UNIVERSITÉ CÔTE D'AZUR

MASTER INFORMATIQUE

Rapport de TER

Ecriture d'une application "multi-effets audio à base de plugins de type WebAudio Modules 2.0"

Réalisé par
Kouyoumdjian Pierre
Marynowicz Michael
Beauchet Quentin
Forner Yann

Encadré par
Michel Buffa

Master Informatique
à Université Côte d'Azur
Second semestre de l'année 2021-2022

Table des matières

Première partie
Introduction

1 Le groupe

Notre groupe pour réaliser ce projet comporte 4 étudiants en Master 1 Informatique :

— Quentin Beauchet
— Yann Forner
— Pierre Kouyoumdjian
— Michael Marynowicz

2 Présentation du sujet

Le but de ce TER était de créer un PedalBoard grâce au SDK Web Audio Module 2 développé par M. Michel Buffa et son équipe. Un PedalBoard est une planche sur laquelle sont fixées plusieurs pédales d'effets reliées en série. Elles permettent de faire varier le son d'un instrument de musique.

Le résultat final consiste en un PedalBoard réalisé en JavaScript qui peut charger plusieurs pédales. Le son passe à travers cette chaîne de gauche à droite. Il est altéré après chaque pédale. La conception permet au PedalBoard d'être lui même une pédale et donc d'être lui aussi inséré dans une chaîne audio.

FIGURE 1 – Un PedalBoard avec des pédales d'effets électroniques

Deuxième partie
État de l'art

3 Contexte

La Musique Assistée par Ordinateur (MAO) est apparue dans la fin des années 90 avec l'Atari ST, puis s'est développée rapidement en imposant ce que l'on appelle les "stations de travail audionumériques", également appelés "sequencers audio" ou encore DAW en anglais (pour Digital Audio Workstations).

Ce sont des logiciels complexes permettant d'enregistrer des chansons piste par piste. On peut enregistrer la voix, puis un piano, puis une guitare, etc. Et retoucher le tout à l'aide d'effets audio en temps réel. Par exemple ajouter une réverbération à la voix, ajouter un effet d'echo à la guitare etc.

Ces logiciels proposent également des instruments virtuels : on utilise un clavier MIDI qui envoie des événements (quelle touche a été appuyée, avec quelle force) et un synthétiseur logiciel va générer le son. On n'enregistre initialement que les événements, les sons pouvant être modifiés à postériori dans le logiciel.

Rapidement, Steinberg a proposé un standard pour des plugins d'extension de sa station de travail audionumérique Cubase, le standard VST pour Virtual Studio Technology. Cubase faisait officiel de logiciel hôte et les effets et instruments chargés sous forme de plugins. Depuis cette époque le marché s'est articulé autours de quatre stations de travail qui dominent le marché : Cubase, Logic Audio (Apple, avec une version simplifiée gratuite qui équipe tous ses ordinateurs, iphones, ipads : GarageBand), Ableton Live et Pro Tools. Et des dizaines de milliers de plugins, gratuits ou commerciaux, ont été développés. Quatre standards de plugins natifs, propriétaires, sont associés à ces logiciels hôtes, et souvent les développeurs doivent compiler et adapter leurs plugins pour ces différentes cibles.

Depuis 2009 le W3C travaille sur le standard Web Audio, qui a pour but d'élargir les possibilités de traitement sonores dans les navigateurs Web. La WebAudio API permet depuis du code JavaScript ou WebAssembly chargé dans une page Web, de développer des applications variées. Par exemple, les logiciels de visio conférence comme Teams ou Zoom, sont des applications Web, et WebAudio est utilisé pour supprimer l'écho ou le bruit dans la voix. Elle permet de produire des effets sonores dans les jeux, de la musique contextuelle, etc. Mais également, elle permet d'apporter la Musique Assistée par Ordinateur dans les navigateurs, la personne à l'origine de la première spécification étant Chris Rogers alors chez Google, qui a travaillé auparavant de nombreuses années chez Apple, et co-concepteur de Logic Audio.

