VOICE DRIVEN SOUND DESIGN

Rapport de projet 3A – Design sonore

Equipe projet

Arthur MINGASSON
Harold BERJAMIN
Thomas BORDONNE
Estelle MERDRIGNAC

Client

Mathieu BARTHET

Encadrant

Cédric MAURY

CENTRALE MARSEILLE

Préambule

La dernière année d'ingénieur centralien est l'occasion pour les futurs diplômés de mener un projet en équipe. Ce travail met en œuvre gestion de projet, gestion d'équipe et ingénierie. Il fait donc la transition entre l'année de spécialisation et le milieu professionnel. En tant qu'élèves du cursus acoustique, nous avons choisi de travailler ensemble sur le design sonore.

Le designer sonore est le bruiteur du 21e siècle. Il conçoit les sons qui entourent notre quotidien et nos loisirs, des sonorités d'un intérieur d'automobile aux légendaires sons des pistolets laser de Star Wars. Le designer sonore est à la fois artiste et technicien. Le passage à l'informatique a décuplé les possibilités de production sonore mais propose parfois des solutions logicielles si lourdes qu'elles sont contreproductives.

Elèves-ingénieurs, notre projet se place logiquement du côté de la technique. Nous proposons au designer sonore un outil lui permettant d'utiliser sa propre voix pour gérer en temps-réel l'ajout de son à un film. La voix n'est donc plus l'instrument du bruiteur mais devient un contrôleur intuitif, au même titre qu'un potentiomètre ou qu'une souris d'ordinateur.

Ce projet est la continuité de l'année de césure d'Arthur M. durant laquelle il a travaillé à Queen Mary University of London sur des outils de design sonore. Le prototype VDSD pour « Voice Driven Sound Design » (design sonore contrôlé par la voix) sera donc livré à notre client, l'université londonienne, par l'intermédiaire de Mathieu Barthet, précédent tuteur de stage d'Arthur.

Etant musiciens, développer des outils de production sonore ou traitement du signal fut enrichissant et motivant. Alors que nous suivions en parallèle le master d'acoustique du LMA, nous avons néanmoins su nous organiser pour proposer un prototype abouti.

Table des matières

I. Introduction

1. Objectifs généraux

Le design sonore est une pratique qui s'est beaucoup développée depuis l'avènement de la synthèse numérique d'images, mais aussi du marketing. Notre projet se concentre sur le design sonore utilisé dans le domaine audiovisuel. Il traite de plusieurs manières d'insérer un son dans la bande sonore d'un film à l'aide de la voix.

2. Équipe projet et client

L'équipe projet se compose d'Arthur Mingasson (chef de projet), Harold Berjamin (secrétaire), Thomas Bordonné et Estelle Merdrignac. Le client du projet est Mathieu Barthet, maître de conférences à Queen Mary University of London. Nous avons eu un entretien Skype avec lui le 25 novembre afin de définir les objectifs de notre projet.

3. Gestion de projet

Pendant la phase d'avant-projet, de courtes réunions étaient organisées à l'école par le chef de projet pour communiquer sur nos recherches puis définir nos objectifs (voir partie II, qui décrit le contenu de nos réunions). Ensuite, lors de la phase projet, des réunions de travail plus longues, sur une demi-journée, ont eu lieu, pour discuter de l'avancement du projet et travailler en équipe.

Le groupe était coordonné pour l'organisation de réunions. La plateforme Dropbox a été utilisée pour le partage des fichiers de travail (comptes-rendus, fichiers de programmation, rapports, présentation externes et internes). Les comptes-rendus de réunion ont également été envoyés à notre tuteur client Mathieu Barthet et notre tuteur école Cédric Maury, pour les tenir informés de l'avancement de notre travail et recueillir leurs avis et suggestions.

La communication interne se faisait principalement oralement lors de nos cours à Centrale Marseille ou au Laboratoire de Mécanique et d'Acoustique. Nous avons créé une liste de diffusion acoustique_ecm_2015@googlegroups.com, ainsi qu'un Google Agenda pour reporter les échéances et dates de réunion. Enfin, pour éviter tout oubli, les réunions étaient rappelées par sms la veille.

II. Avant-projet

1. Étude du besoin et planification

Lors de notre entretien Skype avec Mathieu Barthet du 25 novembre 2015, les besoins des designers sonores ont été évoqués. Après une courte présentation de chacun, le projet a été lancé, avec pour objectif de proposer un prototype démontrant la possibilité de contrôler le design sonore de façon intuitive.

