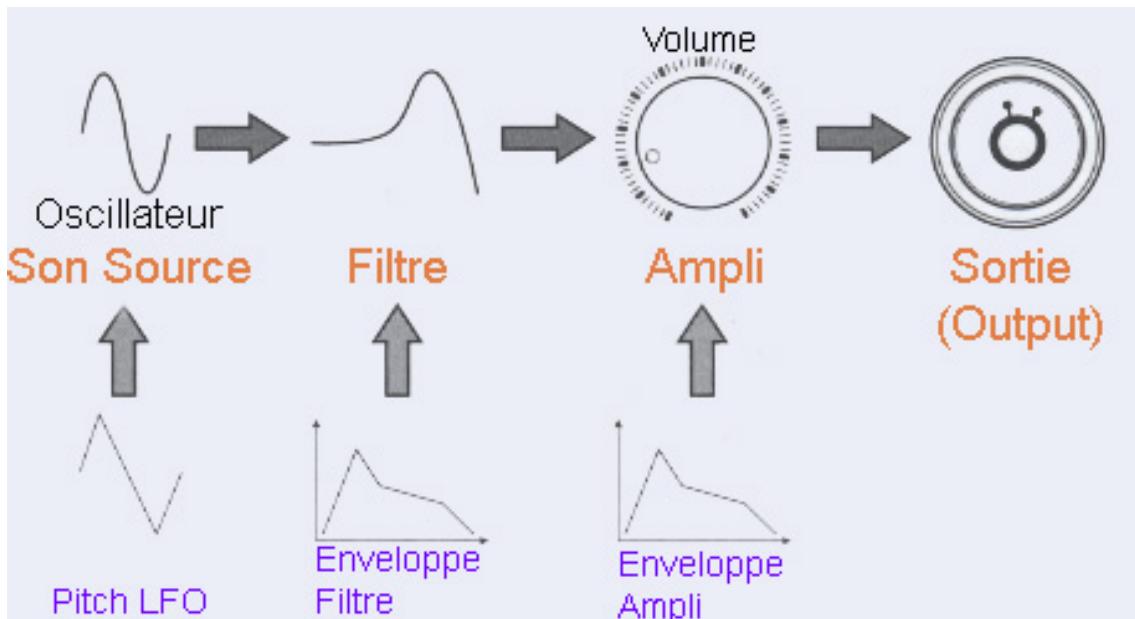


Figure 32: Principe de fonctionnement d'un synthétiseur

Ici nous allons développer 3 modules de traitement de signal :

- 1) L'oscillateur
- 2) L'enveloppe
- 3) Le filtre

#### 7.1.1 L'oscillateur

Le signal audio d'un synthétiseur est généré par l'oscillateur. Il existe différentes formes d'onde contenant divers types et quantités d'harmoniques. Ainsi, la relation de niveau entre le son fondamental et les harmoniques, pour une forme d'onde donnée, est responsable de la couleur ou timbre de base du son.

Voici les types de forme d'onde :

- Onde sinusoïdale : nette et rendant un son clair, cette onde contient uniquement le premier harmonique, c'est-à-dire le son fondamental. L'onde sinusoïdale utilisée seule permet de créer des sons purs comme un sifflement, un diapason, etc.

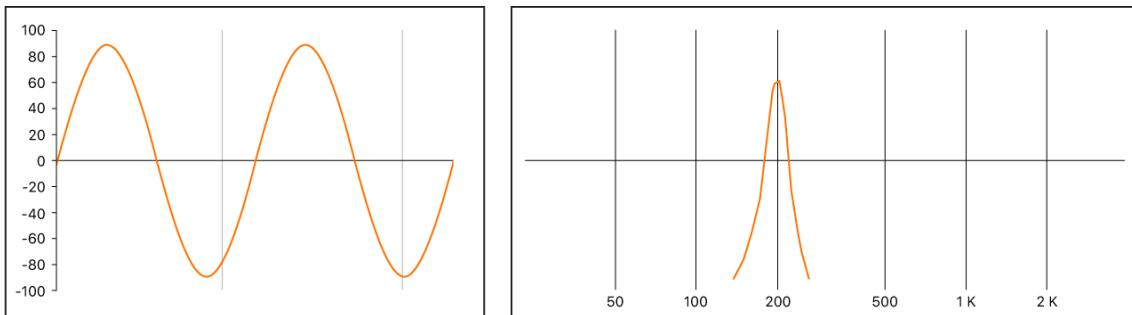


Figure 33: Onde sinusoïdale

- Onde en dents de scie : cette onde qui correspond à un son clair et éclatant contient des harmoniques pairs et impairs, ainsi que le son fondamental. Elle est idéale pour créer des sons d'instruments à cordes, de cuivres, de pad et de basse.

