



École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5^e année
2012 - 2013

Rapport de Projet de Fin d'Etudes

Solution de tests pour l'étude des battements binauraux

Encadrant

Pascal MAKRIS
pascal.makris@univ-tours.fr

Université François-Rabelais, Tours

Étudiant

François DENNIG
francois.dennig@etu.univ-tours.fr

DI5 2012 - 2013

Version du 30 avril 2013

Table des matières

1	Introduction	9
2	Présentation du projet	10
2.1	Contexte et motivations	10
2.1.1	Notion de battement	10
2.1.2	Notion de battements binauraux	11
2.1.3	Hypothèse	12
2.1.4	Motivations	13
2.2	Objectifs fixés	13
2.3	Contexte d'utilisation de la solution	14
2.4	Matériel EEG utilisé	14
2.4.1	Système ActiveTwo	15
3	Méthode de travail	16
3.1	Gestion de projet	16
3.1.1	Outils de gestion de projet	17
3.2	Développement	17
3.2.1	Bibliothèques	18
3.3	Rédaction de compte-rendus	18
4	Solution apportée	19
4.1	Présentation du logiciel final	19
4.1.1	Préalable : définition de la notion de scénario	19
4.1.2	Déroulement d'une séance de test avec le logiciel	21
4.1.3	La fenêtre principale	22
4.1.4	L'onglet de configuration	23
4.1.5	L'onglet de description de scénario	24
4.1.6	L'onglet d'édition de scénario	25
4.1.7	L'onglet de contrôle de scénario	26
4.1.8	L'onglet d'analyse	27
4.1.9	La fenêtre de visualisation des signaux	28
4.1.10	La fenêtre d'analyse	29
4.2	Modélisation	31
4.2.1	Architecture générale	31
4.2.2	Modélisation des classes	33
4.3	Quelques détails sur le développement, difficultés	44
4.3.1	Configuration	44
4.3.2	Module d'édition de scénario	47
4.3.3	Module audio, lecture de scénario	50
4.3.4	Importation de fichiers BDF	52
4.3.5	Pré-traitement	53
4.3.6	Analyse temps-fréquence	62



4.3.7	Gestion de la mémoire vive	69
4.4	Tests réalisés	71
4.4.1	Tests unitaires	71
4.4.2	Tests de validation fonctionnelle	78
4.4.3	Test de validation solution	78
4.4.4	Tests de validation robustesse	79
5	Bilan sur le planning	80
5.1	Planning prévisionnel	80
5.1.1	Sprint 1	80
5.1.2	Sprint 2	80
5.1.3	Sprint 3	80
5.1.4	Mise en production	80
5.2	Comparaison avec le planning effectif	82
6	Evolutions	84
6.1	Améliorations possibles	84
6.1.1	Stimulations auditives multiples	84
6.1.2	Paramétrage de l'analyse	85
6.1.3	Edition graphique pour tous les paramètres	85
6.1.4	Acquisition directe	85
6.2	Nouvelles perspectives pour le projet	85
7	Conclusion	86
A	Références	87
B	Manuel utilisateur	88
B.1	Installation	88
B.2	Lancement pour la première fois	88
B.2.1	Test et choix du périphérique audio	88
B.2.2	Calibration du volume sonore	88
B.3	Création d'un scénario	89
B.3.1	Description d'un scénario	89
B.3.2	Edition d'un scénario	89
B.4	Chargement/Ouverture d'un scénario	90
B.5	Démarrage du scénario	90
B.5.1	Activation du triggering	90
B.5.2	Démarrage	90
B.6	Importation BDF et pré-traitement	90
B.6.1	Configuration des canaux de pré-traitement	90
B.6.2	Pré-traitement	90
B.7	Visualisation des signaux	90
B.7.1	Changement du canal	91
B.7.2	Zoom	91
B.8	Analyse temps-fréquence	91
B.8.1	Configuration de l'analyse	91
B.8.2	Démarrage de l'analyse	91
B.8.3	Interprétation de l'analyse : échelles de couleur, zoom	91
B.8.4	Exportation des résultats	91

C Cahier de Spécifications

92

Table des figures

2.1	Création d'un battement à partir d'ondes sinusoïdales	11
2.2	La formation réticulée au sein du cerveau	12
2.3	Plan de la pièce d'expérimentations. 1) : local de préparation. 2) : zone de test. 3) : zone de contrôle. La ligne traversant le mur entre la zone 2 et la zone 3 est un passage de câbles pour relier l'appareil d'EEG aux ordinateurs.	14
3.1	Cycle de vie des sprints	16
4.1	Structure générale du fichier XML du scénario de l'exemple précédent	19
4.2	Structure XML de l'étape 1 du scénario de l'exemple précédent	20
4.3	Evolution de la fréquence des ondes de chaque canal dans l'exemple de scénario précédent	20
4.4	Fenêtre principale de l'application. Ici l'onglet de contrôle est inaccessible, car aucun scénario n'a été créé ou chargé.	22
4.5	L'onglet de configuration	23
4.6	L'onglet de description de scénario	24
4.7	L'onglet d'édition de scénario, avec la boîte de dialogue d'édition d'étape	25
4.8	L'Onglet de contrôle	26
4.9	L'onglet d'analyse.	27
4.10	Boîte de dialogue de sélection des canaux de pré-traitement	27
4.11	Fenêtre de visualisation des signaux.	28
4.12	Fenêtre de visualisation des signaux, avec un zoom.	28
4.13	Choix du signal à analyser	29
4.14	Représentation de l'analyse temps-fréquence	30
4.15	Représentation de l'analyse temps-fréquence, avec de nouveaux paramètres d'échelle de couleur	30
4.16	Architecture générale de l'application	31
4.17	Interactions entre couches	32
4.18	Diagramme de classes de Scenario et Step	34
4.19	Diagramme de classe de DataSignal	35
4.20	Diagramme de classe de TriggManager	36
4.21	Diagramme de classes de Analyzer	37
4.22	Diagramme de classes de SoundManager et ScenarioPlayer	38
4.23	Diagramme de classes de ScenarioParser	39
4.24	Diagramme de classes de ScenarioDao	40
4.25	Diagramme de classes de BiosemiFileManager	40
4.26	Aperçu des paramètres de l'application dans la base de registres de Windows	46
4.27	Diagramme de communication pour le déplacement d'un point d'une courbe de la représentation graphique de scénario	49
4.28	Les deux sinusoïdes superposées, le signal composé des deux sinusoïdes, puis le résultat du filtrage.	56
4.29	Filtre passe-haut parfait	57
4.30	Comparaison de la réponse en amplitude du filtre de Chebyshev de type 1 avec celui de type2	57

4.31 Réponse en amplitude du filtre Butterworth (ordre 1)	58
4.32 Différences entre filtrage classique et filtrage zero-phase	59
4.33 Illustration du risque du filtrage zero-phase pour une fréquence de coupure trop basse	61
4.34 Résolution de la transformée de Fourier. Les fréquences y sont parfaitement restituées, mais la notion de temps est inexistante.	64
4.35 Problème du paramétrage de la taille de la fenêtre pour STFT (1). STFT avec une très grande largeur de fenêtre. On peut avoir une excellente résolution fréquentielle, mais on ne saura pas quand telle fréquence est présente, juste si elle se trouve dans la 1ère ou la 2ème moitié du signal...	65
4.36 Problème de paramétrage de la taille de la fenêtre pour STFT (2). Ici, la largeur de la fenêtre est plus courte. Le temps y est donc mieux représenté. Cependant, il existe maintenant une plus grande incertitude dans la fréquence.	65
4.37 Illustration du principe de multirésolution	66
4.38 Diagramme de classes de CwtLib++	67
4.39 Exemple de spectrogramme réalisé avec Qt, CwtLib++ et Qwt, pour un signal non-stationnaire avec une première portion à 128 Hz puis une deuxième à 30 Hz. En haut : la représentation du signal. En bas : son spectrogramme.	68
4.40 Test Filtrage 1 : signal initial	72
4.41 Test Filtrage 1 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.	72
4.42 Test Filtrage 2 : signal initial	73
4.43 Test Filtrage 2 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.	73
4.44 Test Filtrage 3 : signal initial	74
4.45 Test Filtrage 3 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.	74
4.46 Test Transformée en Ondelettes 1	75
4.47 Test Transformée en Ondelettes 2	76
4.48 Test Transformée en Ondelettes 3	77
5.1 Planning prévisionnel par sprint	81
5.2 Planning effectif	82
5.3 Tableau d'évaluation du Sprint 1	83

Liste des tableaux

4.1	Test Filtrage 1	72
4.2	Test Filtrage 2	73
4.3	Test Filtrage 2	74
4.4	Test Transformée en Ondelettes 1	75
4.5	Test Transformée en Ondelettes 2	76
4.6	Test Transformée en Ondelettes 3	77

Introduction

Ce rapport s'inscrit dans le cadre des Projets de Fin d'Etudes qui se déroulent au cours de la dernière année d'études à Polytech Tours, au département informatique. Chaque projet est réalisé par un seul étudiant, afin de mesurer ses compétences tout au long de la mise en oeuvre : gestion de projet, spécifications, développement, tests, déploiement.

Le projet en question consiste en la réalisation d'un logiciel permettant de tester les effets des battements binauraux sur le cerveau humain. Il est encadré par Pascal Makris, et assisté par trois membres de l'équipe Autisme de l'UMR INSERM U930 (Université François Rabelais, Centre Universitaire de Pédopsychiatrie, CHRU Bretonneau) : Marie Gomot, Nicole Bruneau (CR1 INSERM) et Sylvie Roux (IR).

L'idée de ce projet provient de la conjointe réflexion entre ces trois personnes, Pascal Makris et moi-même. En effet, c'est de ma propre initiative que je suis allé consulter ces personnes pour faire émerger ce sujet. C'est pourquoi je les remercie de m'avoir si gentiment accueilli dans leurs locaux et d'avoir prêté attention à mes propositions. Ce projet n'aurait pu voir le jour sans le temps qu'elles m'ont accordé.

Présentation du projet

2.1 Contexte et motivations

Cette partie a pour but d'expliquer les raisons ayant poussé à mettre en oeuvre ce projet, et en pose le contexte. Mais d'abord, il nous faut expliquer la notion de battements binauraux.

2.1.1 Notion de battement

Avant de savoir ce que sont les battements binauraux, il nous faut déjà être clairs sur la définition d'un battement.

Le terme de battement est utilisé en acoustique pour désigner le phénomène créé par la rencontre de deux sons dont la fréquence est légèrement différente. Pour donner un exemple, deux cordes de guitare accordées normalement sur la même note, mais pas tout à fait, ne vont pas sonner à l'unison mais vont donner un tremolo lorsqu'elles seront grattées en même temps. Cela est dû au fait que les ondes des deux notes se mélangent.

En effet, lorsque l'on mélange deux ondes, celles-ci forment une onde dont la formule mathématique est la somme des deux premières.

Ainsi, si on considère deux ondes de forme sinusoïdale, de fréquences respectives f_1 et f_2 , et de même amplitude, le mélange de ces deux ondes sera une onde dont la formule est équivalente au produit entre :

- une sinusoïde de fréquence égale à la moyenne entre f_1 et f_2 , soit $(f_1 + f_2)/2$
- et une sinusoïde de fréquence égale à la demi-différence de f_1 et f_2 , soit $(f_1 - f_2)/2$

L'onde résultante étant une modulation d'amplitude de la première sinusoïde ci-dessus, par la deuxième, la fréquence du battement (dans ce cas le battement est appelé "enveloppe") est égal à $(f_1 - f_2)/2$.

Pourtant, l'oreille humaine perçoit ce genre de battement comme ayant une fréquence égale à $f_1 - f_2$. Cela est dû à notre oreille qui n'est pas capable de percevoir le léger décalage temporel entre deux demi-périodes de ce genre de signal.

Ainsi, si on mélange deux sons sinusoïdaux, l'un de fréquence 400 Hz, l'autre de fréquence 410 Hz, nous allons percevoir un son sinusoïdal de fréquence 405 Hz, avec un battement de 10 Hz (10 pulsations par seconde).

La figure suivante illustre le phénomène avec le même exemple énoncé précédemment.