La WebAudio API propose des briques de bas niveau, des "blocs de traitement du son" que l'on assemble dans un graphe appelé "audio graph". A la fréquence d'échantillonage (44100 fois par seconde), le son "avance" dans le graphe par bloc de 128 échantillons sonores (l'audio buffer) et chaque noeud peut modifier ces échantillons. On trouve des noeuds pour changer le volume (gain), pour filtrer le son (ajouter des graves, médium, aigus, etc), pour ajouter de la distorsion, pour retarder la sortie du son etc. On peut même faire des boucles dans le graphe (pour des effets d'echo).

De très nombreuses applications ont été développées entre 2010 (date des premières implémentations dans les navigateurs) et 2015, mais il n'existait pas de standard pour développer des plugins ré-utilisables et inter-opérables. Il n'existait pas non plus de station de travail audio numérique en ligne. En 2015, Jari Kleimola et Olivier Larkin Kleimola et Larkin (2015) ont proposé les Web Audio Modules (WAM), un standard pour des plugins Web Audio, et Jari Kleimola a participé à la création d'AmpedStudio.com, un des premiers sequencers en ligne. Deux autres sont apparus : Bandlab.com et SOundtrap.io (qui appartient à Spotify). Depuis 2015 Michel Buffa et d'autres chercheurs, aidé par des développeurs issus de l'industrie de la MAO, ont fait évoluer ce standard de plugins pour proposer en 2021 une version 2.0 des Web Audio Modules Buffa *et al.* (2022).

De nombreux plugins ont été développé, notamment des effets audio reproduisant les pédales d'effets que les guitaristes utlisent, et des simulations d'amplificateurs de guitare à lampes Buffa *et al.* (2016).

4 Web Audio Module 2

C'est un standard développé par M. Michel Buffa et son équipe qui sert de référence pour le développement de pédales d'effet sur le web. Il est écrit en JavaScript et le code source est trouvable intégralement sur github.
On ne parle donc plus de pédale d'effet mais de plugin.

Le SDK supporte différentes chaînes de compilation. Il est donc possible d'avoir des plugins fait en C,C++, Faust ou Csound qui sont des langages de programmations dédiés au traitement de signal audio. Ces langages sont ensuite compilés vers du web assembly.
Il est aussi possible d'écrire le plugin directement en JavaScript. C'est le cas de notre PedalBoard.

5 Les plugins

Chaque plugin est composé de deux parties. La première partie est la Gui. Elle correspond à l'élément HTML qui sera affiché sur la page web sous la forme d'un Web Component. Cela signifie que c'est un élément qui à son propre Css et JavaScript indépendant du reste de la page. Ceci nous permet de l'importer très facilement. C'est depuis la Gui que l'utilisateur va pouvoir modifier les valeurs des effets grâce à des boutons.

La deuxième partie est appelée la Node. Elle implémente la Web Audio API de JavaScript qui facilite la manipulation de l'audio sur le web.

Les plugins peuvent être utilisés grâce à un host. C'est un script JavaScript qui initialise le contexte audio du navigateur. Il connecte aussi le plugin entre l'entrée et la sortie audio et affiche la gui de celui-ci sur la page.

Troisième partie
Travail effectué

6 Le PedalBoard

6.1 Les fonctionnalités de notre PedalBoard

Tout d'abord, notre PedalBoard possède une zone de sélection avec les miniatures des différents plugins, cette zone se nomme la preview. Lorsque nous cliquons sur la miniature, le plugin correspondant est ajouté à la chaîne audio. Cela permet d'initialiser la Gui que l'on va ajouter sur ce que l'on appelle le Board. C'est l'équivalent de la planche en bois sur laquelle sont fixées les pédales d'effet.
Il est aussi possible de filtrer ces miniatures grâce aux mots-clés de leurs plugins situés dans leurs fichiers descriptor.json respectifs.

Une fois sur le Board, on peut bien évidement modifier les effets des plugins grâce à leurs boutons. Il s'agit d'une fonctionnalité de base du SDK. Nous avons aussi rajouté la possibilité de supprimer un plugin de la chaîne ou de changer l'ordre de celle-ci avec un "Glisser-déposer".

En plus de cela, nous avons implémenté une partie presets. Cela permet de sauvegarder la chaîne courante et la position des boutons des différents plugins dans un système de catégories et de sauvegardes respectivement appelés Banks et Presets. Il est ensuite possible de charger ses presets, de les modifier ou de les supprimer.