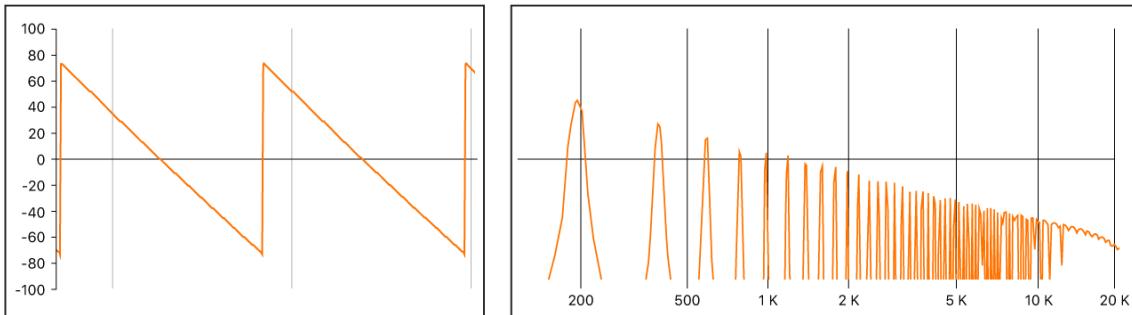


Figure 34: Onde en dent de scie

- Onde carrée : caractérisée par un son creux et boisé, l'onde carrée contient une large gamme d'harmoniques impairs, ainsi que le son fondamental. Elle est utile pour créer des instruments à anche, des pads et des basses. Elle peut également être utilisée pour émuler les grosses caisses, les congas, les toms et d'autres types de percussions. Elle est souvent associée à une autre forme d'onde d'oscillateur telle qu'un bruit.

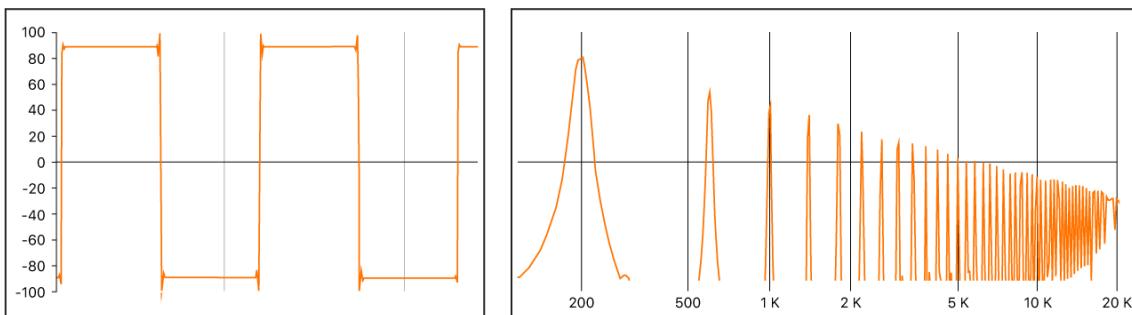


Figure 35: Onde carrée

- Onde triangulaire : les ondes triangulaires ne contiennent que des harmoniques impairs et le son fondamental. Les harmoniques les plus hauts d'une onde triangulaire sont plus rapides que ceux d'une onde carrée et rendent le son plus doux. Elle est idéale pour créer des sons de flûte, de pad et de chœurs vocaux.

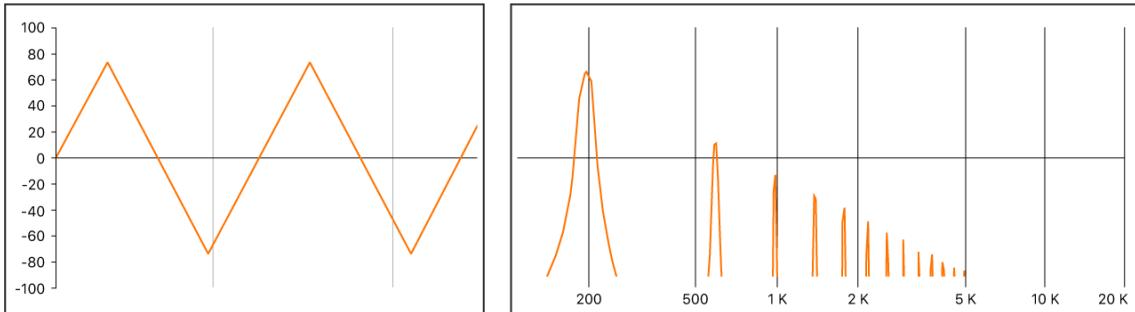


Figure 36: Onde triangulaire

Pour notre projet, nous avons sélectionné 3 oscillateurs particuliers : un oscillateur sinusoïdal, un oscillateur carré, et un oscillateur triangulaire.

**Une étape consiste donc à coder ces oscillateurs afin qu'ils génèrent le signal audio du synthétiseur.**

### 7.1.2 L'enveloppe

Un des paramètres de traitement du signal audio est de contrôler le signal au fil du temps. Prenons comme exemple le son d'un violon. Le son monte graduellement jusqu'à atteindre un niveau de crête (un maximum), à mesure que l'archet frotte contre la corde, il se stabilise un temps jusqu'à ce que l'archet décolle de la corde et chute de manière abrupte à ce moment.

A contrario, le fait de frapper sur une caisse claire à l'aide de baguettes provoque un niveau de crête très rapide sans stabilisation avant que le son ne s'arrête immédiatement — bien qu'il y ait tout de même un certain degré de chute (decay), à savoir le temps mis pour redescendre depuis le niveau de crête.

Ces deux sons présentent clairement des caractéristiques très différentes au fil du temps.

Les synthétiseurs sont capables d'imiter ces caractéristiques sonores en permettant de contrôler les différentes portions du niveau sonore, sur la durée (début, milieu et fin). Ce contrôle est obtenu à l'aide d'un composant appelé générateur d'enveloppe.

Dans l'illustration ci-dessous est représenté l'oscillogramme de son de percussion ; le niveau atteint immédiatement le haut de sa plage avant de chuter. On dessine une ligne autour de la moitié supérieure de cet oscillogramme : cette ligne constitue l'enveloppe du son : une représentation du niveau en fonction du temps. Le rôle du générateur d'enveloppe est de définir la forme de cette enveloppe.