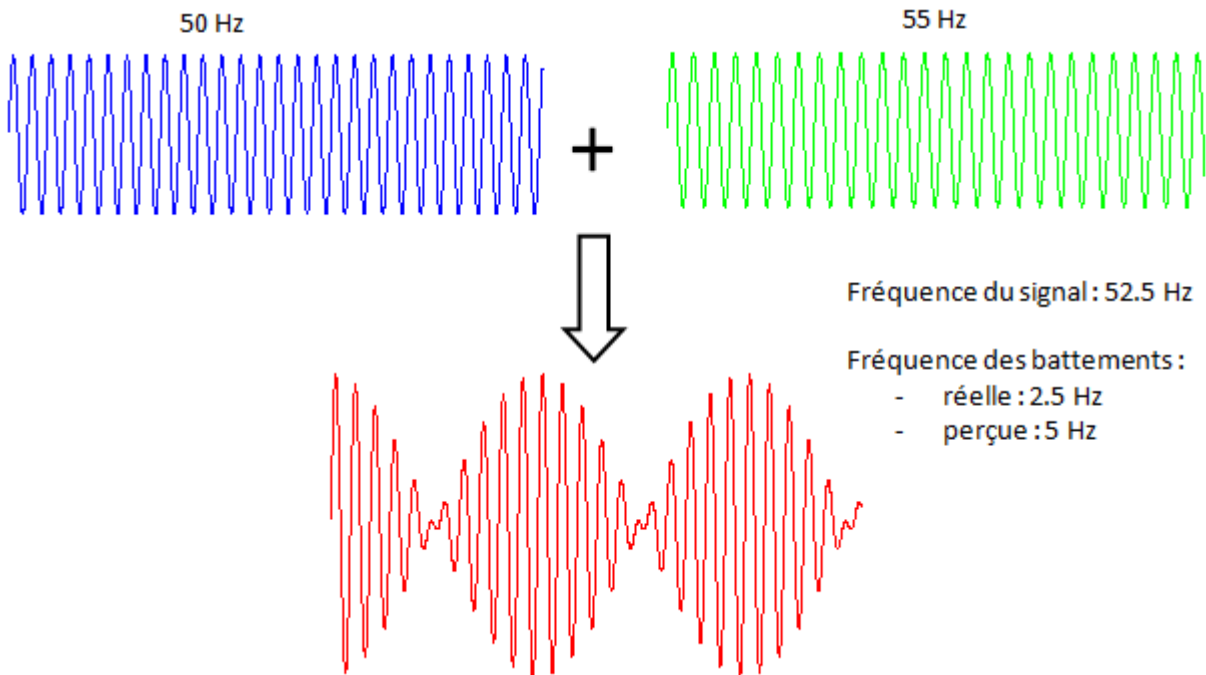
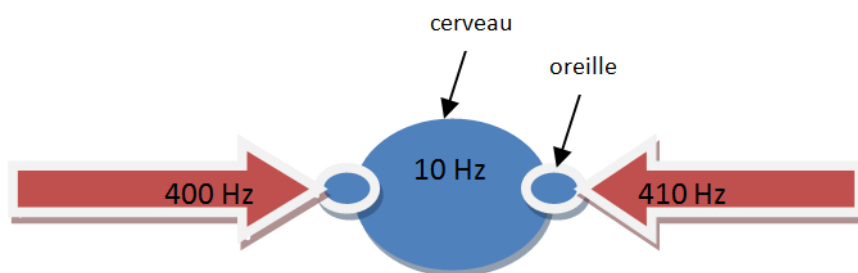


FIGURE 2.1 – Création d'un battement à partir d'ondes sinusoïdales

2.1.2 Notion de battements binauraux

Imaginons maintenant que l'on ne mélange pas les deux ondes. On attribue chaque onde à une oreille différente. Lorsque l'on écoute uniquement le son arrivant dans notre oreille gauche, alors on entend le son de fréquence 400 Hz. Si on écoute uniquement le son arrivant dans notre oreille droite, on entend le son de fréquence 410 Hz. Mais si on écoute le son de chaque oreille, simultanément avec un casque stéréo, alors on entend le phénomène de battement. Pourtant les deux ondes ne sont pas mélangées (du moins à l'extérieur de notre tête). Dans ce cas, on qualifie ce phénomène de **battements binauraux**.



2.1.3 Hypothèse

On sait que les battements binauraux sont détectés au niveau des noyaux olivaires supérieurs (centre de l'audition et de la localisation du son dans l'espace).

On sait aussi que le centre olivaire supérieur peut influencer la formation réticulée, qui possède plusieurs fonctions dont le contrôle cardiovasculaire, la modulation de la douleur, le sommeil, l'attention et l'habituation¹. Il est observé et communément admis que l'activité électrique des neurones est influencé par cette partie du cerveau, et que leur fréquence varie selon l'état de conscience (activité intense, concentration, somnolence...).

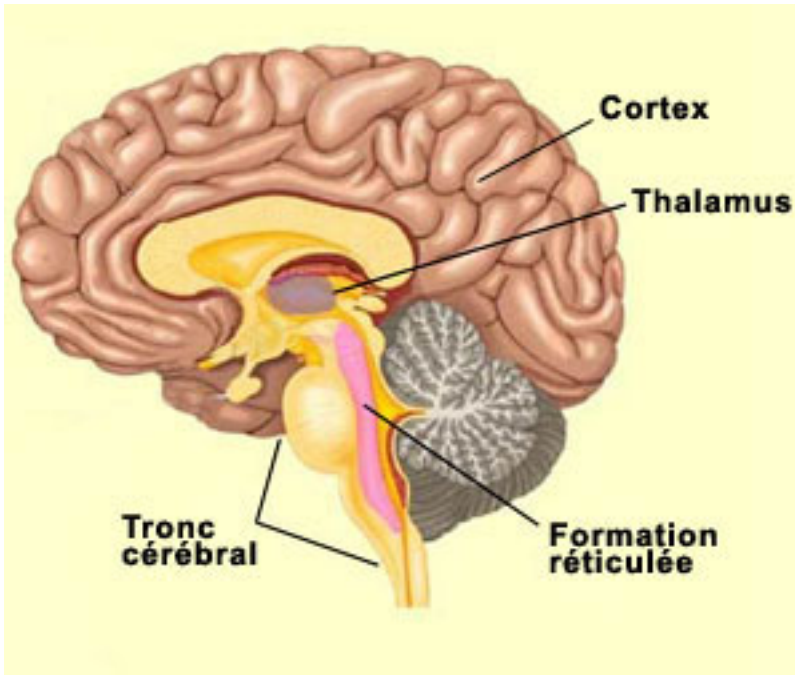


FIGURE 2.2 – La formation réticulée au sein du cerveau

L'hypothèse levée par la plupart des études au sujet des battements binauraux est que ces derniers, en passant dans la formation réticulée, modifient l'état de conscience de la personne. En effet, les battements binauraux ayant une fréquence dans la même plage de fréquences que les ondes cérébrales, pourraient pour ainsi dire "forcer" la formation réticulée à émettre des ondes à cette fréquence.

Cette théorie n'est pas clairement prouvée. En effet, les quelques études existant à ce sujet se contredisent. De plus, la plus grande partie (de celles qui prouvent la théorie) provient de l'Institut Monroe, qui a fait des battements binauraux une utilisation commerciale. Ainsi, peu d'institutions les utilisent. Pourtant, à en croire ce qui en est dit, ils pourraient permettre de traiter le stress, gérer la douleur, améliorer la mémoire, la créativité, faciliter l'attention, ou encore traiter les enfants présentant des problèmes développementaux.

1. L'habituation consiste à ignorer certains stimuli répétitifs insignifiants tout en restant sensibles à d'autres. On peut prendre l'exemple d'une personne qui arrive à s'endormir malgré les bruits incessants du trafic automobile, mais se fait réveiller par sa sonnerie de téléphone.

2.1.4 Motivations

Intéressé par le phénomène, j'ai proposé un sujet de PFE sur la réalisation d'un logiciel de conception et d'application des battements binauraux dans le domaine médical. Le projet aurait été axé sur la génération de paires d'ondes dont la somme donnerait des ondes à la forme très proche de celles émises par les neurones, afin d'optimiser la capacité de perception², puis sur la création de patches et scénarios, avec possibilité de gérer les traitements des patients sur une longue période. Le sujet a été accepté par le jury, mais a été modifié par la suite.

J'ai en effet pris contact, grâce à mon encadrant Pascal Makris, avec l'équipe Autisme de l'UMR INSERM U390 au CHRU Bretonneau de Tours. A l'issue de notre rencontre, il en a été dégagé qu'il était préférable de réaliser un logiciel permettant d'abord d'effectuer des tests sur les effets des battements binauraux sur l'activité électrique du cerveau. C'est donc avec ces personnes que nous avons décidé de mettre au point un logiciel permettant de réaliser des tests facilement. En effet, le processus habituel est assez lourd : il est nécessaire d'utiliser un premier logiciel pour émettre les stimuli, un deuxième pour capturer les signaux transmises par les électrodes, un troisième pour pré-traiter les données, un quatrième pour les visualiser, et un cinquième pour les analyser. Les battements binauraux n'étant pas dans l'axe de recherche de l'équipe de l'INSERM, il aurait été contraignant pour ses membres d'effectuer autant de manipulations pour ces tests. Nous nous sommes dits qu'il serait bien d'avoir un logiciel permettant de faire tout cela à la fois.

2.2 Objectifs fixés

Le projet tel qu'il a ainsi été fixé consiste à mettre en place une solution logicielle donnant la possibilité d'effectuer des tests qui, à long terme, devront permettre aux chercheurs d'établir les effets neurophysiologiques de l'écoute de battements binauraux sur l'humain. Les tests se baseront sur l'analyse temps-fréquence des ondes cérébrales, à mettre en parallèle avec les stimuli appliqués. Voici les principales fonctions du logiciel :

- paramétrage, planification et génération des signaux sonores à émettre
- envoi de signaux triggers (événements) au matériel EEG via le port parallèle
- importation de fichiers BDF contenant les données d'acquisition
- pré-traitement des données
- visualisation des signaux
- analyse des signaux par transformée en ondelettes
- exportation des données

2. SOURCE

2.3 Contexte d'utilisation de la solution

L'équipe de chercheurs de l'UMR INSERM pour qui le logiciel est réalisé travaille dans le bâtiment du département de pédopsychiatrie du CHRU de Bretonneau à Tours.

Une pièce est dédiée à la réalisation d'expérimentations scientifiques avec du matériel d'électroencéphalographie (EEG). Cette pièce, divisée en trois, comporte une partie isolée dans laquelle les sujets de test sont préparés (pose d'électrodes), une partie où le sujet est installé dans un fauteuil et enfermé, et une partie où les chercheurs contrôlent l'expérience de leurs ordinateurs.

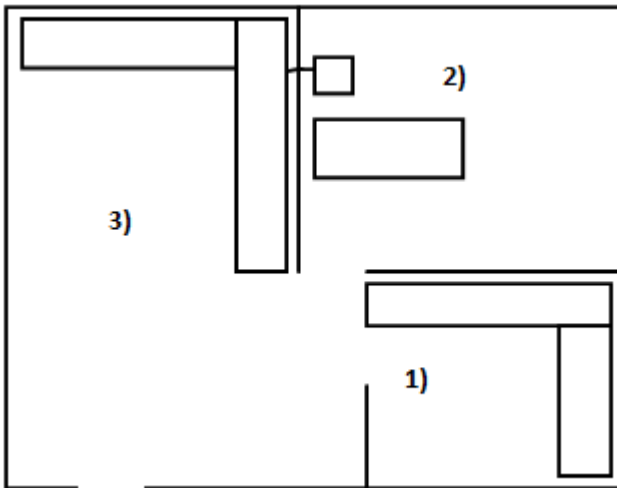


FIGURE 2.3 – Plan de la pièce d'expérimentations. 1) : local de préparation. 2) : zone de test. 3) : zone de contrôle. La ligne traversant le mur entre la zone 2 et la zone 3 est un passage de câbles pour relier l'appareil d'EEG aux ordinateurs.

Pour effectuer des expériences avec le logiciel réalisé à l'issue du projet, il faut au préalable que :

- le sujet de test se soit fait poser des électrodes (zone 1).
- le sujet de test se soit fait poser un casque audio stéréo, préférence tour de nuque pour ne pas interférer avec les électrodes
- le sujet de test soit assis et isolé dans la zone 2.
- le logiciel soit installé sur un des ordinateurs de la zone 3, auquel est relié le matériel de sortie audio où est branché le casque.

2.4 Matériel EEG utilisé

Pour effectuer l'acquisition des signaux électriques émis par le cerveau du sujet de test, les chercheurs utilisent le système Biosemi ActiveTwo.

Biosemi est un groupe créé par les ingénieurs en électronique Robert Honsbeek et Ton Kuipe et le physicien Coen Metting van Rijn en 1998. Leur but est de proposer une communauté scientifique avec un "état de l'art" de l'instrumentation en recherche électrophysiologique. Leurs produits, comprenant hardware entièrement configurable et software open source, sont faits pour être utilisés dans le cadre de la recherche. Ils ne sont pas destinés à une utilisation médicale.

2.4.1 Système ActiveTwo

Le système Biosemi utilisé par l'équipe de recherche est ActiveTwo. Successeur d'ActiveOne, il fournit des standards pour effectuer des mesures biopotentielles avec une haute résolution (24 bit) sur 280 canaux au maximum. Voici sa composition :

Des électrodes actives (pré-amplifiées) avec un casque dans lequel elles peuvent se clipser



ActiveTwo AD-Box : une boîte effectuant la conversion analogique-numérique avec une résolution de 24 bit, une fréquence d'échantillonnage maximale de 16 KHz (1.5Mo/seconde)



USB2 Receiver : une interface USB2 recevant les données de l'AD-Box par fibre optique et les envoyant au PC via USB2. Elle dispose également d'une entrée sur un port parallèle, permettant de recevoir des signaux triggers (événements).



ActiView : un logiciel basé sur LabView effectuant l'acquisition des données et leur enregistrement au format BDF.

Méthode de travail

3.1 Gestion de projet

Pour un projet ne comportant qu'une seule personne, il n'existe pas de méthode existante appropriée. La méthode utilisée est donc une méthode hybride, associant à la gestion de projet classique certaines spécificités des méthodes agiles. Voici les particularités de cette méthode :

- Le concept de cycle en V est conservé, mais on introduit plusieurs cycles dans le projet, appelés sprints. Chaque sprint comporte les étapes suivantes :
 - Développement
 - Tests
 - Déploiement
 - Recette

A l'issue de chaque sprint, une version (ne comportant que certaines fonctionnalités) doit être fournie et vérifiée. Les phases de spécifications, analyse et modélisation sont réalisées avant les sprints (on appelle aussi cette partie Sprint 0).