Suite au lancement du projet, nous avons planifié son déroulement. Le 27 novembre, nous avons établi le diagramme de Gantt ci-dessous, qui définit les phases de travail et jalons nécessaires à une réussite finale du projet. Ainsi, nous avons défini des tâches de management (WP1), de documentation (WP2-WP3), de définition du livrable (WP4), de réalisation (WP5) et de rédaction (WP6).

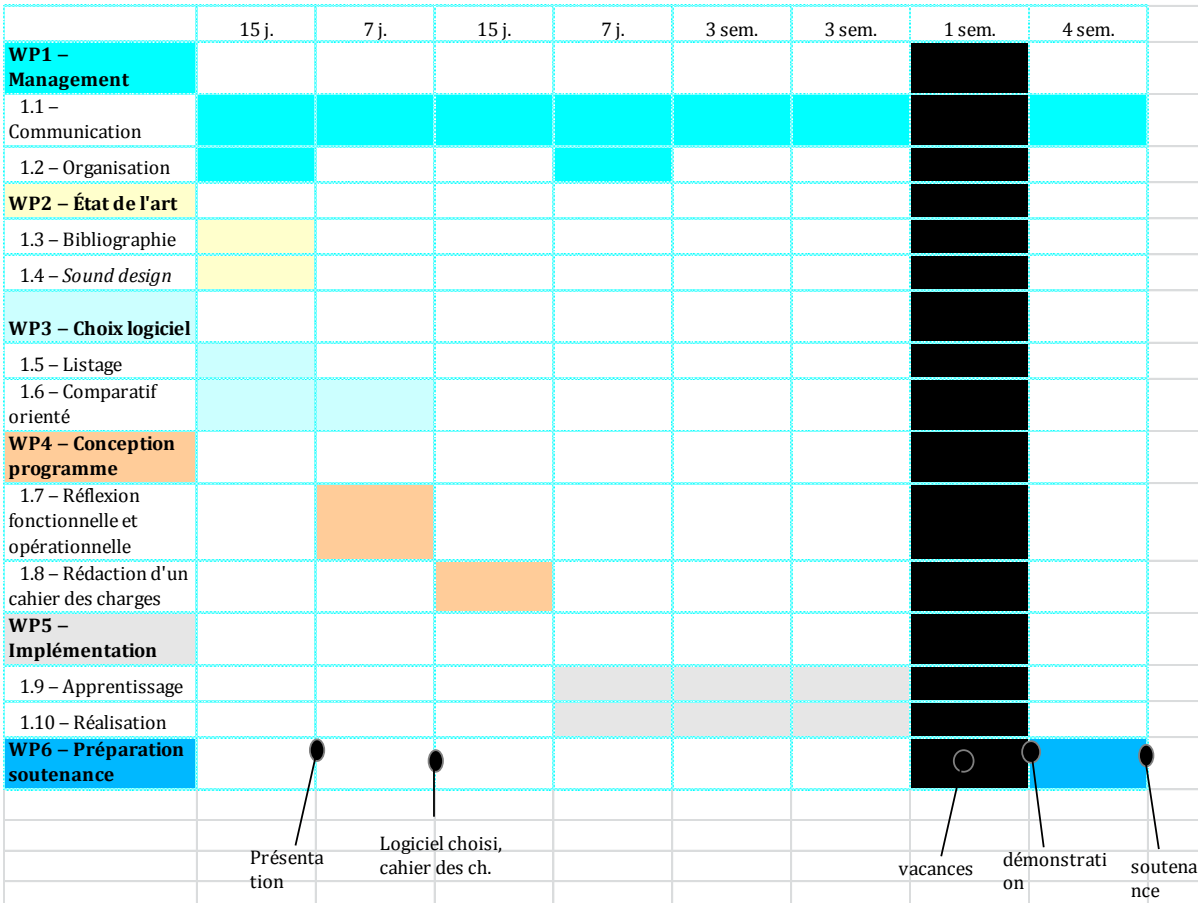

Figure 1 Diagramme de Gantt.

2. Définition du design sonore

Le projet portant sur le design sonore, il est important de connaître le métier et ses pratiques, qui seront présentés dans ce paragraphe. Les recherches sur le sujet ont été effectuées par Thomas et Estelle.

Le designer sonore est chargé de la bande son d'un film, notamment de rajouter les bruitages qui sont absents après avoir tourné et monté une scène. Ce métier s'est beaucoup développé avec l'avènement des CD-Roms, des jeux vidéo, et des films à images de synthèse, puisqu'il faut alors y intégrer tous les sons, par exemple le bruitage de machines irréelles. Avant les années 1990, le cinéma en France avait rarement recours au design sonore (c'était plus le cas au États-Unis, par exemple dans les dessins animés Walt Disney), et c'est depuis l'utilisation d'images de synthèse que son utilisation s'est diversifiée. Ainsi, dans le film Star Wars, plus de 5000 sons ont été créés, comme des sons de machines volantes, ou des sons de mécanique. Lorsque les dialogues des acteurs sont inaudibles, il faut également que le designer sonore les rajoute en post-production.