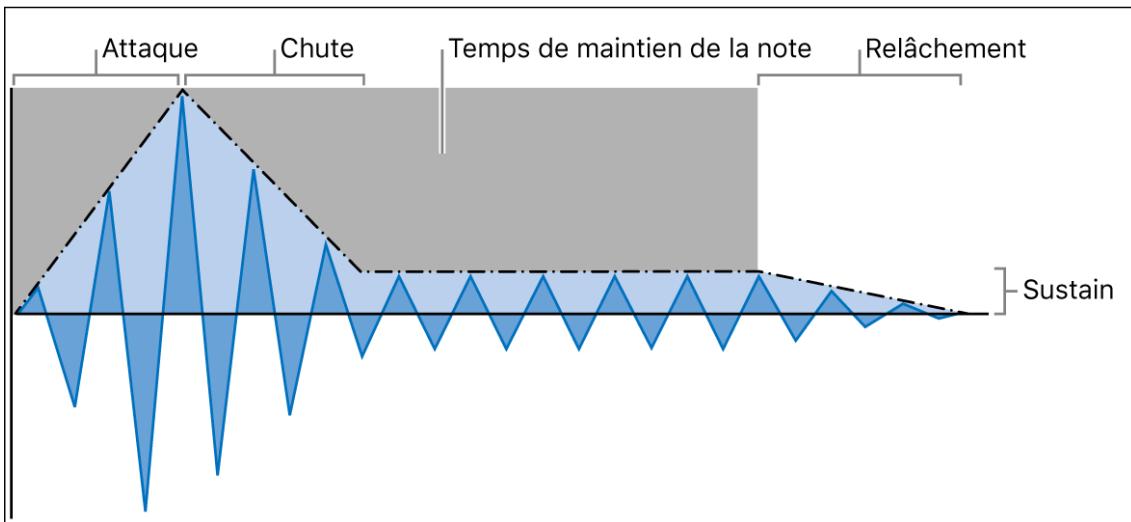


Figure 37: Enveloppe d'un son

Le générateur d'enveloppe possède généralement quatre commandes souvent appelées ADSR, à savoir l'attaque (Attack), la chute (Decay), la tenue (Sustain) et le relâchement (Release).

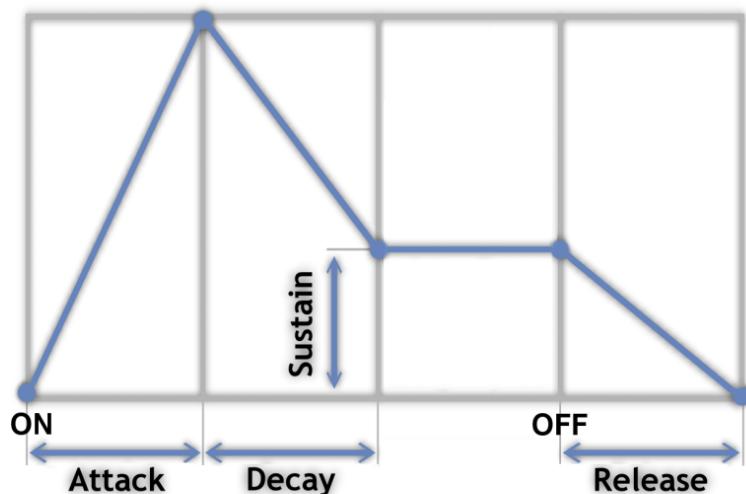


Figure 38: Les 4 Commandes de l'Enveloppe

Les commandes :

**Attack** : détermine le temps nécessaire pour que le signal passe d'une amplitude de 0 à 100

**Chute** : détermine le temps qu'il faut au signal pour passer de 100

**Tenue** : détermine le niveau d'amplitude maintenu lorsque la touche est tenue enfoncée.

**Relâchement** : détermine le temps nécessaire pour que le son chute du niveau de tenue défini à une amplitude de 0 lorsque la touche est relâchée.

Une étape consiste donc à coder ces fonctionnalités afin de jouer sur l'enveloppe du signal de telle sorte à pouvoir obtenir des caractéristiques de sons différentes.

### 7.1.3 Le filtre

L'objectif du filtre, est de supprimer certaines parties du signal (spectre de fréquences) envoyées par les oscillateurs.

Il existe plusieurs types de filtre de base. Chacun d'eux présente un effet différent sur les diverses portions du spectre de fréquences.

**Filtre passe-haut :** Les hautes fréquences sont transmises ; les basses fréquences sont atténuées.

**Filtre passe-bas :** Les basses fréquences sont transmises ; les hautes fréquences sont atténuées.

**Filtre passe-bande :** Seules les fréquences d'une bande donnée sont transmises.

**Filtre coupe-bande :** Seules les fréquences d'une bande donnée sont atténuées.

**Filtre passe-tout :** Toutes les fréquences du spectre sont transmises, mais la phase de sortie est modifiée.

Afin de choisir la plage de fréquences sur laquelle le filtre va agir, il faut choisir une fréquence de coupure. La fréquence de coupure détermine l'endroit où le signal est coupé. Par exemple, avec un filtre passe-bas, si un signal contient des fréquences allant de 20 à 4000 Hz et que la fréquence de coupure est réglée sur 2500 Hz, les fréquences supérieures à 2500 Hz seront filtrées. Le filtre passe-bas laisse passer sans altérer les fréquences inférieures au point de coupure (2500 Hz).