- On évalue d'abord ce qu'on appelle la complexité des tâches : il s'agit d'un chiffre arbitraire permettant de classer les tâches par ordre de difficulté. On calcule ensuite la complexité totale du projet en faisant la somme de toutes les complexités. Estimant que le projet est réalisable en temps imparti, on admet que la complexité totale correspond à la durée du projet. On calcule ainsi la charge de chaque tâche (en jours/homme... donc en jours) à partir de leur complexité en se basant sur sa proportion par rapport à la complexité totale.

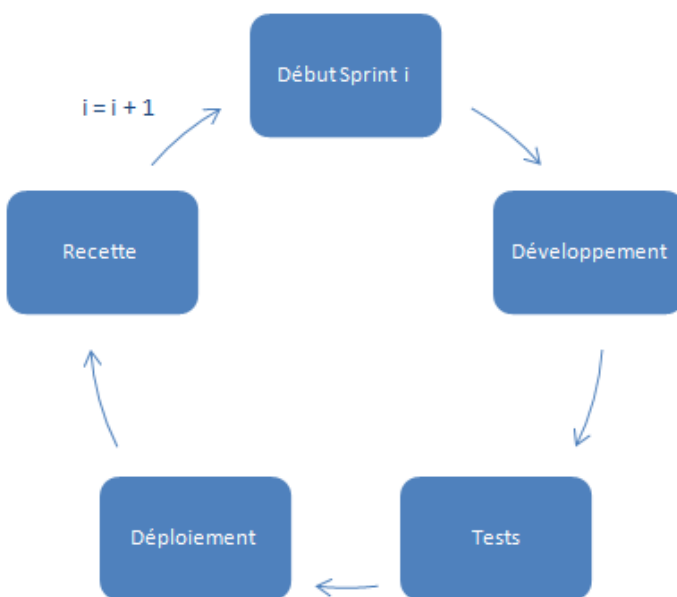


FIGURE 3.1 – Cycle de vie des sprints

3.1.1 Outils de gestion de projet

Côté gestion de projet, Microsoft Project 2010 a été utilisé pour la planification.

Pour suivre l'avancement du projet en temps réel, on se sert de l'application Trello, qui permet de se constituer un tableau de bord pour chaque sprint, contenant la liste des tâches à faire, en cours, en attente de tests, et terminées.

L'outil de versionning choisi est Redmine avec SVN. Le serveur Redmine se trouve à l'école. Cela permet aussi d'avoir une sauvegarde en cas de plantage de la machine de développement.

Les diagrammes (UML et autres) visibles dans ce rapport et dans le Cahier des Spécifications ont été réalisés avec le logiciel Microsoft Visio 2010.

3.2 Développement

Le logiciel est développé en C++ avec le framework Qt 4.8. L'IDE utilisé est Qt Creator. Le système d'exploitation pour le développement est Windows 8. Pour les tests, Windows XP est utilisé en machine virtuelle. Le déploiement est effectué sur un PC équipé de Windows XP.

La convention de nommage utilisée est la suivante :

- Les noms des classes commencent par une majuscule, et la suite est en style CamelCase. Exemple : UneClasse.
- Les noms des variables et des méthodes sont écrits en style CamelCase. Exemple : uneVariable, uneMethodeDeClasse.
- Les attributs de classes sont précédées du caractère `_`. Cela permet de savoir immédiatement, dans le corps d'une méthode, si on utilise un attribut de la classe, ou une autre variable, locale ou paramètre de méthode.
- Les variables de boucle commencent toutes par `i` et ne s'appellent pas `i`. On peut avoir par exemple : `iSignal`, `iVal`, `iListe...` Dans certains cas, on admet la notation `i`, `j`, lorsque le cas est trivial.

Le code est commenté selon plusieurs niveaux :

- Chaque classe est documentée sur plusieurs lignes avec la notation suivante :

```

/!*
 * Documentation
 * de classe
 */
class MaClasse...

```

- Chaque attribut et méthode est documenté

- soit sur une seule ligne, comme ceci :

```
int _monAttribut; /*!< Documentation d'attribut sur une ligne */
```

- soit sur plusieurs lignes, comme cela :

```

/!*
 * Documentation d'attribut
 * sur plusieurs lignes
 */
int _monAttribut;

```

- De grandes parties importantes de code ou de prototypes peuvent être précédées d'un commentaire de ce type :

```

//=====
// méthodes d'exemple
//=====
void exemple1(); /*!< exemple */

```

3.2.1 Bibliothèques

Plusieurs bibliothèques sont utilisées pour soulager la charge de travail. Qwt permet de constituer des représentations graphiques de manière scientifique. FMod Ex permet de gérer le son. EDFlib facilite la prise en charge des fichiers au format BDF (mais nous verrons que ses fonctions ne sont pas assez optimisées pour ce que l'on veut en faire). Cwtlib a servi de base pour la transformée en ondelettes.

Pour vérifier certains algorithmes, le logiciel scientifique d'analyse R est utilisé.

3.3 Rédaction de compte-rendus

Chaque semaine, un compte-rendu est rédigé et envoyé sur le SVN de l'école. Chaque compte-rendu résume ce qui a été fait lors de la ou les séance(s) de PFE de la semaine. Certains points peuvent être particulièrement détaillés, ce qui m'a été bénéfique lors de la rédaction du rapport.

Solution apportée

4.1 Présentation du logiciel final

4.1.1 Préalable : définition de la notion de scénario

Pour démontrer les effets des battements binauraux sur le cerveau, il est nécessaire de les faire écouter pendant un certain temps, et selon divers paramètres. Un scénario est une période de temps pendant laquelle les sons nécessaires à la perception des battements sont joués, tout en présentant une évolution de leurs paramètres au cours du temps. Il est composé de plusieurs étapes, chaque étape présentant des paramètres différents. Ces paramètres sont la fréquence des ondes de chaque canal, leur amplitude, l'amplitude du bruit, la durée. Exemple :

1. Etape 1 : sinusoïde de fréquence 400 Hz sur le canal gauche, sinusoïde de fréquence 410 Hz sur le canal droit, pendant 5 minutes.
2. Etape 2 : sinusoïde de fréquence passant linéairement de 400 Hz à 402 Hz sur le canal gauche, sinusoïde de fréquence passant linéairement de 410 Hz à 406 Hz sur le canal droit, pendant 5 minutes.
3. Etape 3 : sinusoïde de fréquence 402 Hz sur le canal gauche, sinusoïde fréquence 406 Hz sur le canal droit, pendant 10 minutes.
4. Etape 4 : idem Etape 3, mais avec le volume de chaque canal passant linéairement de 100% à 0%, pendant 5 minutes.

La structure d'un scénario s'y prêtant fortement, cette chronologie est représentée dans un fichier XML lors de la sauvegarde d'un scénario. Voici ce que cela donnerait pour l'exemple précédent :

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <scenario>
3   <title>Exemple de scenario</title>
4   <description>Exemple pour le rapport de PFE</description>
5   <steps>
6     <step id="0"><!-- [...] -->
25    <step id="1"><!-- [...] -->
44    <step id="2"><!-- [...] -->
63   </steps>
64 </scenario>
```

FIGURE 4.1 – Structure générale du fichier XML du scénario de l'exemple précédent