Figure 2 *Designers sonores* au travail.

Il a ainsi plusieurs missions :

- rajouter le « bon son » au « bon moment » dans le film, c'est-à-dire trouver celui qui correspond parfaitement à l'effet recherché et le placer précisément où il faut. Il a pour cela à disposition une bibliothèque de sons de tous types, auquel il peut éventuellement rajouter des filtres (atténuation, « effet métallique », etc.).
- il arrive cependant qu'il ne trouve pas le son qu'il désire dans les bibliothèques à disposition : il doit alors créer ce son avec ses propres moyens pour décrire l'action particulière qu'il recherche (par exemple, un bruit de pas sur la paille).

Ce projet s'inscrit dans le but de faire gagner du temps au designer sonore, qui passe souvent beaucoup de temps à rechercher le son adéquat dans les bibliothèques. L'objectif serait donc qu'il puisse imiter ce son et qu'il soit directement recherché dans les bibliothèques, et intégré à la bande son, mais aussi que celui-ci suive les variations de niveaux sonores, pitch et autres paramètres de sa voix, en temps réel. Notre projet se concentre sur la deuxième partie de cet objectif (à considérer à long terme), c'est-à-dire sur la modulation par la voix d'un son choisi et son rajout sur la bande son du film.

3. Recherche de moyens logiciels

En matière de moyens logiciels, l'équipe était limitée par l'accès aux licences de logiciels commerciaux. En effet, il était impensable que l'utilisation d'un logiciel engendre des frais pour le projet. Ainsi, l'équipe était restreinte à utiliser soit des logiciels libres, soit les logiciels accessibles à Centrale Marseille. La recherche de logiciels accessibles a été menée conjointement par Arthur et Harold.
Une liste de critères a été établie :

- possibilités de calcul en temps réel ;
- lecture de fichiers vidéo ;
- lecture de fichiers audio ;
- acquisition de sons ;
- traitement du signal ;
- programmation d'interfaces graphiques ;
- facilité d'utilisation ;
- possibilité de déploiement (facultatif).

Cette liste reprend les besoins minimaux pour réaliser un outil de design sonore avec contrôles intuitifs. Lors de la réunion du 11 décembre, nous avons finalement opté pour le logiciel libre PureData parmi 5 moyens de programmation comprenant Matlab et LabVIEW. PureData répond à presque tous les critères définis ci-dessus. En effet, on pourrait reprocher à PureData de ne pas permettre de produire d'interfaces très esthétiques, ni de distribuer le programme sous la forme d'une application exécutable. Toutefois, ces aspects sont facultatifs et n'empêchent pas la réalisation d'un prototype.

4. Rédaction d'un cahier des charges

La réflexion sur les fonctionnalités du prototype a été menée de manière collective, chacun pouvant y apporter ses idées. Le 12 janvier, nous avons retenu les fonctions suivantes :

- permettre à l'utilisateur de lire une vidéo (marche avant, arrière, avance rapide, pause) ;
- permettre à l'utilisateur d'écrire une bande son de bruitages usuels (spatialisés), en même temps qu'il regarde la vidéo ;
- permettre à l'utilisateur de lire les bandes sons et la vidéo simultanément.

III. Réalisation

1. Définition des sous-objectifs

Le 12 janvier, nous avons établi une liste de sous-objectifs à atteindre pour achever la réalisation de notre prototype à l'aide de PureData :

- la réalisation d'un lecteur vidéo et audio avec les contrôles nécessaires ;
- la gestion des entrées/sorties de fichiers, offrant la possibilité d'ajouter des éléments pré-enregistrés ou synthétisés à une bande son originale ;
- le traitement des signaux vocaux émis par l'utilisateur, notamment la détection de transitoires, la détection de la fréquence fondamentale (pitch frequency), la spatialisation gauche/droite des signaux audio à deux voies (facultatif) ;
- la production de sons non-enregistrés : réalisation de quelques synthétiseurs, contrôlables à partir d'un signal vocal.