6.2 L'ajout des plugins

Avant d'ajouter des plugins, il faut connaître la liste des plugins disponibles. C'est le fichier index.js qui réalise cette opération. Nous avons créé un fichier nommé repositories.json qui contient une liste de liens. Ils pointent chacun vers des fichiers json contenant une liste de plugins.

Nous avons également le fichier wams.json qui contient une liste de plugins à importer. Cela permet au PedalBoard de se passer de serveurs si besoin mais nous expliquerons cette dynamique plus en détails dans la section 8.

Algorithme 1 Contenu de notre fichier repositories.json

```
1   ["./wams.json", "https://wam-bank.herokuapp.com/wams"]
```

Algorithme 2 Contenu de notre fichier wams.json

```
1   [
2     "https://mainline.i3s.unice.fr/wam2/packages/quadrafuzz/dist/",
3     "https://mainline.i3s.unice.fr/wam2/packages/faustPingPongDelay/plugin/",
4     "https://mainline.i3s.unice.fr/wam2/packages/pingpongdelay/dist/",
5     "https://mainline.i3s.unice.fr/wam2/packages/StonePhaserStereo/",
6     "https://mainline.i3s.unice.fr/wam2/packages/TS9_OverdriveFaustGenerated/",
7     "https://mainline.i3s.unice.fr/wam2/packages/BigMuff/",
8     ...
9   ]
```

Depuis Javascript, nous faisons des requêtes aux différents serveurs puis sur les différents plugins qu'ils contiennent ce qui nous permet de les importer. La difficulté à ce niveau là est d'attendre la fin de chaque requête car elles ont des durées différentes à chaque exécution.

Une fois les requêtes terminées, on stocke les modules obtenus après leurs importations. De plus, nous stockons leurs fichiers descriptors.json qui nous permettront d'obtenir les mots-clés, les miniatures et les noms des plugins dans un objet de la classe PedalBoardPlugin appelé WAMS.

FIGURE 2 – Exemple de filtrage avec le mot-clé Faust

Vu que les clés de notre objet WAMS sont les noms de nos plugins cela implique que deux plugins ne peuvent pas avoir le même nom dans leurs descriptor.json pour l'implémentions que l'on a faite de notre PedalBoard. Les clés module et img sont respectivement le module importé précédemment depuis les différents fichiers index.js et l'élément HTML contenant la miniature du plugin.

Algorithme 3 Contenu de PedalBoardPlugin.WAMS

```
{
  "Faust PingPongDelay": {
    "url": "https://mainline.i3s.unice.fr/wam2/packages/faustPingPongDelay
        /plugin/",
    "descriptor": {
      "name": "Faust PingPongDelay",
      "vendor": "Shihong Ren",
      "description": "A PingPongDelay written in Faust with its default UI
          ",
      "version": "1.0.0",
      "apiVersion": "2.0.0",
      "thumbnail": "screenshot.png",
      "keywords": [
        "delay",
        "faust"
      ],
      "isInstrument": false,
      "website": ""
    },
    "module": {},
    "img": {}
  },
  ...
}
```

A partir de la Gui, nous pouvons initialiser la partie preview du PedalBoard. Il suffit alors de cliquer sur la miniature d'un plugin pour l'ajouter à la fin de la chaîne audio de notre PedalBoard.

6.3 Le board

Tandis que la partie preview s'apparenterait à un carton ou l'on rangerait ses différentes pédales d'effets entre chaque utilisation, le board lui correspond à la planche de bois des PedalBoard physiques ou l'on dispose en chaîne chaque pédale pour affecter le signal audio.

C'est donc depuis celui ci que l'on peut modifier les boutons des différents plugins une fois qu'ils ont été initialisés en cliquant sur leurs miniatures. Dans notre cas le son se déplace de gauche à droite et donc chaque plugin aura en entrée le son modifié après le passage dans son voisin de gauche.

FIGURE 3 – Board avec des plugins chargés

Le board permet aussi de supprimer les plugins de celui-ci grâce à la croix à coté de leurs noms et de changer leur ordre dans la chaîne audio grâce à l'API Glisser-déposer même si celle-ci est différente selon les navigateurs et nous a donc posé quelques problèmes de comptabilité.