```

25 <step id="1">
26   <leftFreq>
27     <from>400</from>
28     <to>402</to>
29   </leftFreq>
30   <rightFreq>
31     <from>410</from>
32     <to>406</to>
33   </rightFreq>
34   <amplitude>
35     <from>100</from>
36     <to>100</to>
37   </amplitude>
38   <noise>
39     <from>0</from>
40     <to>0</to>
41   </noise>
42   <duration>5</duration>
43 </step>

```

FIGURE 4.2 – Structure XML de l'étape 1 du scénario de l'exemple précédent

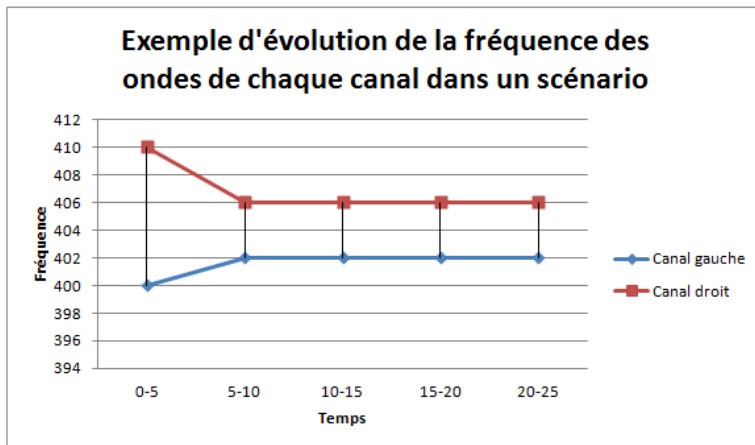


FIGURE 4.3 – Evolution de la fréquence des ondes de chaque canal dans l'exemple de scénario précédent

4.1.2 Déroulement d'une séance de test avec le logiciel

Voici comment se déroulera une séance de test en utilisant le logiciel :

0. Pré-requis : configuration

Le logiciel doit être correctement configuré avant d'effectuer quoi que ce soit. Pour cela, on doit tester si les deux canaux sont bien isolés : si, en jouant un son censé être présent uniquement à droite, on entend du son à gauche, alors la configuration n'est pas bonne. Cela peut être dû à un défaut temporaire de la carte son. Pour être sûr de la configuration, on doit sélectionner un périphérique audio (carte son), et tester les sons dans chaque oreille. Les sons joués pour ces tests sont de simples bruits blancs.

Ensuite, l'utilisateur ne doit pas avoir à changer sa configuration audio matérielle (ie. le volume sonore en sortie de son amplificateur, par exemple). Tout doit se régler au sein du logiciel. Une barre de volume permet de faire ceci. De plus, si on mesure le volume à l'aide d'un sonomètre, on peut enregistrer la valeur réelle (en dB) correspondante à une valeur de volume logiciel. Ainsi, il sera possible de savoir quel sera le volume exact de chaque son à tout moment du scénario.

1. Choix ou création du scénario

L'utilisateur choisit un scénario existant, ou en crée un lui-même.

2. Installation du patient et du matériel

L'appareil d'EEG doit être correctement installé.

- La sortie USB doit être branchée sur un PC muni du logiciel ActiView qui permet d'enregistrer les signaux.
- Le port parallèle doit être relié au PC muni du présent logiciel afin de délivrer des signaux d'événements. C'est sur ce même PC que doit être installé le matériel audio.

Le sujet doit être relié aux électrodes selon le protocole établi par les chercheurs. Il doit en plus être équipé d'un casque stéréophonique, de préférence avec l'arceau placé sur la nuque pour ne pas interférer avec les électrodes.

3. Lecture du scénario

Le chercheur déclenche la lecture du scénario. Le logiciel ActiView doit être lancé.

4. Sauvegarde de l'acquisition

Le chercheur enregistre les données acquises par ActiView, au format BDF.

5. Récupération des données

Les données sont copiées et importées dans le présent logiciel. Elles sont partiellement pré-traitées, et visualisables.

6. Analyse

Le chercheur procède à l'analyse temps-fréquence des données grâce au présent logiciel.

7. Exportation des données

Il est possible d'enregistrer les données pré-traitées, au format BDF, ainsi que les coefficients d'ondelettes, au format texte (matrice). Le chercheur pourra éventuellement effectuer des compléments d'analyse avec le logiciel de son choix.

4.1.3 La fenêtre principale

La fenêtre principale de l'application a été conçue pour que la navigation y soit la plus intuitive possible. Presque chaque étape de la procédure de test représente un onglet.

L'ouverture, la création et la sauvegarde de fichiers de scénarios se fait en haut de la fenêtre. Tout le reste se fait dans les différents onglets, à utiliser dans l'ordre, un par un.

De plus, lorsque l'on essaie d'accéder à un onglet alors que l'on n'a pas encore effectué les étapes préalables, le contenu de cet onglet est grisé, et un message en gras est affiché indiquant ce qu'il faut effectuer avant de passer à cet onglet.

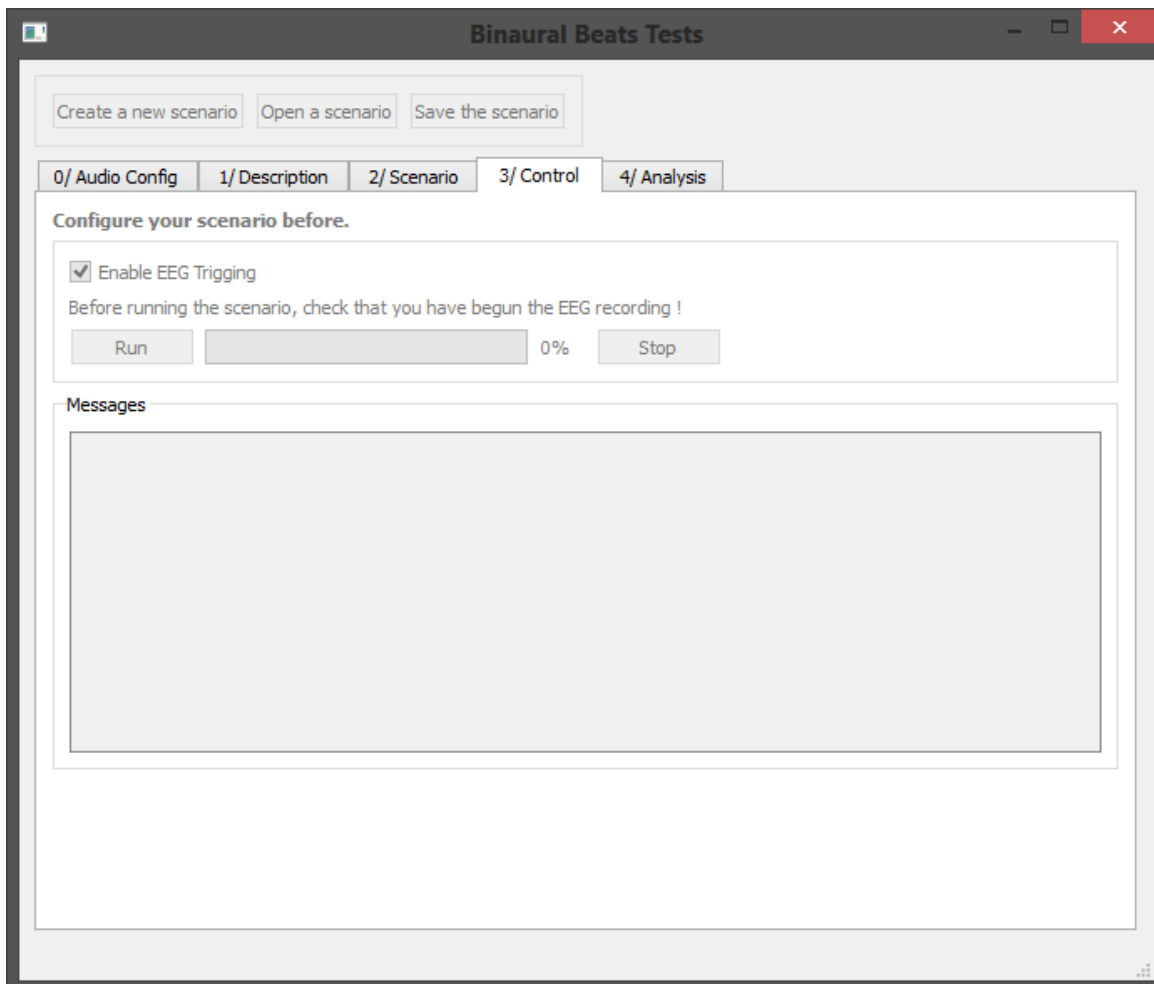


FIGURE 4.4 – Fenêtre principale de l'application. Ici l'onglet de contrôle est inaccessible, car aucun scénario n'a été créé ou chargé.

4.1.4 L'onglet de configuration

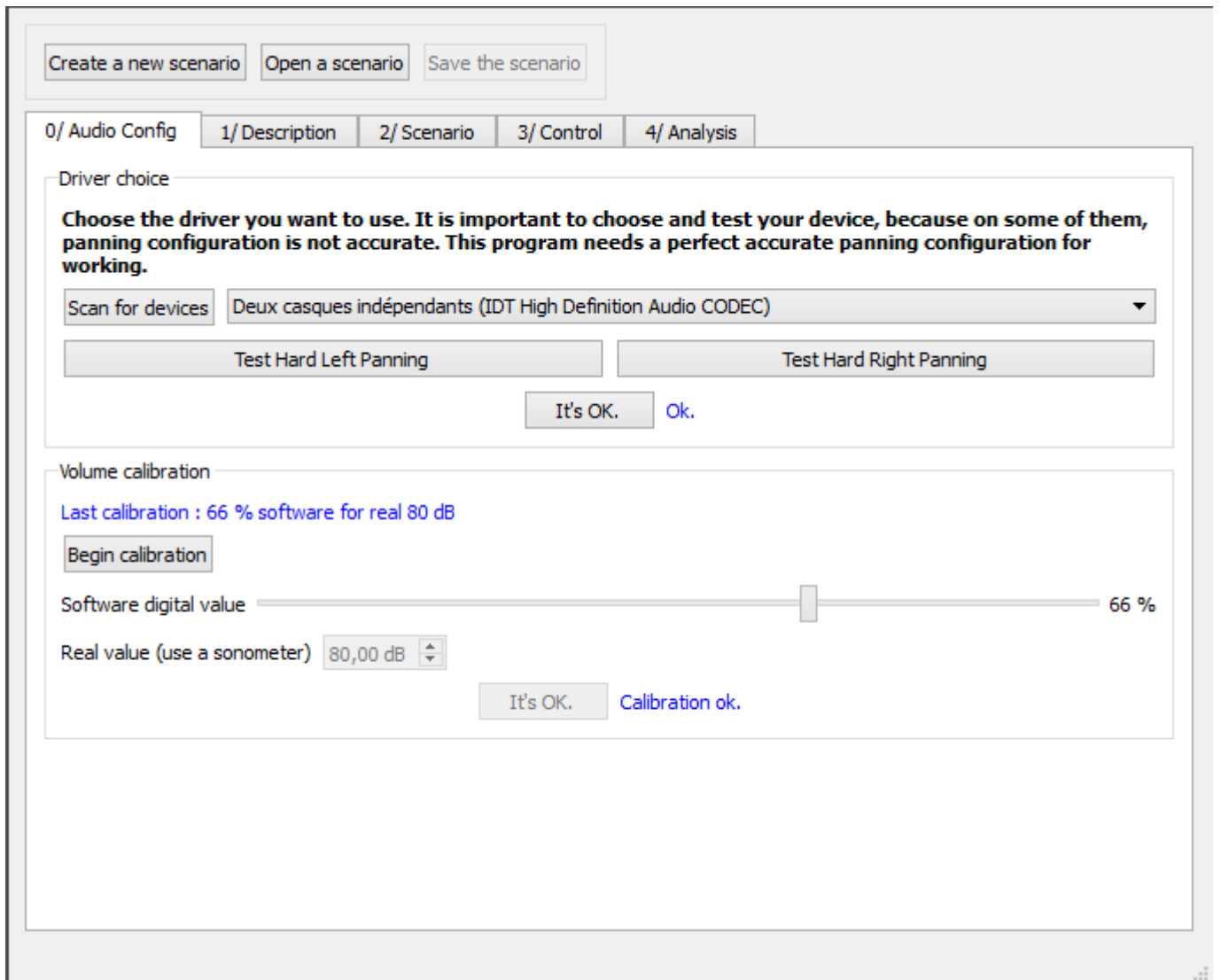


FIGURE 4.5 – L'onglet de configuration

Ce premier onglet est nécessaire pour choisir le bon périphérique audio et le tester. En effet, il est possible que dans certains cas, suivant le périphérique choisi, la balance ne soit pas correcte. Par exemple, en positionnant un son uniquement à gauche, on va tout de même entendre un léger son à droite. C'est pour cela que les deux boutons de test sont présents : ils permettent chacun d'écouter du bruit blanc, soit uniquement à gauche, soit uniquement à droite. Ainsi, il n'est pas possible que l'expérience soit faussée par une mauvaise balance.