2. Répartition des tâches et planification

PureData est basé sur le même principe que Max/MSP. Arthur et Thomas ayant déjà programmé à l'aide de Max/MSP, les tâches les plus délicates leur ont été attribuées, c'est-à-dire la gestion des fichiers, le traitement du signal vocal et la synthèse audio. La réalisation des lecteurs audio et vidéo a été confiée à Estelle et Harold, qui faisaient leurs débuts avec ce type de moyens de programmation. Un premier jalon avait été fixé le 14 février. Cette date permettait de planifier un éventuel travail de finition, à réaliser au cours des vacances d'hiver (21 février – 1er mars 2015).

3. Présentation du logiciel sélectionné Pure Data

Pure Data est un logiciel de programmation graphique. Comme mentionné précédemment, il fonctionne de la même manière que Max/MSP, dont il est la version libre. C'est un logiciel dédié initialement au traitement et à la synthèse de signaux audio, mais des *add-ons* (programmes additionnels) ont été développés pour pouvoir travailler également avec de la vidéo et des images.

La programmation graphique permet de remplacer la syntaxe de programmation par des blocs –des objets– reliés entre eux dans un espace de travail. Ce langage de programmation permet donc une prise en main instinctive. Il nous a donc permis de commencer à programmer rapidement et facilement.

Chaque espace de travail est sauvegardé et contient un ensemble de blocs reliés entre eux par des fils. Cet ensemble s'appelle une abstraction, ou un patch. On trouve dans un patch des blocs reliés entre eux. La position de ces blocs est très importante. En effet, la « lecture » d'un patch, son exécution, se fait de droite à gauche et de haut en bas. Lorsque l'on souhaite effectuer plusieurs opérations successives, respecter cet ordre est donc important.

Chaque bloc réalise une fonction plus ou moins complexe. Tous les blocs possèdent aux moins une entrée et une sortie, dont le nombre est variable suivant le type de fonction qu'il réalise. On trouve deux types de fonction, celle opérant sur un signal audio, et celle

opérant sur des données. Au niveau de l'affichage, les premières sont repérées par un ~ et des entrées et sorties bleutées, alors que celles des secondes restent noires. Cela permet de différentier facilement le type de données que l'on transmet et que l'on traite.

Chaque patch peut être muni d'entrées et de sorties extérieures (via des blocs *inlet* et *outlet)*, et ainsi être appelé dans d'autres patchs sous forme de bloc. Il devient alors un sous-patch. Cette fonctionnalité permet de créer des fonctions n'existant pas dans Pure Data et de faciliter la lecture d'un patch.
Plusieurs signaux peuvent transiter entre des blocs :

- le « Bang ». Il s'agit d'un 1 envoyé impulsivement. Il permet de déclencher des actions.
- les données. Il s'agit du type de signal le plus classique, et elles peuvent être constituées soit de nombres, avec ou sans décimales, de mots, ou de listes de lettres ou de nombres. Les images font partie de ce type.
- les signaux audio. Ils sont spécifiques aux fonctions portant un ~, et sont repérés entre les blocs par des fils bleutés.

4. Présentation du travail réalisé

a. Production sonore

Le but de notre projet étant de faire du design sonore, nous avons donc dû synthétiser des sons. Pour cela, nous nous sommes appuyés sur des programmes tirés de l'ouvrage de référence d'Andy Farnell sur le design sonore[1], que nous avons modifiés de manière à pouvoir les contrôler en temps réel à partir d'une même interface générale.

Lors de la phase d'avant-projet, nous avons mis en avant 4 catégories de sons sur lesquels travailler :

- Sons ponctuels sans paramètres, comme par exemple le bruit d'un pistolet ou le claquement d'une porte.
- Sons à contrôle ponctuel, comme des bruits de pas.
- Sons à contrôle ponctuel avec mise à jour qui servirait pour définir le début, l'évolution et la fin d'une ambiance. Une ambiance ventée, la pluie, des chants d'oiseaux ou la mer, par exemple.
- Sons à contrôle continus, à l'instar du bruit de moteur, du hurlement de loup ou encore du vent.

Pour la sélection de ces sons, l'idéal aurait été que le programme détecte automatiquement le type de son et lance le contrôle approprié. Nous nous sommes rendu compte plus tard que les nuances de la voix n'était pas suffisamment détectables et dépendaient de l'utilisateur. A titre illustratif, personne n'imitera de la même manière une voiture et un avion. On devrait alors définir un « standard » pour chaque son, de manière à éviter toute confusion. De plus, un son peut être présent dans deux catégories, le vent par exemple, qui peut à la fois être considéré comme un son d'ambiance, ou un son que l'on contrôle directement. Il aurait aussi été nécessaire de définir des catégories entre les mêmes types de sons pour éviter à nouveau la confusion.

[1]Farnell, Andy. *Designing Sound*. s.l. : The MIT Press, 2010.