Pour réaliser ces différentes actions il faut garder en mémoire une liste qui incorpore l'ordre des différents plugins. Pour cela nous avons choisi l'approche qui nous paraissait la plus intuitive, c'est a dire la HTMLCollection des fils du board. En effet, il est très simple de supprimer ou de modifier l'ordre des fils d'un élément HTML depuis JavaScript. Les fonctions de base de JavaScript nous évitent d'avoir a gérer ces problèmes nous même ce qui impliquerait l'utilisation d'une liste et cela rendrais l'opération beaucoup plus complexe.

En revanche il faut modifier la chaîne audio nous même, pour cela nous avons choisi la facilité, nous déconnectons tous les plugins puis on réalise l'action, donc le changement d'ordre ou la suppression d'un plugin, puis nous les reconnectons tous selon l'ordre dans la HTMLCollection.

Chaque Gui possède un id unique donné à sa création ce qui nous permet de pouvoir récupérer son audioNode correspondante, les différentes audioNodes sont stockées dans un objet de la classe PedalBoardNode nommé nodes pour pouvoir garder en mémoire leurs paramètres, les clés de celui ci étant les id des Gui.

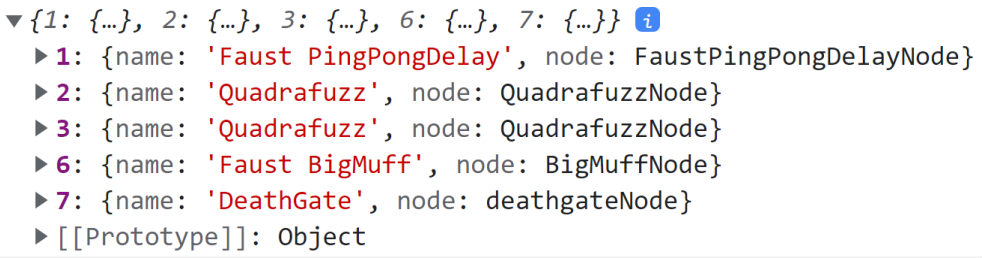

FIGURE 4 – L'objet nodes pour le board de la Figure 3

A chaque changement sur la chaîne audio il faut lancer un évènement qui sera écouté par l'host pour l'informer d'une évolution du nombre de paramètres du plugin. Celui-ci s'appelle « wam-info ». Cet évènement est principalement utilisé pour l'automation que nous allons introduire dans la subsection 6.6.

Nous l'avons déjà évoqué précédemment mais la gui de chaque plugin est sous la forme d'un Web Component et cela implique que l'on ne peut pas agir sur son css car on ne connait pas la structure de celui-ci.

Malheureusement tous les plugins ne font pas la même taille et il a donc fallu les redimensionner pour notre PedalBoard en leur appliquant un effet de changement d'échelle (scale en css) ce qui n'a pas été si simple que cela.

6.4 Les presets

L'objectif des presets est de pouvoir sauvegarder la liste des plugins qui sont sur le board. Ils permettent également de sauvegarder les réglages comme l'ordre des plugins sur le board et leurs boutons afin de pouvoir réutiliser cette configuration plus rapidement à l'avenir.

Les presets sont sauvegardés dans un système de dossier. En effet, chaque preset est sauvegardé sous une catégorie appelée banks et il n'y a pas de limite de presets sous chaque banks. Nous avons intégré des presets de base qui se chargent à l'initialisation du PedalBoard, ceux ci se trouvent dans le fichier initialState.json, si le PedalBoard est initialisé sans un état de départ alors ce fichier sera chargé à la place.

Il est bien évidement possible de créer, supprimer et renommer un preset ou une banks en revanche chaques presets sous la même banks doivent avoir un nom différent car ceux ci servent de clés dans la banks. Pour charger un preset, il suffit de cliquer dessus.

FIGURE 5 – Partie presets du PedalBoard

6.5 La sauvegarde et l'initialisation du plugin

Nous avons aussi implémenté les méthodes de la classe WamNode du SDK saveState et loadState, qui permettent respectivement de sauvegarder l'état courant du PedalBoard dans un objet et de recharger le PedalBoard depuis un état précédemment sauvegarder.

Ces méthodes ont été facilitées car il nous suffit d'appeler les mêmes méthodes sur les nodes de notre board car se sont aussi des WamNode, l'état correspond à un objet avec deux clés principales. Current qui contient les états des plugins actuellement sur le board et la partie presets que l'on a évoqué dans la subsection 6.4.