De plus, l'équipe de chercheurs intéressée par le projet a émis le désir de pouvoir régler le volume sonore directement via le logiciel. En effet, leur volume est réglé sur Windows au maximum, et l'amplificateur est réglé sur une valeur fixée et enregistrée. Cela est plus simple pour l'étalonnage. C'est pourquoi, via le logiciel, on peut effectuer une calibration du volume, lors de laquelle on fixe le volume que l'on veut. Toujours suggéré par l'équipe, il est possible de spécifier le volume réel en décibels (à mesurer via un sonomètre) correspondant au volume logiciel que l'on a fixé. Ainsi, lorsque l'on fixera le volume d'un son à 50% par exemple, sachant que l'on a fixé la calibration à 80dB, alors le logiciel est en mesure de dire que le volume réel de sortie sera de 40dB.

4.1.5 L'onglet de description de scénario

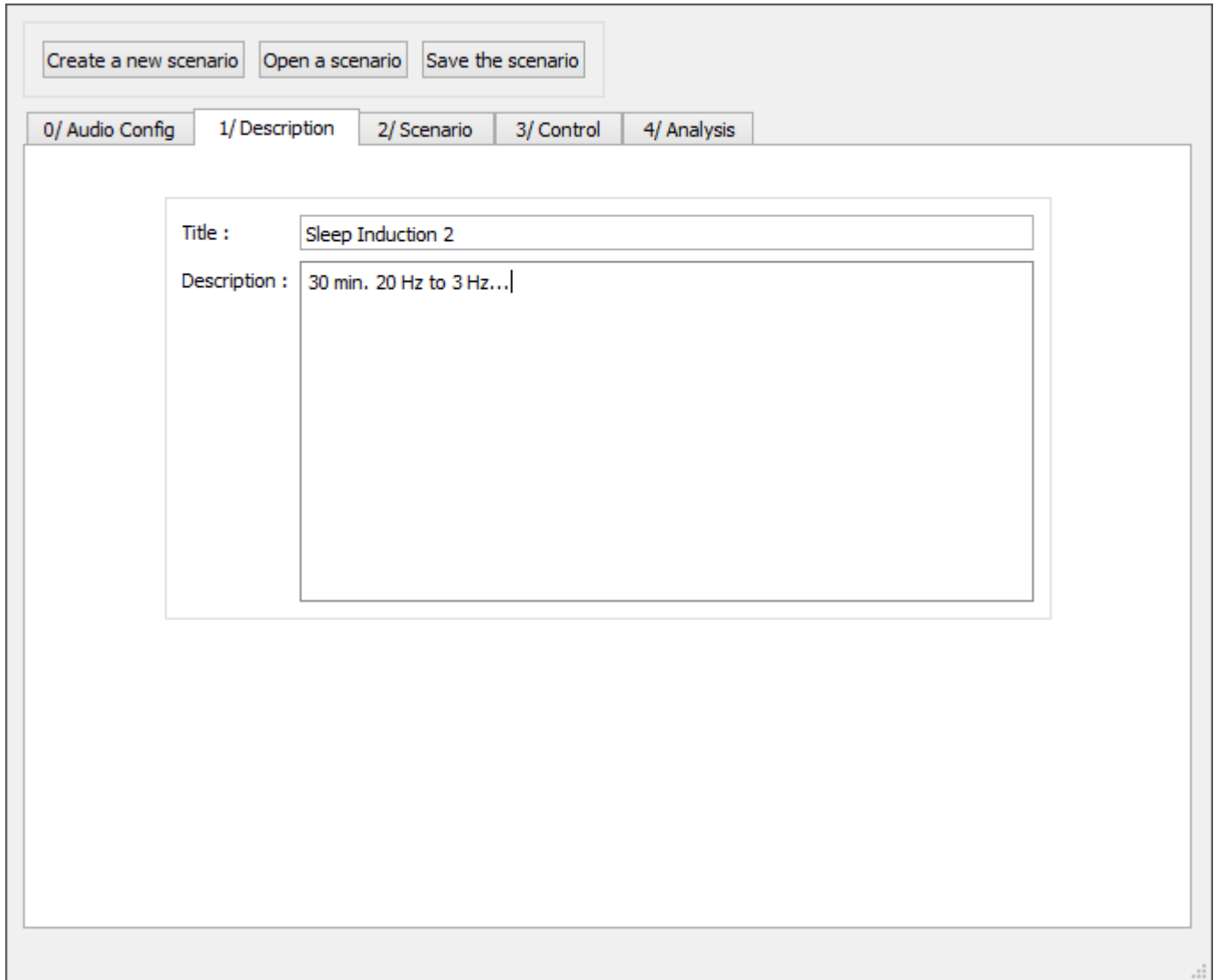


FIGURE 4.6 – L'onglet de description de scénario

Cet onglet sert simplement à donner un titre au scénario, ainsi qu'une description. Cette fonction est laissée libre au chercheur qui peut en faire ce qu'il veut.

4.1.6 L'onglet d'édition de scénario

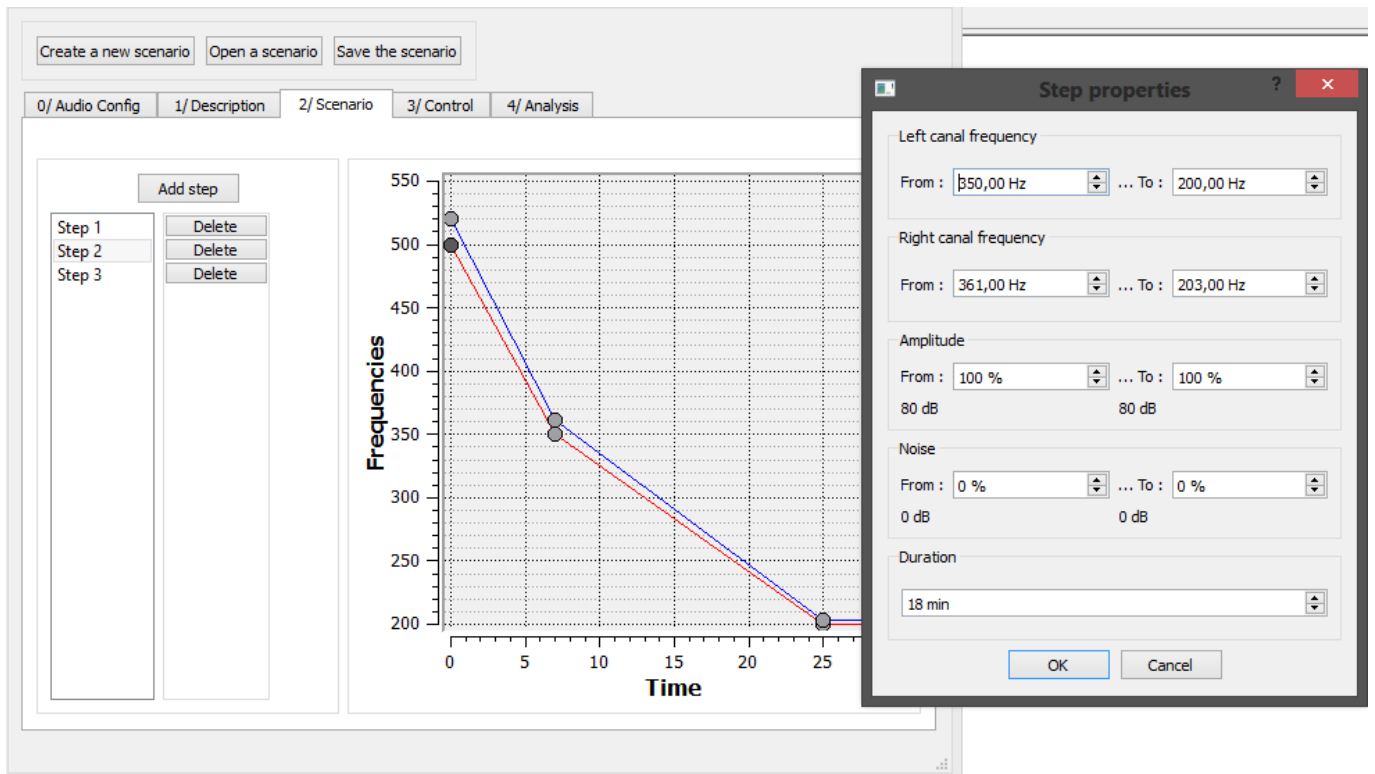


FIGURE 4.7 – L'onglet d'édition de scénario, avec la boîte de dialogue d'édition d'étape

C'est ici que le chercheur conçoit son scénario en paramétrant les différentes étapes.

Il est possible de le faire directement sur le graphe : un clic droit ajoute un point à la courbe, et le clic prolongé sur un point entraîne son déplacement selon les mouvements de la souris. Le déplacement est "aimanté" dans le sens des abscisses : il suit automatiquement le quadrillage, et il est impossible de réaliser une courbe incohérente (par exemple le point $i+1$ placé avant le point i). Cette méthode n'est pas des plus précises pour le paramétrage des fréquences : celles-ci pouvant être réglées au centième près, il est difficile d'avoir précisément la fréquence que l'on veut uniquement avec la souris.

Si l'on désire paramétrer précisément les fréquences, on peut ajouter les étapes via une interface textuelle. En cliquant sur le bouton "Add Step", une boîte de dialogue s'ouvre pour régler tous les paramètres de l'étape : fréquences des deux signaux, amplitude des signaux et du bruit, durée.

Quelle que soit la manière dont aura été ajoutée l'étape, elle sera ajoutée dans la liste présente à gauche de la fenêtre. En cliquant, on accède directement à la boîte de dialogue permettant de la modifier. Il est aussi possible de la modifier directement en déplaçant les points de la courbe.

4.1.7 L'onglet de contrôle de scénario

Lorsque le scénario est choisi, il faut le jouer. C'est dans cet onglet que cela se passe.

La case à cocher permet d'activer ou non le triggering, c'est-à-dire l'envoi de signaux d'événements au matériel Biosemi, afin de signaler les changements d'étape du scénario (utile pour l'analyse).

Pour démarrer la lecture, il faut cliquer sur le bouton Run. Le bouton Stop arrête tout et remet la lecture à 0.

A chaque modification de paramètre, la zone de messages l'indique en temps réel.

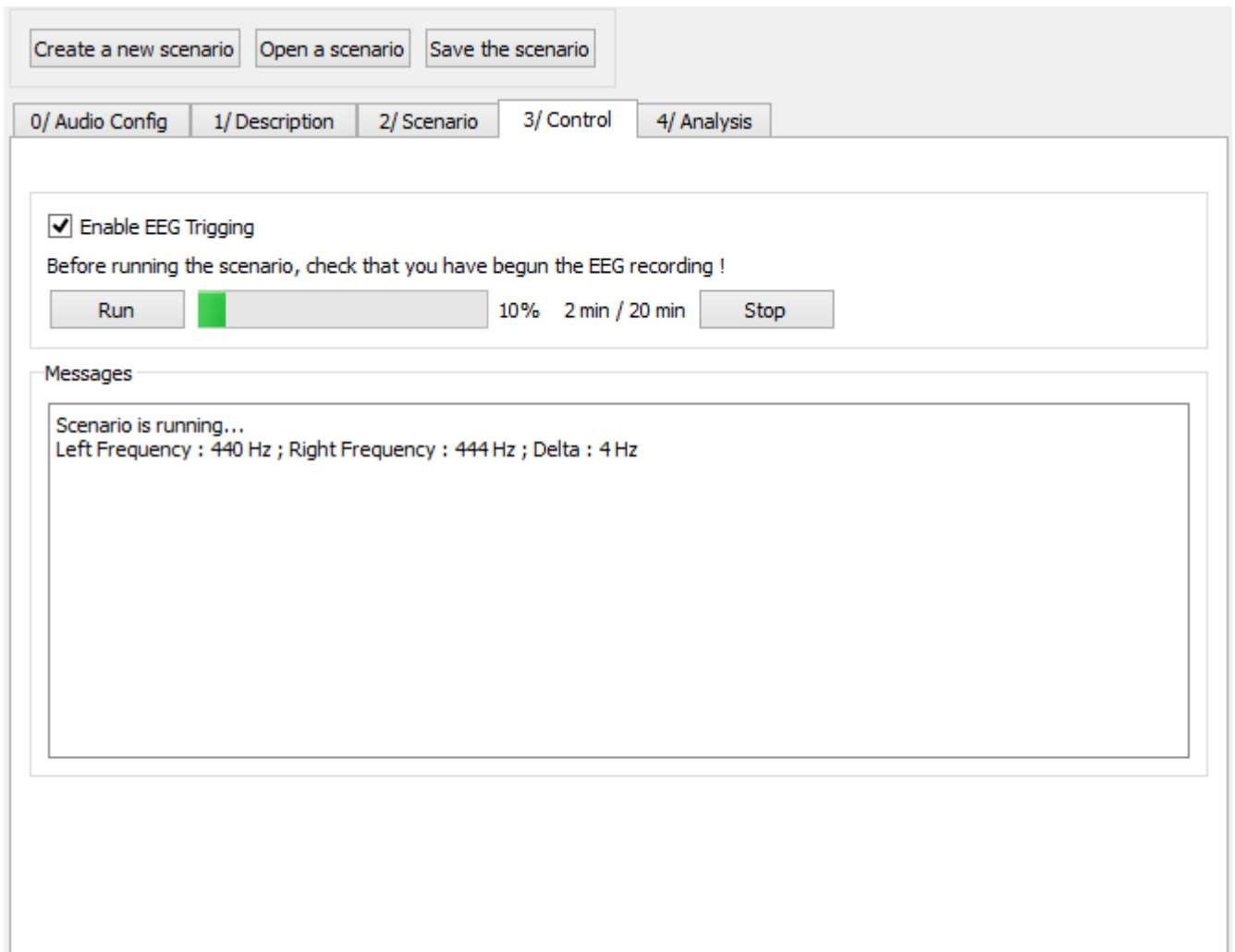


FIGURE 4.8 – L'Onglet de contrôle

4.1.8 L'onglet d'analyse

Cette section permet d'importer un fichier BDF enregistré par Biosemi ActiView. Lorsque c'est fait, le programme tente de trouver les canaux correspondant au pré-traitement. S'il les a trouvés, il le signale à l'utilisateur mais lui laisse le choix de valider ou modifier les affectations des canaux de pré-traitement. S'il ne les a pas trouvés, il demande à l'utilisateur de les affecter manuellement, ou de ne pas les activer.

Pour canaux de pré-traitement activés, le pré-traitement se lance automatiquement et l'utilisateur doit patienter jusqu'à la fin de la procédure.

Il est ensuite possible de visualiser les signaux en cliquant sur le bouton "Show Raw Signals", ou d'effectuer l'analyse d'un signal en cliquant sur "Analyze".

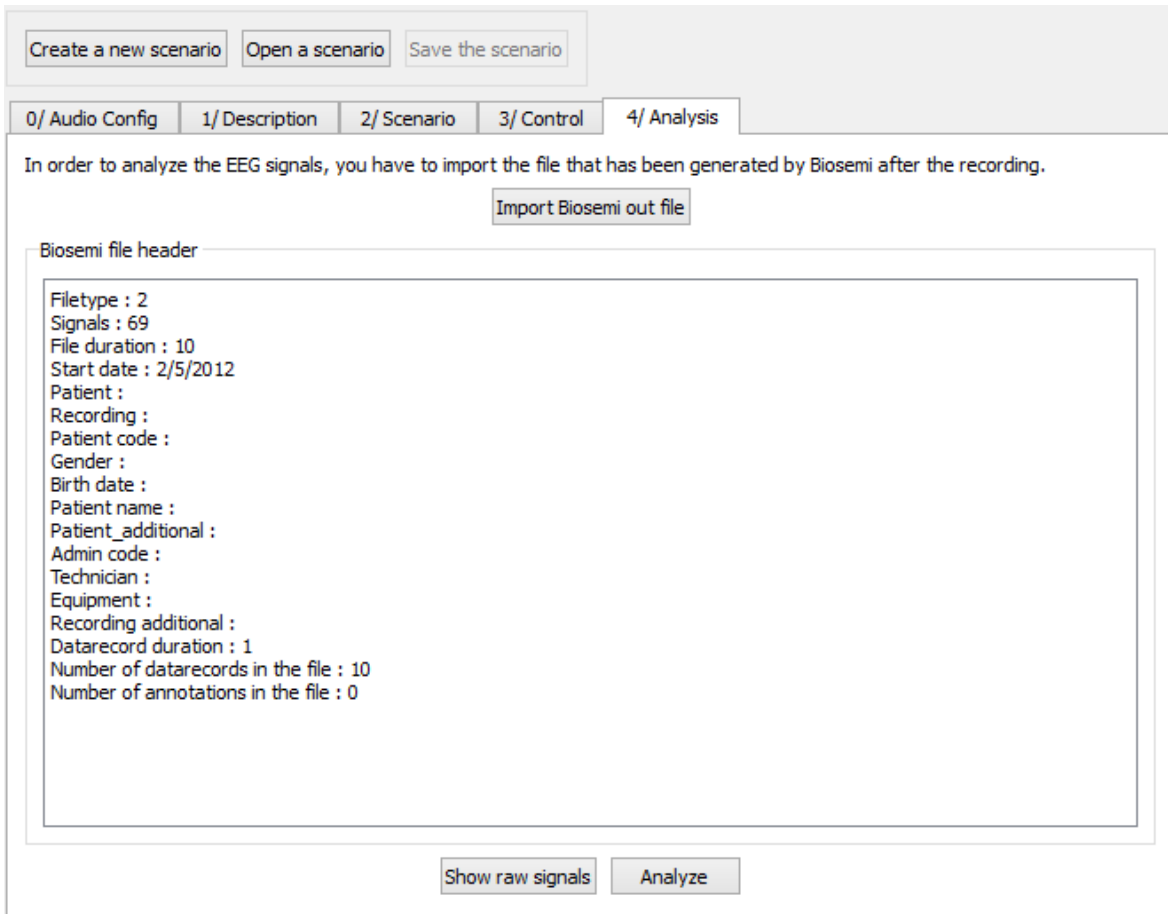


FIGURE 4.9 – L'onglet d'analyse.

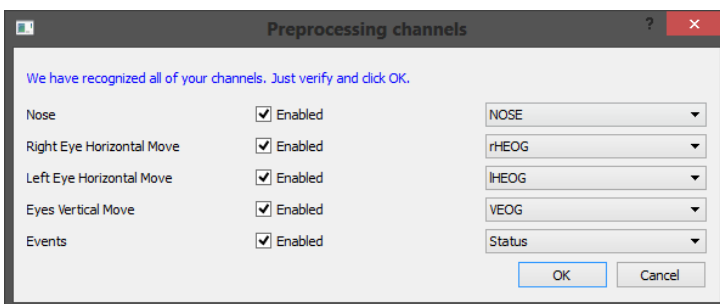


FIGURE 4.10 – Boîte de dialogue de sélection des canaux de pré-traitement

4.1.9 La fenêtre de visualisation des signaux

La visualisation des signaux se fait dans une fenêtre à part, constituée d'une liste des signaux disponibles, et de la représentation graphique du signal sélectionné dans la liste. Il est possible de zoomer très finement en dessinant des rectangles avec la souris dans la zone d'affichage. Le zoom arrière se fait avec le clic droit de la souris.