Nous avons donc fait le choix de sélectionner les sons à partir d'une liste, d'une part pour simplifier leur utilisation, et d'autre part, pour éviter les confusions mentionnées précédemment. Seules les catégories « Sons à contrôle continu » et « Sons ponctuels sans paramètres » ont été implémentées dans un premier temps, les autres pourront l'être *a posteriori*.

Pour les sons à contrôle continu, nous avons mis au point des sons de moteurs et un son d'ambiance ventée. Pour les premiers, seule une fréquence contrôle la vitesse du moteur, mais pour le vent, une amplitude contrôle le niveau de sortie et une fréquence contrôle la vitesse.

Pour la seconde catégorie, les sons ponctuels sans paramètres, nous avons utilisé un claquement de porte. Son contrôle est plus simple, puisque seul un ordre est nécessaire, un « Bang » (cf Partie III.3), déclenché sur une transitoire de la voix.

b. Traitement du signal

Les trois informations utiles pour contrôler ces sons (fréquence, amplitude et attaque) sont directement extraites de la voix. Pour ce faire, nous avons développé un patch extrayant directement ces trois paramètres : le « *pitch tracker* ». Grâce à un outil de Pure Data, le *fiddler*, qui extrait l'enveloppe et l'amplitude d'un signal, couplé à un microphone, nous obtenons ainsi la fréquence fondamentale de la voix, l'amplitude de celle-ci en dB, et un bang à chaque transitoire.

Pour contrôler plus précisément ces paramètres, nous avons également mis au point un patch se chargeant de la détection d'activité, afin d'éviter tout artefact lié aux sons environnants, le « *activity detection* ». Grâce à un seuillage défini empiriquement, le patch renvoie un 0 ou un 1, ce qui permet ainsi, en le couplant avec le premier patch, de ne garder que le signal utile.

Enfin, nous nous sommes également servi d'un normalisateur, qui, comme son nom l'indique, nous sert à normaliser à la fois la fréquence et l'amplitude. Le contrôle de chaque synthétiseur de son en est alors simplifié. Cette normalisation se fait à partir de la moyenne d'un signal que l'on enregistre, et qui vient diviser ensuite à chaque instant la valeur en entrée du patch.

Nous avons ensuite rassemblé ces trois patchs dans un seul et unique patch, main_signal_processing, qui permet donc d'avoir en sortie les signaux d'attaques seuillés, l'amplitude, ainsi que la fréquence normalisée et seuillée. Le contrôle des synthétiseurs est alors rassemblé dans un seul patch, ce qui simplifie grandement la programmation et l'utilisation.

c. Lecture et synchronisation audio/vidéo

L'objectif de cette partie était de réaliser un lecteur audio ainsi qu'un lecteur de vidéo, puis de les synchroniser afin de réaliser un lecteur de vidéo complet, auquel on puisse rajouter une bande son à un moment donné (voir Partie Interface Principale)

Lecteur Audio
Le lecteur audio a été réalisé en utilisant l'objet *tabplay~* de Pure Data, qui permet de lire un signal stocké dans un tableau, en partant de la ligne dont le numéro est indiqué en entrée. On peut aussi arrêter la lecture facilement. Le fichier .wav est donc chargé, puis stocké dans un tableau. Sa fréquence d'échantillonnage est de 44.1 kHz. On obtient alors un sous-patch avec en entrée un bouton stop, le nom du fichier à jouer, et l'échantillon de départ ; en sortie le nombre total d'échantillons et la sortie audio.

Lecteur Vidéo
Le lecteur vidéo se base sur la bibliothèque GEM de Pure Data, qui permet de lire des fichiers vidéo sans son. Cette bibliothèque utilise les codecs du système d'exploitation, ce qui peut causer certains problèmes de compatibilités (voir Partie problèmes rencontrés). Nous avons utilisé la fonction *pix_movie* de GEM, qui ouvre une fenêtre, lit une vidéo de format type .AVI, ou .MOV dans certains cas (dépendant des codecs de décompression installés sur la machine).

Le fichier vidéo .avi est chargé et son adresse est donnée en entrée de la fonction *pix_movie*. Cette fonction prend également en entrée : la fonction *gemhead***,** qui gère le clip vidéo ; un interrupteur 0/1 qui démarre/arrête la lecture ; le no. de l'image où démarre la lecture. En connaissant la taille de la vidéo (qui est récupérée en sortie de la fonction *pix_movie*), on peut donc déterminer à partir de quelle image on démarre la lecture. On peut également régler la taille de la fenêtre de lecture via la fonction *gemwin*. Ceci permet de l'adapter à la taille des images en pixels. La fonction *gemwin* permet également de régler la vitesse de lecture (fixée par défaut à 25 images par seconde) et de fermer la fenêtre de lecture.