Algorithme 4 Exemple d'état du PedalBoard sauvegardé

```
{
  "current": [
    {
      "name": "Quadrafuzz",
      "state": {
        "midHighGain": 0.5,
        "enabled": 1,
        "midLowGain": 0.800000011920929,
        "lowGain": 0.6000000238418579,
        "highGain": 0.5
      }
    },
    {
      "name": "Faust BigMuff",
      "state": {
        "/BigMuff/Output": 100,
        "/BigMuff/Input": 0,
        "/BigMuff/Drive": 1,
        "/BigMuff/bypass": 0,
        "/BigMuff/Tone": 0.5
      }
    }
  ],
  "presets": {
    ...
  }
}
```

6.6 Automation

L'automation est le fait de créer des événements qui vont modifier les valeurs des boutons d'un plugin au fil du temps selon une courbe définie par l'utilisateur.

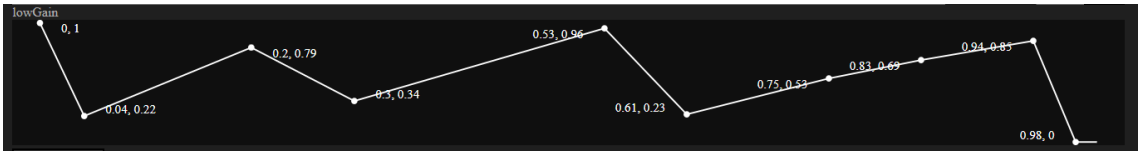

FIGURE 6 – Exemple d'une courbe d'automation

Mais avant d'expliquer comment nous avons réalisé l'automation de notre PedalBoard nous

devons vous expliquer comment un bouton peut influer sur la sortie audio.

Tous les plugins implémentés depuis le SDK et le SDK-Parammgr (abréviation de gestionnaires des paramètres du sdk) de WAM2, ont une série de paramètres qui permettent de modifier les valeurs des AudioNodes qu'ils contiennent.

Ceux-ci sont soit reliés à une seule AudioNode soit associé à plusieurs AudioNodes ce qui permet à un seul bouton de modifier plusieurs AudioNodes en même temps. C'est la classe ParamMgrNode qui permet de le faire grâce à trois objets passés à l'initialisation de celle ci :

— **paramsConfig** : Les paramètres visibles depuis l'extérieur du plugin, ils correspondent aux boutons de la gui.

— **internalParamsConfig** : Ces paramètres sont les AudioNodes internes du plugins, non accessibles depuis l'extérieur.

— **paramsMapping** : Fait l'association entre les paramètres externes et internes.

Algorithme 5 Exemple d'initialisation d'une ParamMgrNode.

```
1         const paramsConfig = {
2             mix: {
3                 defaultValue: 0.5,
4                 minValue: 0,
5                 maxValue: 1
6             }
7         }
8         const internalParamsConfig = {
9             dryGain: dryGainNode.gain,
10            wetGain: wetGainNode.gain,
11        };
12        const paramsMapping = {
13            mix: {
14                dryGain: {
15                    sourceRange: [0.5, 1],
16                    targetRange: [1, 0],
17                },
18                wetGain: {
19                    sourceRange: [0, 0.5],
20                    targetRange: [0, 1],
21                },
22            },
23        };
24        const option = {
25            paramsConfig,
26            internalParamsConfig,
27            paramsMapping
28        };
29        const paramMgr = await ParamMgrFactory.create(wam, option);
```

Dans cet exemple on peut voir que le paramètre mix est associé à deux AudioNodes différentes, dryGain et wetGain, donc lorsque l'on va modifier son bouton, les deux AudioNodes seront modifiées à leurs tours.‘

Passer des associations de paramètres à l'initialisation fait la force de la classe ParamMgrNode mais c'est aussi un handicap. En effet, notre PedalBoard est dynamique, c'est-à-dire que le nombre de paramètres qu'il expose varie à chaque instant. L'utilisateur peut ajouter et supprimer

des plugins du Board, ce qui implique que le nombre de ceux-ci change au fil du temps et donc leurs paramètres aussi.