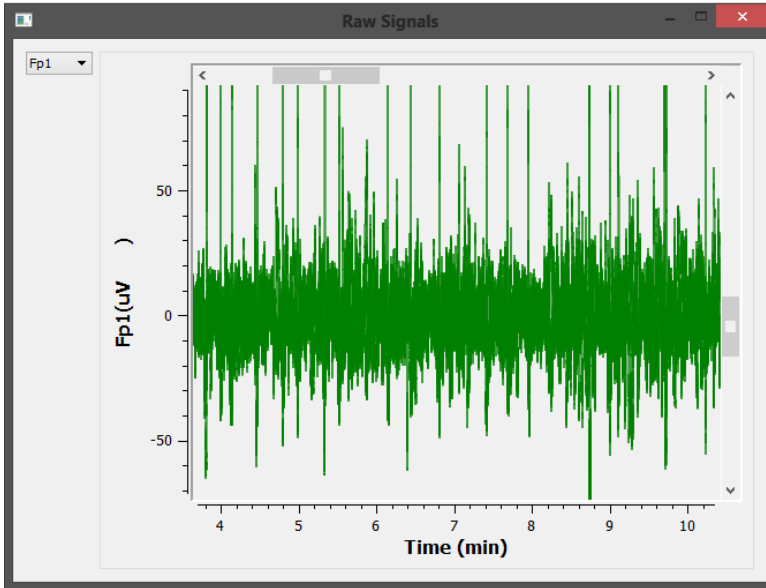


FIGURE 4.11 – Fenêtre de visualisation des signaux.

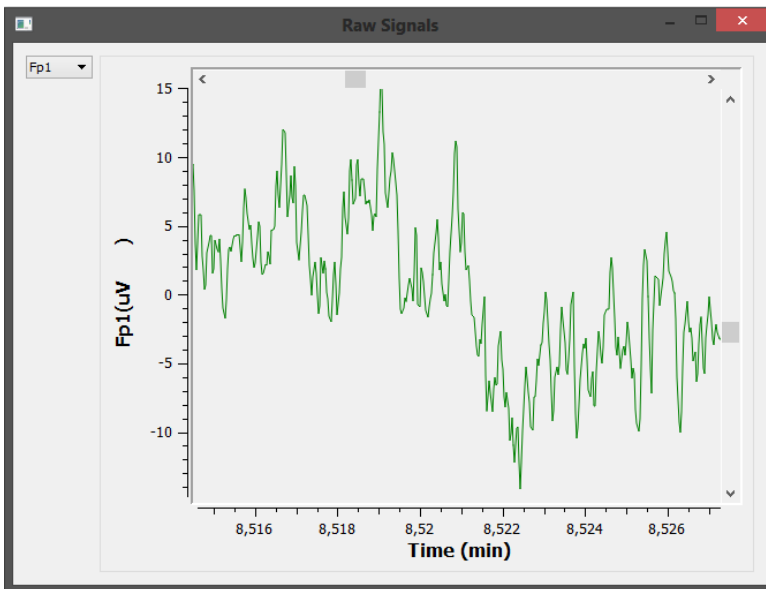


FIGURE 4.12 – Fenêtre de visualisation des signaux, avec un zoom.

4.1.10 La fenêtre d'analyse

Choix du signal à analyser

Il n'est pas possible d'analyser tous les signaux à la fois, dans un souci de temps de calcul et de préservation de la mémoire vive. C'est pourquoi il faut d'abord sélectionner le signal que l'on désire analyser avant de commencer.

On peut aussi choisir la bande de fréquences que l'on souhaite analyser. Attention à ne pas choisir une bande trop large, car la quantité de mémoire vive utilisée est très vite importante.

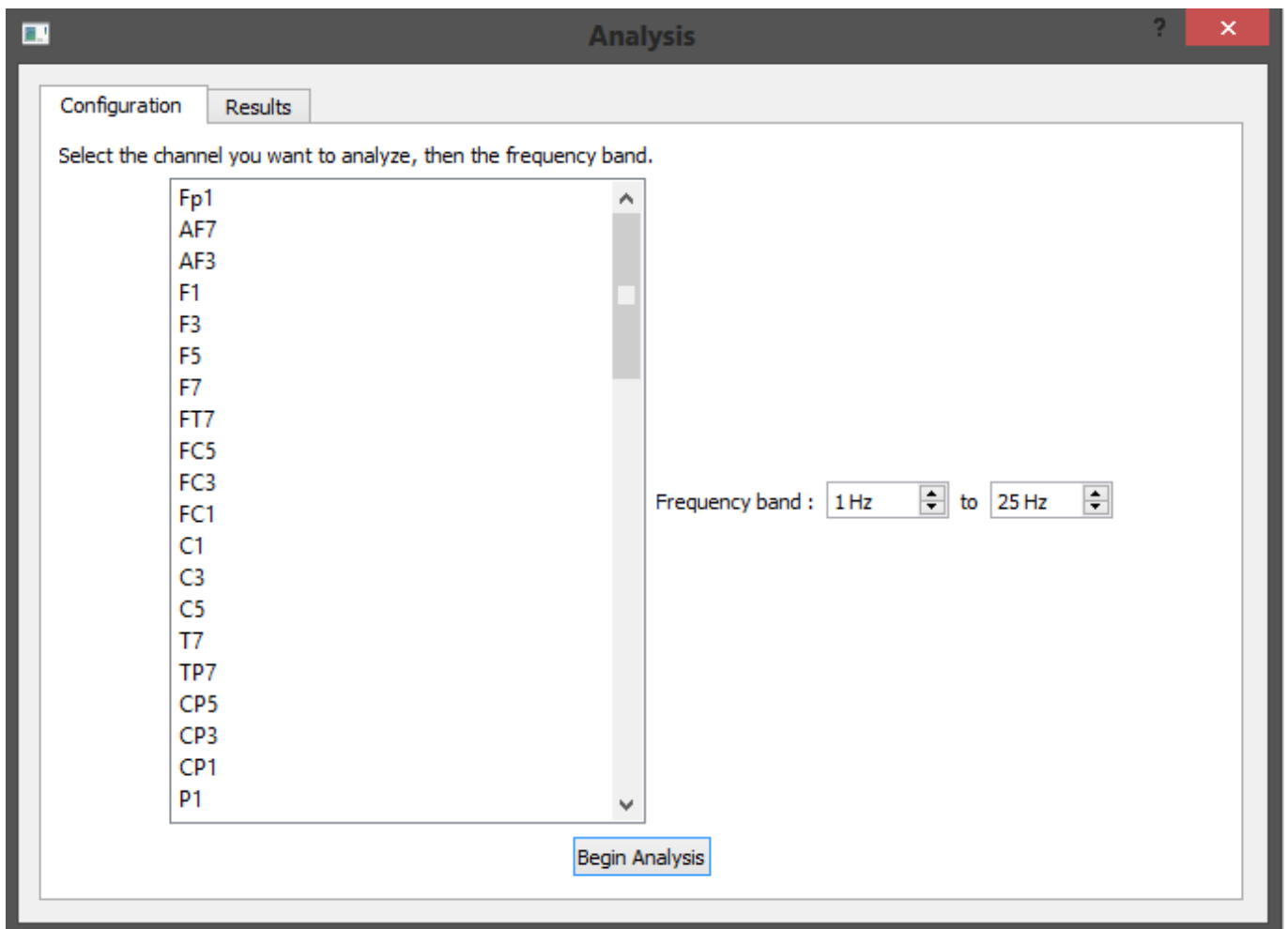


FIGURE 4.13 – Choix du signal à analyser

Représentation temps-fréquence

Le signal est analysé via la transformée en ondelettes continue. L'ondelette mère utilisée est Morlet. Les résultats de l'analyse sont des coefficients rangés dans une matrice échantillons-échelle, convertie en matrice temps-fréquence. La résolution de l'analyse est de 3 : temps, fréquence, et puissance. La puissance est caractérisée par la valeur de chaque coefficient, et représentée par une couleur sur le graphe. Ainsi on peut voir avoir un spectre fréquentiel sur la gamme que l'on veut, sur une longue période de temps.

Il est possible de zoomer en dessinant des rectangles avec la souris, et de revenir en arrière en effectuant un clic droit. De plus, l'échelle de couleur peut être modifiée : si on a du mal à distinguer certaines fréquences, il suffit d'agrandir la zone d'acceptation de la couleur rouge par exemple.

Enfin, on peut exporter les résultats sous forme de matrice stockée dans un fichier texte.

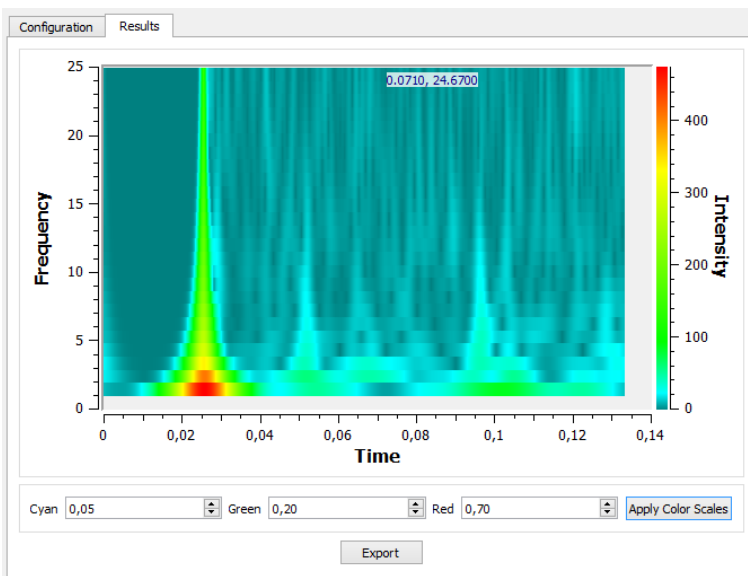


FIGURE 4.14 – Représentation de l'analyse temps-fréquence

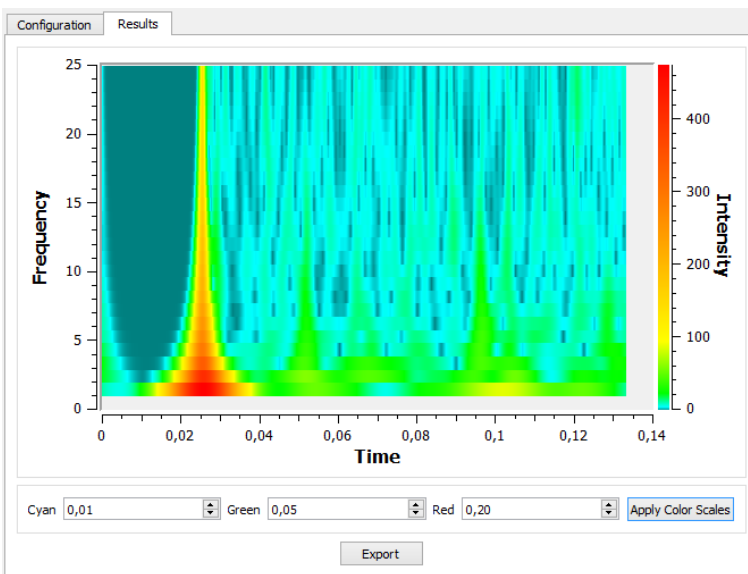


FIGURE 4.15 – Représentation de l'analyse temps-fréquence, avec de nouveaux paramètres d'échelle de couleur

4.2 Modélisation

4.2.1 Architecture générale

Le logiciel est basé sur une architecture trois-tiers dont voici les différentes parties :

- Couche *Accès aux données*. C'est la partie qui accède aux données persistantes. Ici, il s'agit de fichiers XML et BDF.
- Couche *Logique métier*. C'est la partie fonctionnelle du logiciel, elle traite les données et les transmet aux deux autres couches.
- Couche *Présentation*. C'est la partie graphique, directement en relation avec l'utilisateur : elle lui transmet les informations venant de la couche Logique Métier, et intercepte en premier les actions de l'utilisateur.

La communication n'est possible qu'entre couches adjacentes.

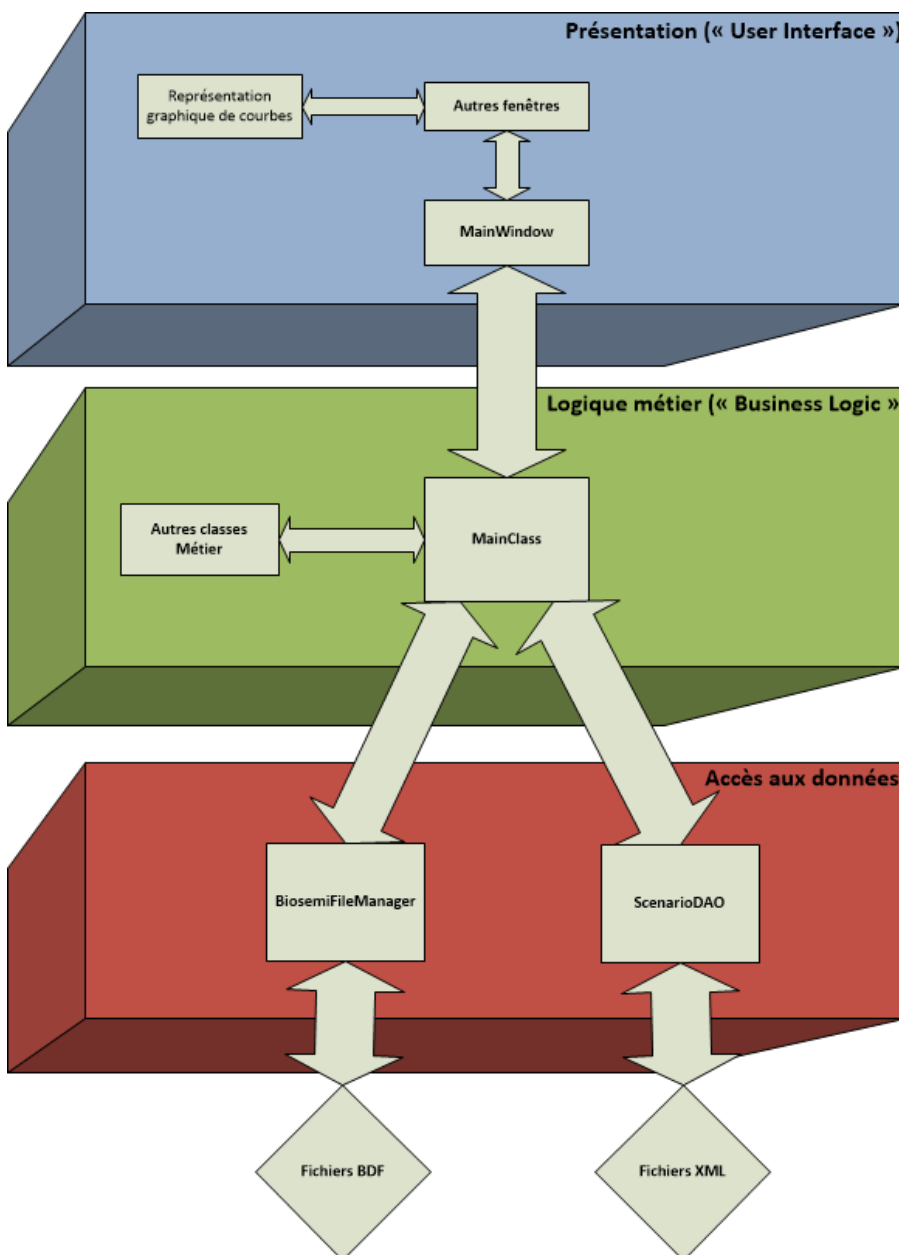


FIGURE 4.16 – Architecture générale de l'application

Cette séparation entre couches permet une meilleure maintenabilité du logiciel. En effet, celui-ci a été développé de manière à **faciliter sa reprise lors de projets futurs**. Dans l'arborescence du projet, les couches sont représentées par des dossiers.

La point de départ de l'application est située dans la couche Logique métier. Il s'agit de la classe `MainClass`. C'est elle qui coordonne toutes les autres classes, et permet la communication entre couches adjacentes. Elle peut directement appeler les méthodes publiques des classes principales. La communication dans le sens "autre classe" -> "`MainClass`" se fait par la transmission de signaux Qt. Le système de signaux est efficace notamment lorsque l'on utilise le multithreading. Or, dans Qt, la gestion de l'interface graphique est toujours faite dans un thread à part. Elle peut ainsi envoyer et recevoir des informations tout en laissant les algorithmes tourner.

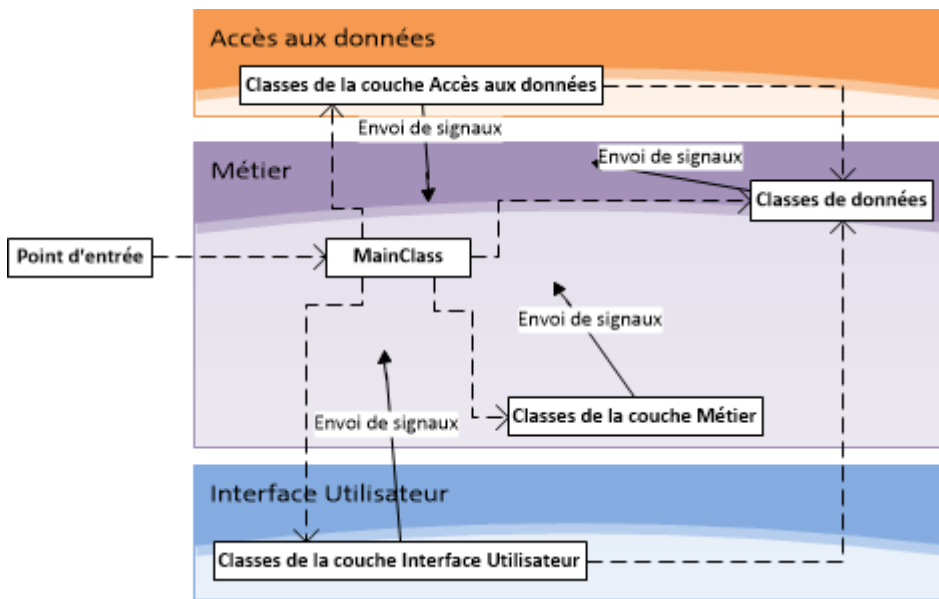


FIGURE 4.17 – Interactions entre couches

Sur le diagramme ci-dessus, on voit les interactions possibles entre classes de couches différentes. `MainClass` est instanciée en premier et instancie les autres. Elle communique directement avec elles. Les autres classes envoient des signaux, que `MainClass` est capable de recevoir.

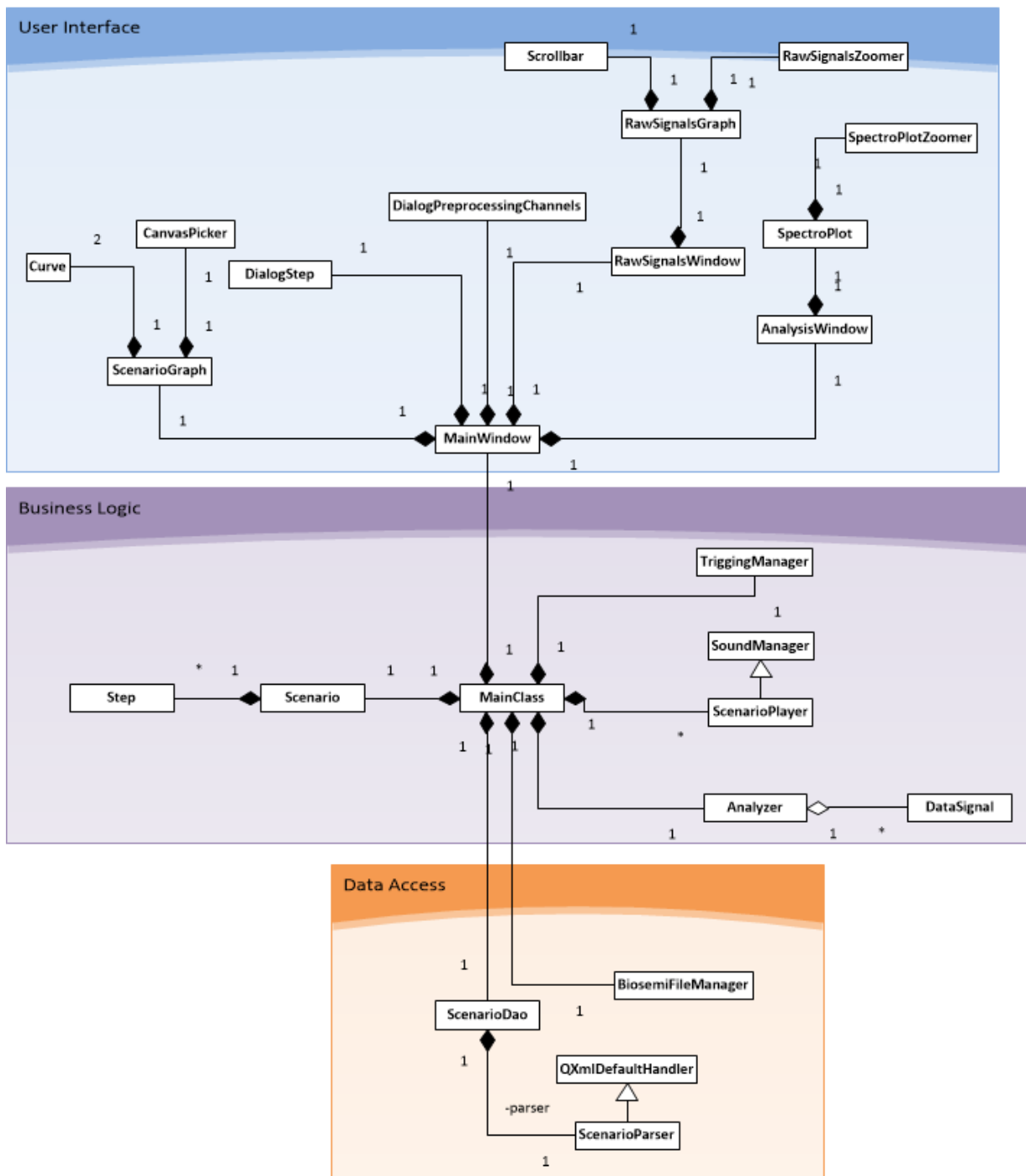
Il faut aussi noter que dans la couche Métier, il existe les *classes de données*. Ces classes représentent certaines structures de données, qui sont utilisables par toutes les autres couches. C'est l'**unique exception** dans le principe de l'interaction "Métier - Autres couches" qui normalement ne se fait que par l'intermédiaire de la classe `MainClass`.