On obtient alors un sous-patch avec en entrée un bouton play, un bouton pause, le nom du fichier à jouer, le numéro de l'image de départ, et un bouton pour fermer la fenêtre du lecteur ; en sortie un voyant de fin de lecture, le nombre d'images total de la vidéo, et le numéro de l'image de départ.

Lecteur complet (synchronisation des deux lecteurs)
Afin de synchroniser les lecteurs, il faut tout d'abord qu'ils soient lancés en même temps : on relie donc le bouton Play de la vidéo à une entrée du lecteur audio afin de les déclencher simultanément. De même pour l'arrêt de la lecture. Pour que la lecture soit synchrone, les numéros des échantillons de départ audio et vidéo doivent correspondre. Avant de démarrer la lecture, le numéro de l'image de départ est divisée par le nombre total d'images, le résultat est multiplié par le nombre total d'échantillons audio, dont la partie entière correspond au numéro de l'échantillon audio associé.

On obtient un sous-patch avec en entrée les fichiers vidéo et audio à lire, un bouton pour fermer le lecteur vidéo, un bouton play, un bouton pause, et la position relative du curseur de lecture (nombre compris entre 0 et 1). En sortie, on calcule le numéro d'échantillon à un instant donné grâce au chronomètre (objet *metro*). Celui-ci est incrémenté toutes les 100 ms et est remis à zéro à chaque lecture. À la pause de la vidéo, son résultat est multiplié par 4410, puis rajouté au numéro d'échantillon audio de départ, ce qui donne le numéro de l'échantillon audio courant. En divisant par le nombre total d'échantillons audio, on obtient la position de cet échantillon dans la vidéo sur une échelle de 0 à 1.

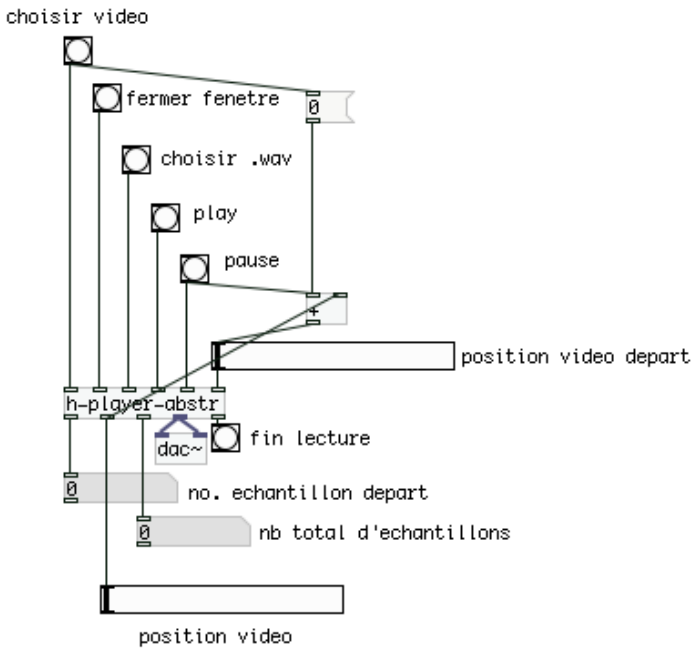

Figure 3 Lecteur vidéo/audio sous Pure Data, avec ses entrées et sorties.

Finalement, on obtient le patch ci-avant, composé du sous-patch des lecteurs synchronisés, avec ses entrées (qui sont pour la plupart des boutons bang) et ses sorties (différentes informations sur la position de l'échantillon lu, la sortie audio et un bang de fin).

d. Enregistrement et exportation de données

Le principe est relativement simple. On extrait la taille de la vidéo en nombre d'images. On crée ensuite un tableau de même taille. Lorsqu'on lit la vidéo, le même tableau est lu en même temps. Ainsi, lorsqu'on décide d'enregistrer des sons, on vient remplir le tableau à la position courante. On obtient ainsi une bande sonore qui peut se superposer à la première. On choisit d'exporter les résultats en enregistrant le tableau dans un fichier .WAV. Au départ, nous voulions également créer un fichier contenant uniquement les instants où un son a été déclenché par la voix. Cette amélioration pourra être intégrée facilement dans une version ultérieure.

e. Interface principale

On a précédemment décrit les différentes briques constitutives du système. Grâce à cette division, nous avons pu travailler en même temps sur notre logiciel sans être trop dépendant des avancées des autres :

- lecteurs audio/vidéo : lecture du film et de sa bande-son originale
- traitement du signal : extraction de paramètres (temps, fréquence et transitoires) sur le signal du microphone
- production sonore : mapping de ces paramètres extraits vers différents synthétiseurs ou échantillonneurs

- enregistrement : sauvegarde de la bande-son d'effets créée via la « production sonore »

L'interface principale réunit et connecte tous ces éléments. Contrairement à son concurrent payant Max/MSP, Pure Data ne propose pas de mode « présentation » permettant de n'afficher que ce qu'on le souhaite. On est donc contraint d'afficher les connexions et les différents sous-blocs, ce qui nuit à la clarté des applications.