Pour résoudre ce problème, nous avons dû rechercher dans le SDK les méthodes qui étaient utilisées lorsque nous voulons connaître les paramètres externes d'un plugin et leurs valeurs. Ces méthodes sont getParameterInfo qui renvoie un objet avec le nom des boutons en tant que clés et leurs informations en tant que valeurs, et getParameterValues qui renvoie la valeur d'un bouton au moment de l'appel de la méthode.

Algorithme 6 Exemple de résultat de la méthode getParameterInfo.

```
{
   "Mix":{
      "id":0,
      "defaultValue":1,
      "label":"Mix",
      "maxValue":1,
      "minValue":0,
      "type":"float"
   },
   ...
}
```

Algorithme 7 Exemple de résultat de la méthode getParameterValues pour le paramètre Mix.

```
{
   "Mix":{
      "id":"0",
      "normalized":"undefined",
      "value":0.6000000238418579
   }
}
```

Nous n'allons pas rentrer dans les détails du code mais pour resumer, notre implementation est la suivante :

Dans getParameterInfo, pour chaque plugin actuellement sur le Board, nous appelons sa méthode getParameterInfo ce qui nous permet de réunir ses informations dans un objet qui a pour clé un identifiant unique du plugin suivi de son nom. Cela évite les conflits entre deux plugins du même nom.

En ce qui concerne getParameterValues nous utilisons le même procédé, il suffit d'appeler la méthode selon l'id du plugin passé en paramètre.

Nous sommes à présent capable de partager les informations de nos plugins à l'extérieur mais il reste encore un problème, comment mettre à jour les valeurs de ceux-ci ?

Après de longues recherches dans le SDK nous avons trouvé la méthode scheduleEvents qui permet de modifier les valeurs des paramètres à un instant "t". Elle agit donc en tant que générateur d'événements. C'est donc cette méthode qui nous permet de réaliser l'automation des boutons de notre PedalBoard. Dans notre cas, tout comme pour getParameterInfo et getParameterValues, il nous suffit de propager la méthode aux plugins correspondant à l'id passé en paramètre.

6.7 Utilisation de shaders avec BabylonJS

En parallèle de notre TER nous avons tous les quatre suivi le cours de Mr Buffa intitulé PROGRAMMING 3D GAMES ON THE WEB durant lequel nous avons appris à nous servir de BabylonJS qui est un moteur 3D développé en JavaScript pour le web.

Au milieu de notre TER Mr Buffa nous a montré une démonstration du travail d'un de ses collègues qui était entrain d'implémenter un shader dans plugin et celui ci variait selon les paramètres du plugin. Un shader est un bout de code qui communique directement avec le GPU pour réaliser des rendus graphiques très rapides et performants, dans notre cas il est écrit en GLSL.

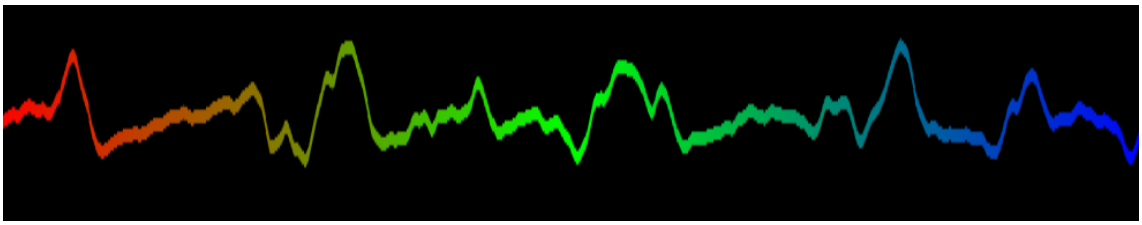

FIGURE 7 – Le shader de notre PedalBoard

A l'aide d'une AnalyserNode positionnée en fin de chaîne de notre PedalBoard, on peut récupérer la forme de l'onde de notre audio en sortie. Malheureusement on ne peut pas directement l'afficher dans un shader car le type de celle ci n'est pas reconnu, il faut d'abord la transformer en tant que texture puis on peut l'envoyer au shader sous la forme d'un Sample2D. Le shader est ensuite affiché grâce au Post Process de BabylonJS, celui ci agit comme un filtre sur le canvas ce qui permet de garder les bonnes proportions lorsque l'on redimensionne le PedalBoard.