4.2.2 Modélisation des classes

Cette partie traite de la modélisation des classes de chaque couche, une par une.

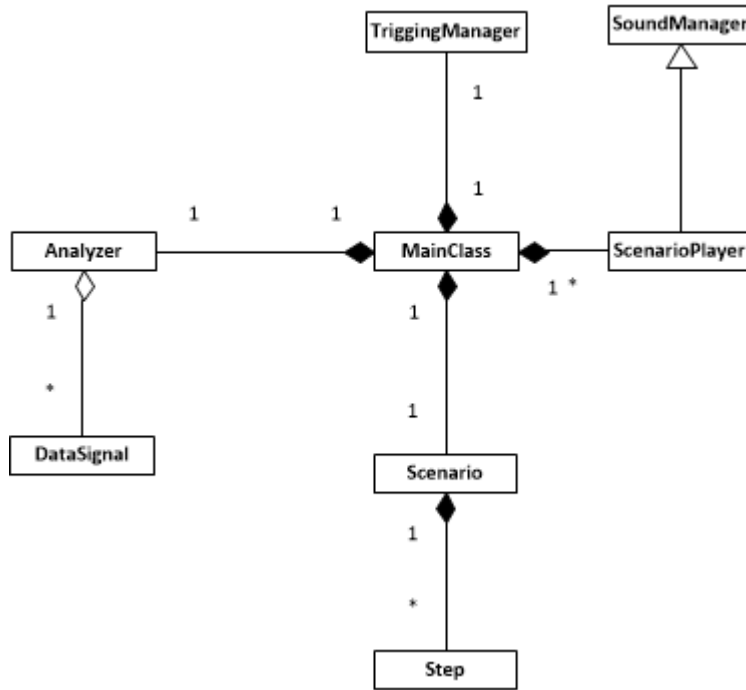
Le diagramme des classes qui suit permet d'avoir une vision générale de l'architecture du logiciel tout en ayant connaissance du détail des classes.

Les interactions entre classes de données (Scenario, Step, DataSignal) et autres classes ne sont pas affichées ici, car cela alourdirait le schéma.



Classes de la couche Logique métier

La couche Logique Métier est la couche d'entrée de l'application, de par sa classe MainClass. Elle instancie les différents modules, que ce soit ceux de sa propre couche ou ceux des autres couches.



Classes Scenario et Step

Les classes Scenario et Step sont des classes de données. Elles peuvent donc être utilisées par toutes les classes de toutes les couches (cf. partie précédente).

La classe Scenario est une représentation d'un scénario. Step représente une étape d'un scénario. Un scénario contient ainsi une liste d'objets Step.

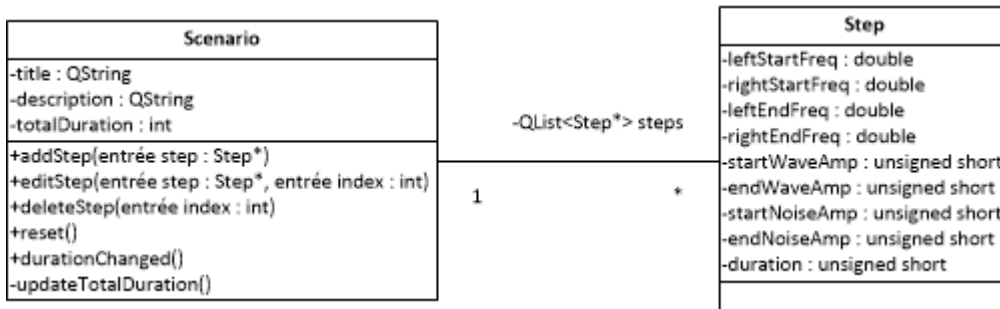


FIGURE 4.18 – Diagramme de classes de Scenario et Step

- Attributs de Scenario
 - title : titre du scénario.
 - description : description du scénario.
 - totalDuration : durée totale du scénario (somme des durées de toutes les étapes).
 - steps : liste d'étapes (objets Step) du scénario.
- Méthodes de Scenario
 - addStep : ajoute une étape au scénario.
 - editStep : édition d'une étape du scénario.

- deleteStep : suppression d'une étape du scénario.
- durationChanged : recalcule la durée totale du scénario (lorsque la durée d'une étape est modifiée, cette méthode est automatiquement appelée).
- accesseurs et modificateurs

- Attributs de Step
 - leftStartFreq : fréquence de l'onde du canal gauche au début de l'étape
 - rightStartFreq : fréquence de l'onde du canal droit au début de l'étape
 - leftEndFreq : fréquence de l'onde du canal gauche à la fin de l'étape
 - rightEndFreq : fréquence de l'onde du canal droit à la fin de l'étape
 - startWaveAmp : amplitude des ondes en pourcentage au début de l'étape
 - endWaveAmp : amplitude des ondes en pourcentage à la fin de l'étape
 - startNoiseAmp : amplitude du bruit en pourcentage au début de l'étape
 - endNoiseAmp : amplitude du bruit en pourcentage à la fin de l'étape
 - duration : durée de l'étape en minutes

Classe DataSignal

La classe DataSignal est aussi une classe de données.

Elle regroupe les données d'un signal sous forme de liste de nombres réels, et des informations utiles telles que l'entête EDF/BDF du signal. Elle permet de charger ou décharger en mémoire vive les données sans pour autant supprimer l'objet et les informations importantes.

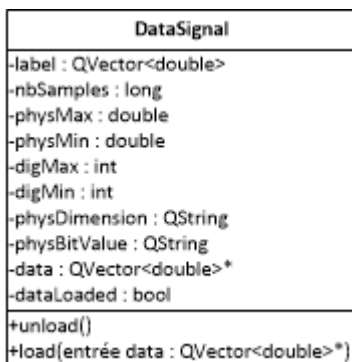


FIGURE 4.19 – Diagramme de classe de DataSignal

- Attributs
 - label : nom du signal/canal
 - nbSamples : nombre d'échantillons
 - physMax : valeur physique maximale
 - physMin : valeur physique minimale
 - digMax : valeur numérique maximale
 - digMin : valeur numérique minimale
 - physDimension : dimension physique (par exemple : μV)
 - physBitValue : correspondance bit/valeur physique. Permet de convertir entre valeurs physiques et valeurs numériques.
 - data : données du signal
 - dataLoaded : booléen indiquant si les données sont chargées en mémoire vive ou non
- Méthodes
 - unload : décharge la mémoire vive des données du signal
 - load : charge les données du signal

Classe TriggManager

La classe TriggManager isole dans ses méthodes du code en langage C pour envoyer des signaux d'événements via le port parallèle.

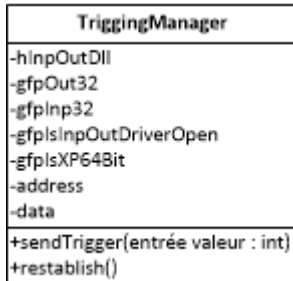


FIGURE 4.20 – Diagramme de classe de TriggManager

- Attributs
 - hInpOutDll : instance Win32 de la bibliothèque dynamique InpOut32
 - gfpInp32 : foncteur vers la fonction Inp32 qui permet de lire dans le port parallèle
 - gfpOut32 : foncteur vers la fonction Out32 qui permet d'écrire dans le port parallèle
 - gfplInpOutDriverOpen : foncteur vers la fonction IsInpOutDriverOpen qui permet de savoir si le pilote InpOut est ouvert
 - gfplsXP64Bit : foncteur vers la fonction IsXP64Bit qui permet de savoir si le système est en 64 bit.
- Méthodes
 - sendTrigger : envoi d'un signal d'événement (valeur à partir de 1), attend 5 millisecondes puis appelle la fonction reestablish.
 - reestablish : rétablit le port parallèle à la valeur 0.

Classe Analyzer