Nous avons tout de même fait un important effort pour pouvoir aboutir à une interface à la fois claire pour un utilisateur non averti, mais aussi complète pour qu'un expert puisse facilement accéder aux contrôles plus bas-niveau du prototype.

La figure ci-dessous présente l'interface. On peut distinguer la partie supérieure qui contient les contrôles principaux. On peut se contenter de cette partie pour utiliser le logiciel. Dans la partie inférieure, on trouvera les différents modules (Lecteur Vidéo, Traitement du signal de voix, etc.) auxquels on peut accéder d'un simple clic.

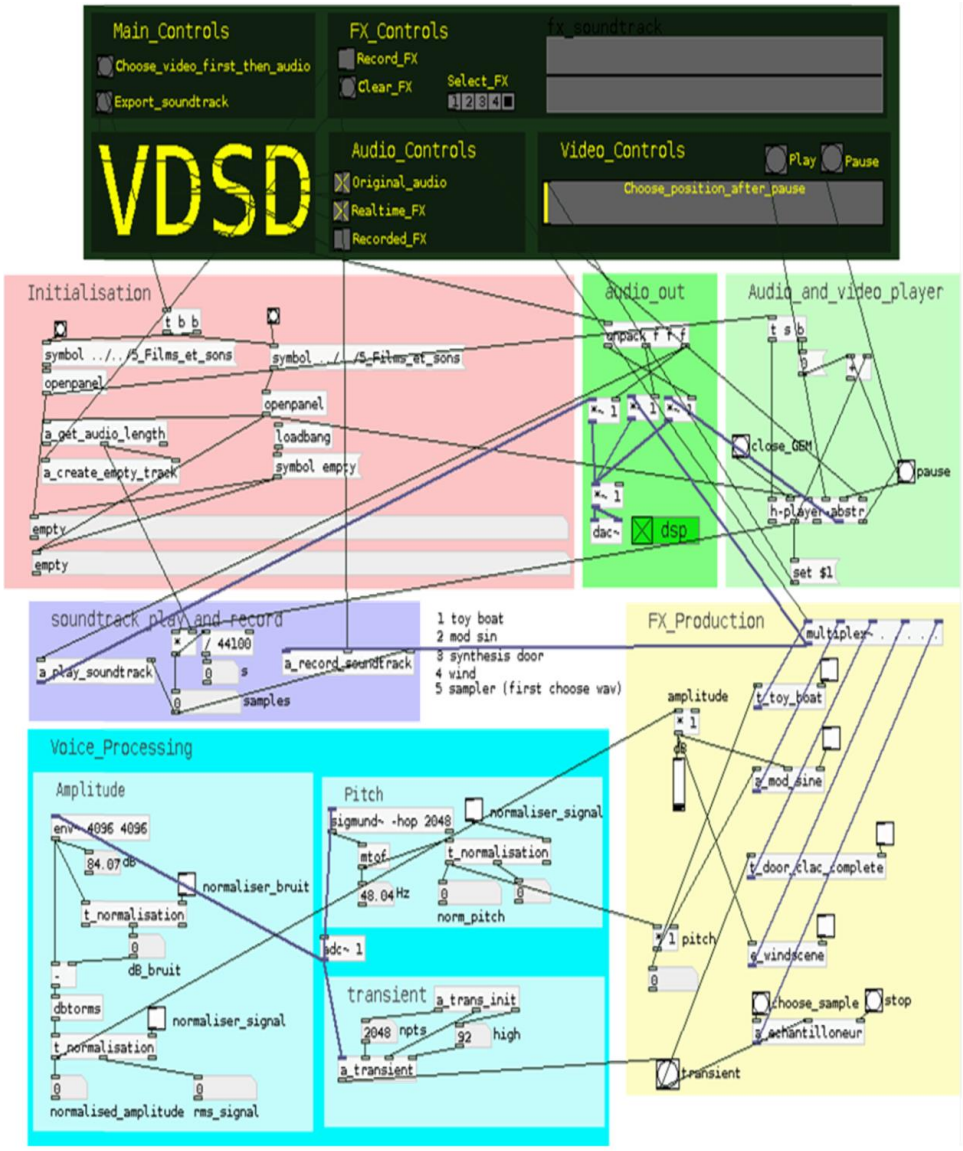

Figure 4 Interface principale

Conclusion

1. Bilan

a. Objectifs non atteints et remise en question

L'objectif de ce projet est de réaliser un programme Pure Data qui prenne en entrée un signal voix (par le biais du micro) qui, soit contrôle en fréquence et en amplitude un son de synthèse (par exemple, de vent), soit déclenche des évènements ponctuels. Ce son doit ensuite être enregistré en temps réel et rajouté à la bande son d'une vidéo, qui est lue par le programme.

Le contrôle en fréquence est fonctionnel, mais celui en amplitude paraît encore instable à la fin du projet. Une difficulté réside aussi dans la phase d'enregistrement, puisqu'on enregistre de manière non continue sur une bande son qui est lue en parallèle de la bande son originale de la vidéo. De plus, la synchronisation entre les deux n'est pas optimale. Enfin, nous avons remarqué que le programme n'était pas robuste. Il se produit parfois des opérations non désirées. Par exemple, un sifflement se produit lorsqu'on allume un oscillateur.

Nous avons été relativement ambitieux dans la phase d'avant-projet car nous pouvions difficilement évaluer la complexité de la programmation sous Pure Data. Nous avons heureusement su revoir nos objectifs au cours du projet et supprimer quelques fonctions proposées dans le cahier des charges.

b. Points positifs dans la programmation et apport du projet

L'architecture du programme est bonne dans l'ensemble, si l'on met de côté les problèmes inhérents rencontrés dans les sous-patchs. Le lecteur vidéo, ainsi que le programme de modulation en amplitude par la voix ont été testés séparément et étaient fonctionnels.

Ce projet nous a aussi permis de développer nos compétences en programmation Pure Data. Nous avons aussi développé nos compétences d'organisation en répartissant nos tâches en fonction de nos compétences, afin que chacun fasse un travail qui lui soit adapté, et en travaillant principalement en groupe pour optimiser notre productivité.