L'illustration ci-dessous présente une onde en dents de scie. Le filtre est ouvert, avec une valeur de coupure maximale. En d'autres termes, cette forme d'onde n'est pas filtrée.

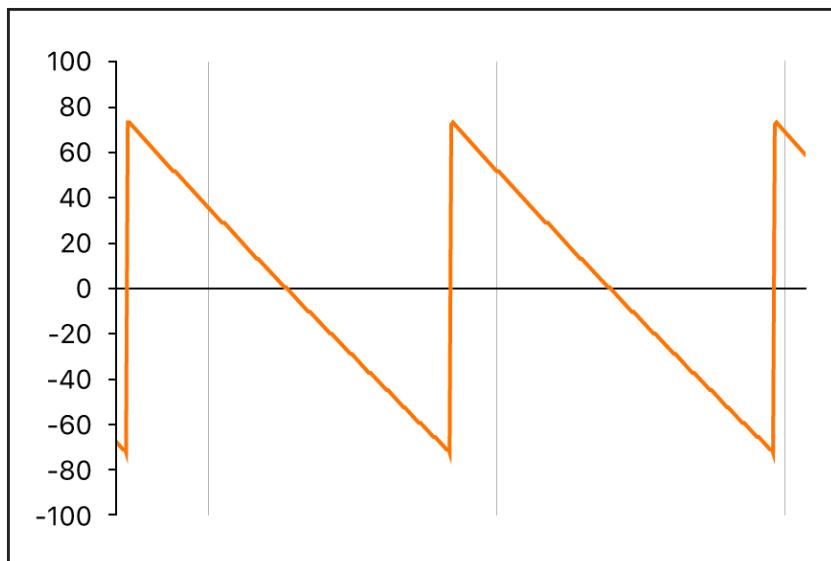


Figure 39: Signal en dent de scie non filtré

L'illustration ci-dessous présente une onde en dents de scie avec un filtre dont la fréquence de coupure est réglée sur environ 50%. Ce réglage de filtre entraîne la suppression des hautes fréquences et un "adoucissement" des angles de l'onde en dents de scie qui donnent un résultat ressemblant à une forme d'onde sinusoïdale. Ce réglage adoucit le son et le rend moins "cuivré".

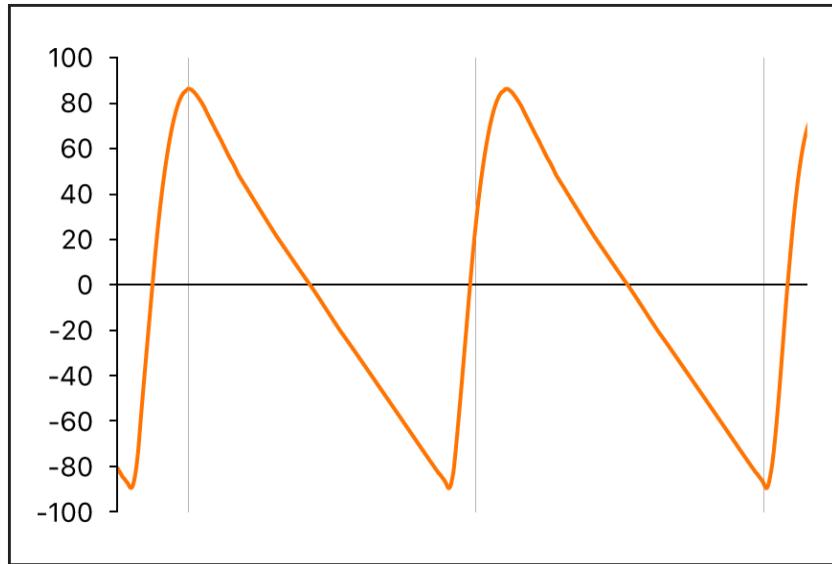


Figure 40: Signal en dent de scie après application du filtre

Cet exemple montre comment l'utilisation d'un filtre pour éliminer certaines parties du spectre de fréquences entraîne une modification de la forme d'onde et, par conséquent, un changement du timbre sonore.

**Une étape consiste donc à coder chaque filtre puis de récupérer son effet visuellement. Il faut également veiller à laisser à l'utilisateur le choix d'une fréquence de coupure de manière à pouvoir modeler le son grâce aux filtres.**

## 7.2 Design du clavier (MARIE)

Cette partie s'attache à la partie design du clavier virtuel. Nous avons décidé de le créer nous même. Nous aurions pu peut-être trouver un clavier sur les assets de Unity, mais créer nous même le clavier nous permettait d'être sûrs d'avoir les fonctionnalités voulues.

Cette partie consiste en plusieurs étapes :

- La création du clavier à l'aide Game Objects
- Le changement de couleur lors d'un appui sur une touche
- La création d'un curseur volume que l'on a finalement pas eu le temps de réaliser
- L'implémentation du son pour chaque touche

### 7.2.1 Création du clavier à l'aide Game Objects

Cette partie est facile à réaliser mais relativement fastidieuse. Il faut créer des Games Objects pour chaque note de clavier et chaque bouton de clavier ainsi que pour la plateforme.

On crée ainsi des "Cubes blancs" d'une certaines tailles pour les notes do, re, mi, fa, sol, la, si, do et des "Cubes noirs" plus petits pour les dièses.

On crée ensuite 3 cubes rouges de même taille pour les différents types d'oscillateurs.

Enfin, on crée un cube assez fin pour la plateforme.