7 La collaboration avec le TER n°13

En parallèle de notre TER, Mr Buffa encadrait un autre groupe dont le sujet était « Cross compilation C++/Rust/Web assembly, écriture d'un logiciel hôte pour des plugins WebAudio écrits en WebAssembly » et testable sur ce lien.

Leur projet correspond à un gestionnaire multipistes audio et ils voulaient intégrer notre PedalBoard pour pouvoir donner des effets à chacune de leurs pistes, et c'est très simple à faire car le PedalBoard est lui même un plugin comme expliqué précédemment donc ils ont pu facilement l'importer sur leur projet.

En revanche il a fallu modifier un peu la Gui pour que celle ci aille avec leur style graphique et cette version est disponible sur la branche TER13-custom-pedalboard de notre projet.

FIGURE 8 – Notre PedalBoard intégré sur leur projet

8 Le serveur

Très vite, nous avons eu besoin de créer nos propres plugins pour pouvoir les utiliser dans notre PedalBoard. C'est pour cela que nous avons créé un serveur qui nous permet d'host le PedalBoard et nos propres plugins en ligne. Cela permet aussi au PedalBoard d'être directement importable depuis une url de la même manière que pour les plugins qu'il contient.

Le serveur est très simple et permet d'accéder à chaque plugins individuellement grâce a leurs chemins relatifs. De plus, tout comme le PedalBoard, le serveur dispose d'un fichier regroupant ces chemins nommé plugins.json.

A chaque ajout de plugin sur le serveur, il faut rajouter son chemin relatif dans ce fichier pour que celui-ci soit pris en compte. L'ajout d'un plugin au serveur est détaillé dans le point 5 de la section 9.

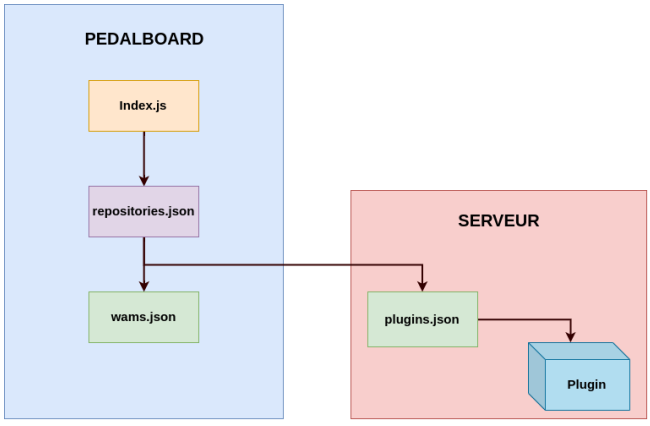

FIGURE 9 – Organisation du serveur

9 Création d'un plugin

Dans cette partie nous allons vous expliquer pas à pas comment créer puis ajouter un plugin au PedalBoard depuis Faust IDE :

1. Premièrement, il vous faut disposer d'un code DSP.
2. Vous pouvez alors l'ajouter sur Faust IDE et le compiler avec l'option WAP GUI Builder.
3. Dans la partie GUI Builder il est alors possible de définir le style des boutons et des labels. La personnalisation n'etant pas assez complète à notre goût donc nous avons réalisé un outil pour nous permettre de corriger ses défauts.
4. Une fois que le résultat vous convient vous pouvez cliquer sur PUBLISH / PREVIEW PLUGIN WAM2 puis Download Zip.
5. Il ne vous reste plus qu'à dézipper le plugin puis à le déplacer dans le dossier plugin du serveur. Il faut toutefois penser à donner un nom et une miniature au plugin dans son fichier descriptor.json pour ne pas causer de problème à l'affichage. Il faut également ajouter le chemin relatif du plugin dans le fichier plugins.json du serveur pour qu'il puisse être récupéré par le PedalBoard.

FIGURE 10 – Exemple du Plugin SweetWah

Au total nous avons du recréer la GUI de plus d'une dizaine de plugins. Ils sont tous accessibles dans le dossier plugin de notre serveur.

À force d'utiliser le GUI Builder de FaustIDE nous en sommes venus à la conclusion qu'il serait peut-être pertinent de créer une nouvelle version de celui-ci, peut être même dans le cadre d'un futur TER, et qui s'inspirerait de ce qu'a fait BabylonJS pour sa GUI.