c. Gestion des problèmes

Les difficultés rencontrées concernent principalement les bugs du logiciel Pure Data. Il suffit parfois de relancer le logiciel pour pallier le problème. Des difficultés ont également été rencontrées avec le module GEM de Pure Data (qui gère la vidéo) lorsque l'on travaille sur un système d'exploitation différent. En effet, ce module utilise les bibliothèques de codecs propres au système d'exploitation. Que l'on travaille sous Windows, OSX, ou Linux, les résultats d'un même programme ne sont pas identiques : par exemple, sous OSX, une vidéo .AVI est lue beaucoup plus rapidement par GEM, car le décodage de la vidéo n'est pas fait correctement. La synchronisation audio/vidéo est alors complètement faussée. Il a été vérifié par d'autres utilisateurs que les vidéos .AVI sont bien lues sous Windows et les vidéos .MOV sous OSX. Quant à Linux, cela dépend des codecs installés par l'utilisateur, mais il peut *a priori* gérer un grand nombre de

formats. Après constatation de cette variabilité entre les systèmes, nos programmes utilisant GEM ont donc été réalisés sous Windows avec des vidéos .AVI.

2. Perspectives

Mathieu Barthet travaille sur des descripteurs émotionnels et descriptifs du son. Ce que l'on appelle le Music Information Retreival (MIR). Une perspective possible de ce projet serait l'intégration de méthodes de reconnaissance du son telles que M. Barthet les étudie, par exemple pour remplacer la liste de sélection que nous utilisons. Cela permettrait une utilisation plus intuitive du programme.

Ce projet a pour objectif final de permettre au *sound designer* de commander l'ajout de sons sur une bande film et de moduler ce son avec sa voix. Un plus grand projet, intitulé SkAT-VG (Sketching Audio Technologies using Vocalizations and Gestures) et réalisé par l'IRCAM et d'autres partenaires, est en cours. Il s'inscrit dans cette problématique de faciliter le travail des *sound designers*.

De nombreuses améliorations sont encore réalisables sur notre projet, et notamment la possibilité de contrôler d'autres paramètres par la voix. Le contrôle en fréquence peut également être amélioré. Actuellement, il fonctionne bien pour un sifflement, mais est moins robuste pour un signal parlé. Les sons qu'on peut ajouter et contrôler sont pour le moment restreints à une sinusoïde, un moteur, du vent et des bruits de pas. La bibliothèque peut donc être élargie à une plus grande gamme de sons.