Pour aller, plus vite, on peut dupliquer les Game Objects.

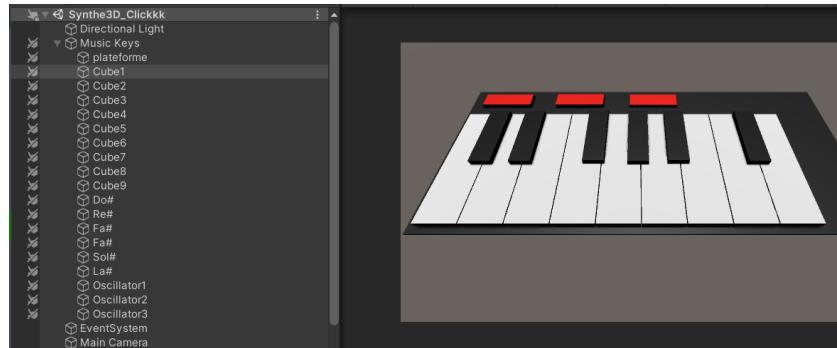


Figure 41: Design du clavier sur Unity3D avec utilisation de Game Objet

A noter qu'à la base, nous n'avions pas utiliser de "Game Objects" mais directement des "Buttons". Il est plus difficile d'implémenter des actions sur des Games Objects que sur des "Buttons". Mais les "Buttons" ne sont pas réalisables en 3d d'où l'utilisation de Game Objects.

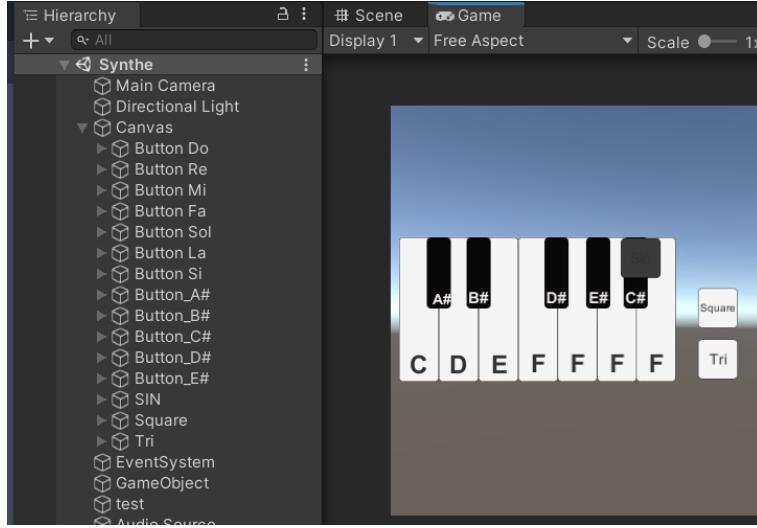


Figure 42: Design du clavier sur Unity3D avec des boutons, donc en 2D

### 7.2.2 Changement de couleur lors d'un appui sur une touche

Dans cette partie, on cherche à ce que le Game Objects change de couleur à chaque click. On impose une durée de changement de couleur à 0.1 ms.

Voici le code utilisé pour le changement de couleur :

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Colorchange : MonoBehaviour
{
    public Material[] materials; // allows input of material colors in a set sized array
    public Renderer Rend; // what are we rendering ? The Piano Key

    private int index=1;//initialize at 1, otherwise you have to press the key twice to change color at first.

    void Start()
    {
        Rend=GetComponent<Renderer>(); //gives functionality for the renderer (the key)
        Rend.enabled=true; //makes the rendered 3d object visible if enabled.
    }

    void OnMouseDown()
    {
        if (materials.Length==0) //if there are no materials present nothing happens
        {
            return;
        }
    }
}
```

Figure 43: Code changement de couleur partie 1

```

        }
        if (Input.GetMouseButtonUp(0)){
            StartCoroutine(ChangeColor());
        }

        IEnumerator ChangeColor(){
            index+=1;//when mouse is pressed down we increment up to the next index location
            if(index==materials.Length+1)//when it reaches the end of the materials it starts over
            {
                index=1;
            }
            print(index);//used for debugging
            Material mat =Rend.sharedMaterial;
            Rend.sharedMaterial=materials[index-1];
            yield return new WaitForSeconds(0.1f);
            Rend.sharedMaterial=mat;
        }
    }
}

```

Figure 44: Code changement de couleur partie 2

La particularité de ce code est que l'on utilise le système de coroutines. Une coroutine est comme une fonction qui a la capacité de suspendre l'exécution et de renvoyer le contrôle à Unity, mais de continuer là où elle s'était arrêtée sur l'image suivante. En C , une coroutine est déclarée grâce à la commande : Ienumerator fonction()

Il s'agit essentiellement d'une fonction déclarée avec un type de retour Ienumerator et avec l'instruction yield return incluse quelque part dans le corps. La ligne yield return null est le point auquel l'exécution s'arrêtera et reprendra l'image suivante. Pour définir une coroutine en cours d'exécution, on utilise la commande StartCoroutine. A la place de yield return null, on peut utiliser yield return new WaitForSeconds qui permet de mettre un délai.

### 7.3 Implémentation des notes, des paramètres et des oscillateurs (COK-ILA)

Précédemment, nous avons réalisé le design du clavier du synthétiseur, et ajouter des boutons pour les paramètres. Il s'agit maintenant, après avoir construit le clavier d'associer les touches à des fréquences, les paramètres de choix d'oscillateur et de créer un son.

But : L'objectif ici est de rendre le clavier fonctionnel. Pour cela nous allons créer trois oscillateurs (associés aux trois boutons rouges) ainsi que 15 notes (associés aux touches du clavier).

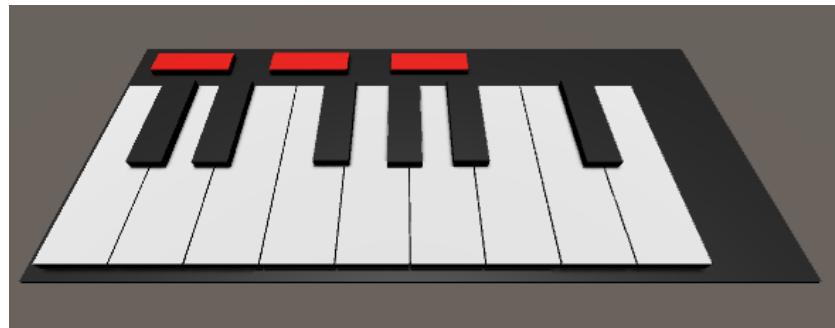


Figure 45: Design du clavier sur Unity3D

Comme décrit précédemment, l'oscillateur sera notre générateur de sons. Ce générateur de sons est codé en C# dans le fichier "Son". Voici le détail du code :