C'est-à-dire un système de grilles, les boutons disposés proportionnellement à leurs parents, plus d'options de customisation et surtout une interface plus ergonomique.

Quatrième partie
Gestion du projet

10 Notre organisation

Pour l'organisation, nous avons déployé deux dépôts github nous permettant de suivre les modifications et de se partager le code, le premier était exclusivement pour le PedalBoard et le deuxième comprend le PedalBoard, le serveur et les différents plugins que nous avons créé.

Michael s'est chargé de quelques plugins, il a participé à la preview et aux presets du pedalboard.

Pierre s'est chargé de mettre en place le serveur de plugin, a participé au Glisser-déposer et il a aussi fait quelques plugins.

Yann a fait le système de presets du pedalboard, a participé à la stylisation et a programmé quelques plugins.

Quentin quant à lui a touché à tout, les plugins, le serveur, la sauvegarde, l'interface et l'automation.

11 Les difficultés rencontrées

Vous trouverez ci dessous quelques exemples des difficultés rencontrées lors du développement :

— Les plugins que nous importons n'avaient pas la même taille. Il a donc fallu les redimensionner grâce au css et ça a été une grande source de problèmes.

— L'API Glisser-déposer de JavaScript a une implémentation différente selon les navigateurs internet. Nous avons donc dû refaire ce que nous avions déjà fait plusieurs fois pour être sûrs que cela marche sur tous les principaux navigateurs.

— Pour l'automation, il a fallu fouiller dans le SDK manuellement pour comprendre comment l'implémenter dans notre projet. En effet, le seul exemple de plugin regroupant d'autre plugin que nous avions était une version précédente du PedalBoard faite en React par un stagiaire de Mr Buffa. Or, celui-ci n'arrivant pas à faire marcher l'automation non plus, il a préféré modifier le SDK pour ses besoins.

  Vu que nous ne voulions pas suivre cette voie qui empêche la mise à jour du SDK, nous avons dû nous débrouiller par nous même.

— Pour ajouter un shader dans notre projet, nous avons aussi eu beaucoup de mal car premièrement nous n'avions jamais codé de shader. Ensuite, pour afficher une courbe audio il a fallut la transformer en texture et ce processus n'était pas très bien documenté dans BabylonJS.

12 Idées abandonnés

Au départ nous avions dans l'idée de refaire la Gui du PedalBoard, qui est entièrement en JavaScript, en React pour faciliter sa compréhension une fois le PedalBoard terminé. Cependant, avec les nombreux plugins qu'il nous restait à implémenter et la fin du projet qui approchait nous avons préféré abandonner cette idée qui au final n'apportait pas grand chose de plus pour notre TER.

Cinquième partie
Conclusion

Ce projet était très long et très complet mais nous a beaucoup intéressé. La possibilité de réaliser un projet grâce à un SDK développé par un de nos professeurs et son équipe de professionnels a été très enrichissant et motivant.

Grâce à cela, nous avons acquis de solides bases pour la manipulation de l'audio sur le web et nous avons pu solidifier nos connaissances de l'HTML et du JavaScript.

Même si brève, l'utilisation que l'on a fait des shaders et l'infini possibilité qu'ils offrent est quelque chose qui nous a beaucoup impressionné et nous a donné envie d'en faire nous même dans le futur.

Pour ce qui est du sujet en lui même, nous pensons avoir rempli toutes les cases et nous espérons que le rendu final saura entièrement satisfaire notre encadrant Mr Michel Buffa.

Sixième partie
Perspectives et réflexions personnelles

Pour la suite, nous allons présenter notre pedalboard à la Web Audio Conférence qui se tiendra au Palais des Festivals à Cannes du 6 au 8 juillet, nous serons accompagnés des membres du TER 13 et de notre professeur Mr Michel Buffa.

Bibliographie

Buffa, M., Demetrio, M. et Azria, N. (2016). Guitar pedal board using WebAudio. *In Proceedings of the Web Audio Conference*, Atlanta, United States. Georgia tech Institute, Atlanta.

Buffa, M., Ren, S., Campbell, O., Burns, T., Yi, S., Kleimola, J. et Larkin, O. (2022). Web audio modules 2.0 : an open web audio plugin standard.

Kleimola, J. et Larkin, O. (2015). Web audio modules. *In Proceedings of the Sound and Music Computing Conference*.