```
void OnAudioFilterRead(float[] data, int channels) {

    increment = frequency*2.0* Mathf.PI / sampling_frequency; // On crée la pulsation qui vaut 2*PI*frequence

    if(oscillator==1) { // Oscillateur sinusoïdal

        for (var i = 0; i < data.Length; i = i + channels) {

            phase = phase + increment; // On ajoute la phase à la pulsation
            data[i]=(float)(gain * Mathf.Sin((float)phase)); // On génère le signal sinusoïdal

            if (channels==2) { //Pour les canaux du son
                data[i+1]=data[i]; // this is where we copy audio data to make them “available” to Unity
            }

            if (phase>(Mathf.PI*2)){ // On reset la phase quand on a fait 4 boucles
                phase=0.0;
            }
        }
    }
}
```

Figure 46: Code de l'oscillateur sinusoïdal

```

if(oscillator==2) { // Oscillateur rectangulaire

    for (var i = 0; i < data.Length; i = i + channels) {

        phase = phase + increment; // On ajoute la phase à la pulsation
        if (gain*Mathf.Sin((float)phase)>=0*gain) { // On génère le signal créneau pour les valeurs au dessus de y=0
            data[i]=(float)(gain * 0.6f);
        }

        else {
            data[i]=(-(float)gain)*0.6f; // On génère le signal créneau pour les valeurs au dessous de y=0
        }

        if (channels==2) { //Pour les canaux du son
            data[i+1]=data[i]; // this is where we copy audio data to make them "available" to Unity
        }

        if (phase>(Mathf.PI*2)){ // On reset la phase quand on a fait 4 boucles
            phase=0.0;
        }
    }
}

```

Figure 47: Code de l'oscillateur carré

```

if (oscillator == 3)
{
    for (var i = 0; i < data.Length; i = i + channels)
    {

        phase = phase + increment;
        data[i] = (float)(gain * (double)Mathf.PingPong((float)phase, 1.0f)); // On génère le signal triangulaire grâce à la fonction PingPong

        if (channels == 2) //Pour les canaux du son
        {
            data[i + 1] = data[i]; // this is where we copy audio data to make them "available" to Unity
        }

        if (phase > (Mathf.PI * 2))
        { // On reset la phase quand on a fait 4 boucles
            phase = 0.0;
        }
    }
}

```

Figure 48: Code de l'oscillateur triangulaire

Ces oscillateurs seront activés suite à l'appui sur un bouton. Il faut pour cela associer chaque oscillateur à un GameObject puis associer le script à la touche dans Unity3D :

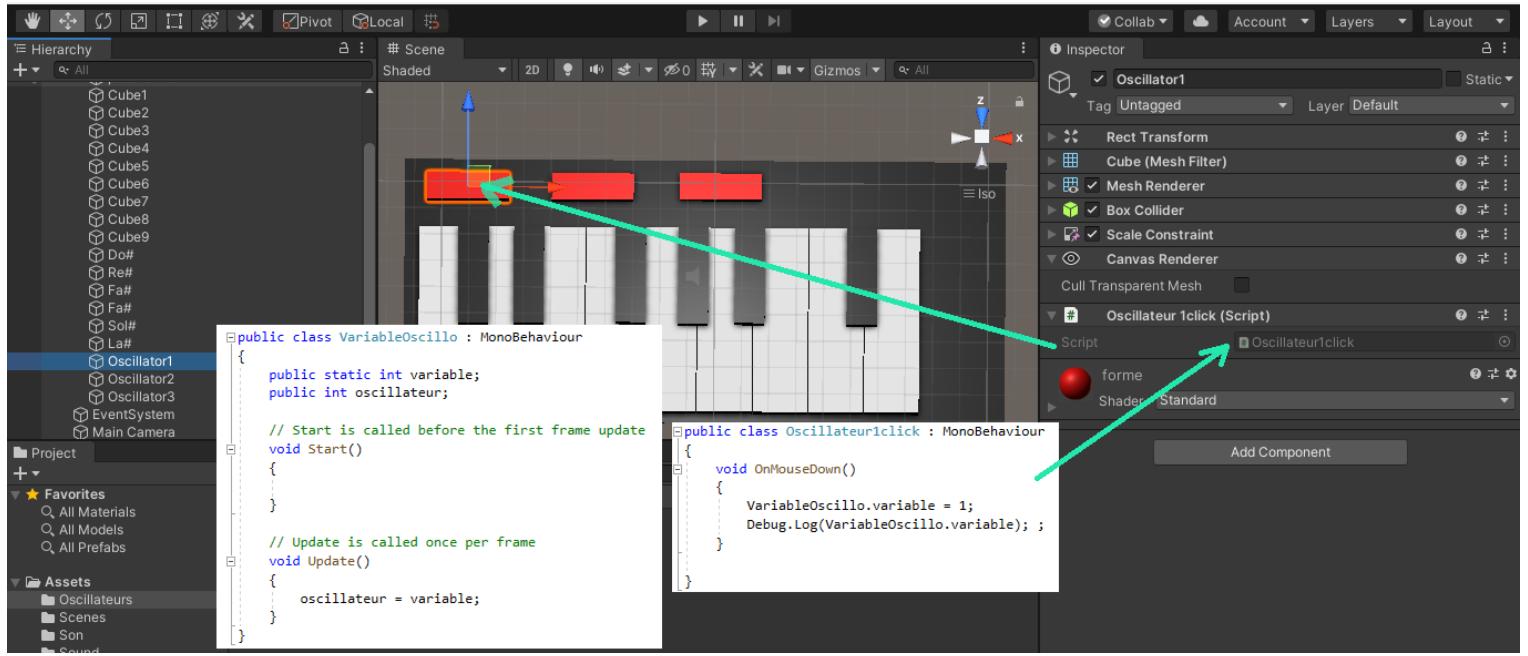


Figure 49: Association Script Oscillateur et GameObject sur Unity3D

Il faut ensuite associer les touches du clavier à des notes. Voici ci-dessous le code qui a permis d'implémenter cela. Il faut ensuite placer chacun des scripts dans le GameObject de la touche qui lui correspond comme ceci :

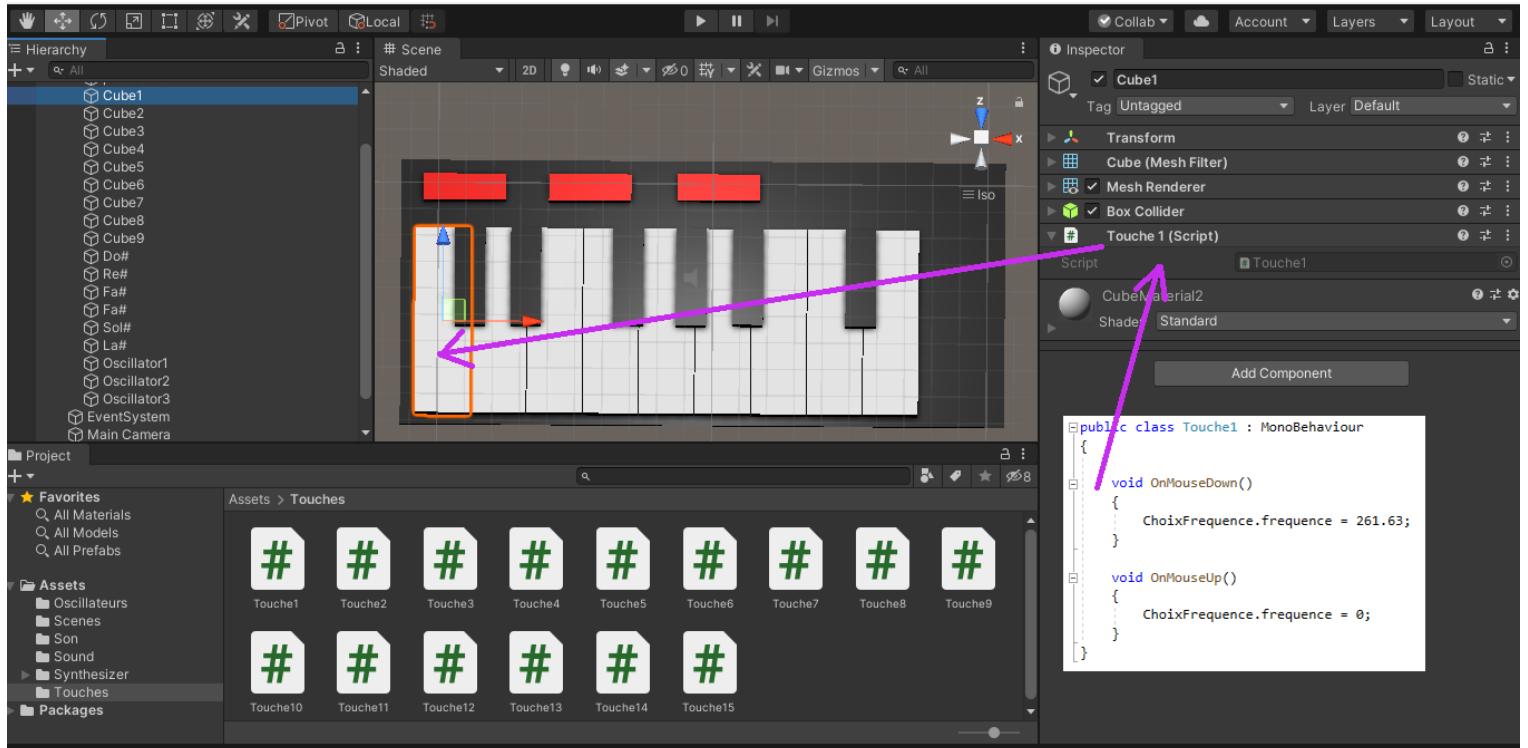


Figure 50: Association Script Touche et GameObject sur Unity3D

Les notes qui seront entendues seront la combinaison de la fréquence activée par l'appui d'une touche du clavier avec l'oscillateur choisi en amont.

C'est pour cela que les variables de l'oscillateur et de la fréquence seront importées dans le fichier "Son" et vérifiées à chaque frame :

```
public class Son : MonoBehaviour
{
    public double frequency; // fréquence donnée à l'oscillateur
    private double increment; // variable qui va permettre de se placer sur l'axe des abscisses
    private double phase; // variable qui va permettre de se placer sur l'axe des ordonnées
    private double sampling_frequency = 48000.0; // fréquence générée par Unity par défaut
    public float gain; // Puissance du son de sortie par rapport à l'entrée
    public int oscillator; // numéro de l'oscillateur choisi

    public float volume = 0.1f; // volume d'entrée

    void Start()
    {
        gain = 1; // On place automatiquement le gain à 1 au lancement de la scène
    }

    void Update()
    {
        oscillator = VariableOscillo.variable; // on cherche à savoir à chaque frame quel est l'oscillateur choisi par l'utilisateur
        frequency = ChoixFrequence.frequence; // On récupère la fréquence qui a été choisie après un clic sur une touche du clavier
    }
}
```

Figure 51: Récupération des variables précédentes

Et voilà le tour est joué !

## 7.4 Difficultés rencontrées

Les principales difficultés étaient se familiariser avec l'utilisation des GameObjects et le lancement des scripts sur Unity3D car les tutoriels à disposition ont tendance à faire l'impasse dessus.

Il a également été difficile de récupérer les données du répertoire. En effet, comme énoncé précédemment, le répertoire était créé et supprimé dix fois par seconde, ce qui ne laissait pas le temps au serveur de Unity de lire la donnée.

## 7.5 Améliorations et prochaines démarches

Ainsi, il s'agira par la suite de réussir à lire les données du répertoire afin de relier correctement les actions de l'utilisateur sur le clavier réel et les actions sur le clavier virtuel.

## 8 ANNEXES

### 8.1 Annexe 1 : Bibliographie des tutoriels et guides suivis

**UdpClient Receive Méthode:** <https://docs.microsoft.com/fr-fr/dotnet/api/system.net.sockets.udpclient.receive?view=net-5.0>

**Super explication sur le fonctionnement des synthétiseurs :**

<https://support.apple.com/fr-afri/guide/logicpro/lgsife418e65/mac>

**Bons tutoriels pour la prise en main du logiciel Unity3D :**

<https://www.bloodianchronicles.com/fr/unity-tutoriel-n1-debuter-avec-unity-3d/>

<https://www.bloodianchronicles.com/fr/unity-tutorial-n2-les-gameobjects/>

<https://www.bloodianchronicles.com/fr/unity-tutorial-n3-manipuler-les-gameobjects-par-code/>

<https://www.youtube.com/watch?v=Md7siqXr7pMlist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=1>

<https://www.youtube.com/watch?v=JHZNvMqqr4list=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=16>

<https://www.youtube.com/playlist?list=PLUWxWDlz8PYLKlr6FfwCs02DH1g2hrgS>

<https://www.youtube.com/watch?v=OBv3EU0DK4c>

**Tutoriels suivi pour la conception du clavier :**

<https://www.youtube.com/watch?v=bkE1YSSdOLUlist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=2>

<https://www.youtube.com/watch?v=VN1gryMOpWolist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=13>

<https://www.youtube.com/watch?v=r5ZV9hHi7olist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=17>

<https://www.youtube.com/watch?v=GqHFGMy51clist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=5>

<https://www.youtube.com/watch?v=kmD3n5LcHEYt=1118s>

**Tutoriels suivis pour la visualisation du son :**

<https://www.youtube.com/watch?v=rHh43ilfnyIlist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=15>

<https://www.youtube.com/watch?v=dbVz0tYfGcwlist=PLTiW7zMUkrWMArQLtsI-HbxH3nGRK56Ddindex=9>

<https://www.youtube.com/watch?v=PzVbaaxgPco>

<https://www.youtube.com/watch?v=4Av788P9stkt=107s>

**Bons tutoriels pour en savoir plus sur Unity3D mais ils n'ont pas servi pour notre projet :**

<https://www.youtube.com/watch?v=5UjwOSwnIC8t=357s>

<https://www.youtube.com/watch?v=gx0Lt4tCDE0t=209s>

**Très mauvais tutoriels car trop longs ou pas compréhensibles pour débutants :**

## **8.2 Annexe 2 : Lien gitHub**

Voci le lien du gitHub, où vous pourrez retrouver tous nos travaux !

Lien : <https://github.com/SYNTHEASYQUEST/SYNTHEASYQUEST>

## **9 Remerciements**

Merci à toute l'équipe pour l'élaboration de ce projet et pour tout le travail fourni !

Un grand merci au professeur de l'ENSEA M.REYNAL Sylvain de nous avoir donné l'opportunité de conceptualiser ce projet.

Merci également aux professeurs qui ont investi de leur temps et qui nous ont prêté main forte dans notre projet (M.BARES et M.PAPAZOGLOU).

**“Synth’easy Quest, becoming a master has never been so easy”**