La classe Analyzer s'occupe du pré-traitement des données ainsi que de leur analyse. Elle possède un pointeur vers la liste des signaux et modifie directement les données lors du pré-traitement. Dans l'étape d'analyse, de nouvelles données sont créées à partir des données initiales (ou pré-traitées).

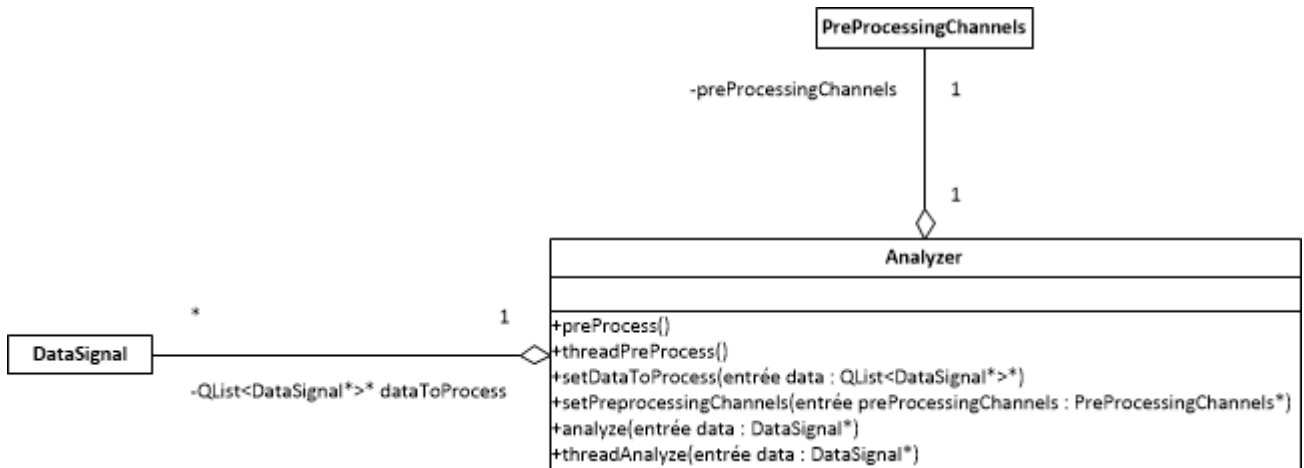


FIGURE 4.21 – Diagramme de classes de Analyzer

- Attributs
 - dataToProcess : les données des signaux
 - preProcessingChannels : les signaux de prétraitement et d'événements
- Méthodes
 - preProcess : lancement du prétraitement
 - threadPreProcess : prétraitement. Se fait dans un thread à part.
 - setDataToProcess : assignation des données à traiter
 - setPreprocessingChannels : assignation des canaux de prétraitement
 - analyze : lancement de l'analyse du signal indiqué.
 - threadAnalyze : analyse. Se fait dans un thread à part.

Classes SoundManager et ScenarioPlayer

La classe ScenarioPlayer gère la lecture du scénario. La notion de temps et de son intervient. Le temps est géré par un objet de type QTimer. Pour ce qui est du son, ScenarioPlayer est en fait une classe dérivée de la classe SoundManager, qui s'occupe exclusivement de la gestion de l'audio.

- Attributs de SoundManager
 - leftFreq : fréquence de la sinusoïde du canal gauche
 - rightFreq : fréquence de la sinusoïde du canal droit
 - waveAmp : amplitude des ondes en pourcentage (appliqué aux deux canaux)
 - noiseAmp : amplitude du bruit (appliqué au canal stéréo du bruit)
 - softwareVolFactor : facteur de conversion de volume utilisateur
 - fmodSystem : système audio FMOD
 - fmodDspLeftOsc : oscillateur FMOD gauche
 - fmodDspRightOsc : oscillateur FMOD droit
 - fmodChanLeft : canal FMOD gauche
 - fmodChanRight : canal FMOD droit
 - fmodChanNoise : canal FMOD du bruit (stéréo)
 - fmodWhiteNoise : son FMOD de bruit blanc

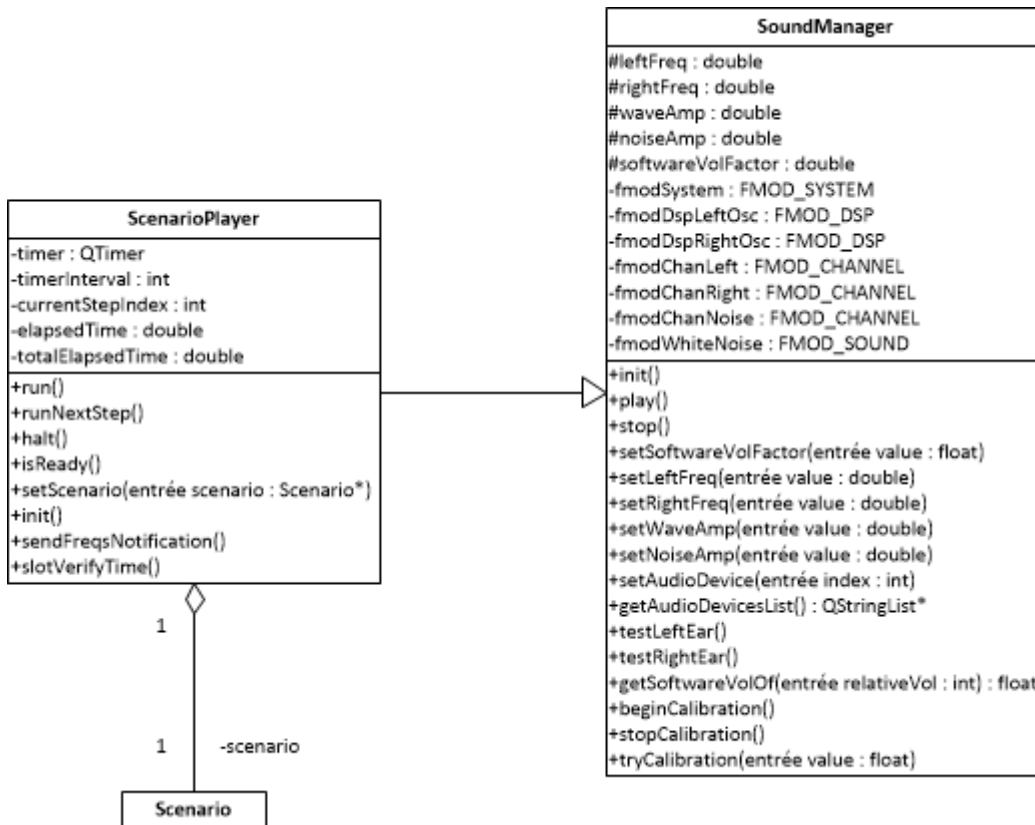


FIGURE 4.22 – Diagramme de classes de SoundManager et ScenarioPlayer

- Méthodes de SoundManager
 - play : début de la génération/lecture du son
 - stop : arrêt de la lecture du son
 - setAudioDevice : choix du périphérique audio (carte son) à utiliser
 - getAudioDevicesList : retourne une liste de chaînes de caractères correspondant aux périphériques audio disponibles sur l'ordinateur
 - testLeftEar : teste le son du canal gauche en y jouant un bruit blanc
 - testRightEar : même chose pour le canal droit
- Attributs de ScenarioPlayer
 - timer : gère le temps
 - scénario : pointeur vers le scénario à jouer
 - timerInterval : résolution du timer en millisecondes
 - currentStepIndex : numéro de l'étape en cours du scénario
 - elapsedTime : temps écoulé depuis le début de la lecture de l'étape en cours
 - totalElapsedTime : temps écoulé depuis le début de la lecture du scénario
- Méthodes de ScenarioPlayer
 - run : début de la lecture du scénario
 - runNextStep : lecture de l'étape suivante
 - halt : arrêt
 - isReady : retourne vrai si tous les paramètres sont corrects pour permettre la lecture
 - sendFreqsNotification : envoie un message indiquant les fréquences en cours
 - verifyTime : vérification des canaux à chaque intervalle de temps

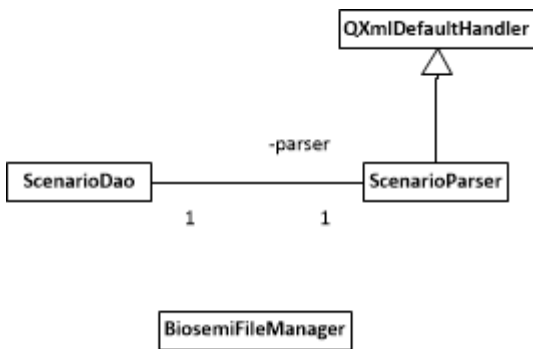
Classes de la couche Accès au données

La couche Accès aux données permet de :

- lire dans des fichiers, et charger leur contenu en mémoire vive dans des objets de la couche Métier
- écrire dans des fichiers le contenu d'objets de la couche Métier.

Les fichiers concernés sont :

- des fichiers XML stockant des scénarios
- des fichiers BDF contenant les données des signaux enregistrés par Biosemi ActiView.



Classe ScenarioParser

La classe ScenarioParser hérite de la classe QXmlDefaultHandler, qui permet de parcourir un fichier XML en permettant d'attribuer des actions à chaque détection de début ou fin de balise.

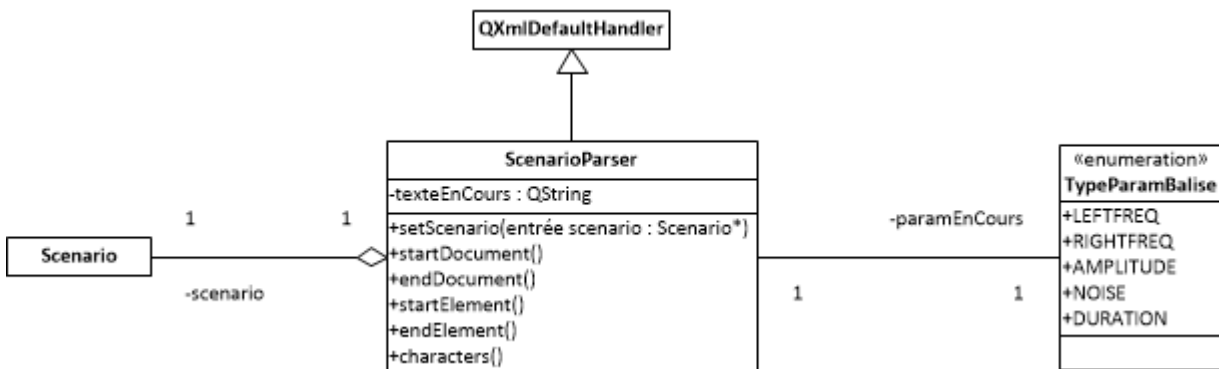


FIGURE 4.23 – Diagramme de classes de ScenarioParser

Méthodes importantes de QXmlDefaultHandler réimplémentées par ScenarioParser :

- startElement. Cette méthode permet d'effectuer des actions à la rencontre d'un début de balise XML. Dans notre cas, on teste le nom de la balise. S'il s'agit d'une balise "step", on ajoute une nouvelle étape à l'objet Scenario.
- endElement. Cette méthode permet d'effectuer des actions à la rencontre d'une fin de balise XML. Suivant le nom de la balise, s'il s'agit d'un paramètre d'étape, on modifie l'étape en cours du scénario.

Classe ScenarioDao

La classe ScenarioDao utilise ScenarioParser pour charger les données du fichier XML du scénario. Il lui donne un pointeur vers l'objet Scenario à remplir.

ScenarioDao effectue les actions de lecture et écriture demandées par la couche Métier.

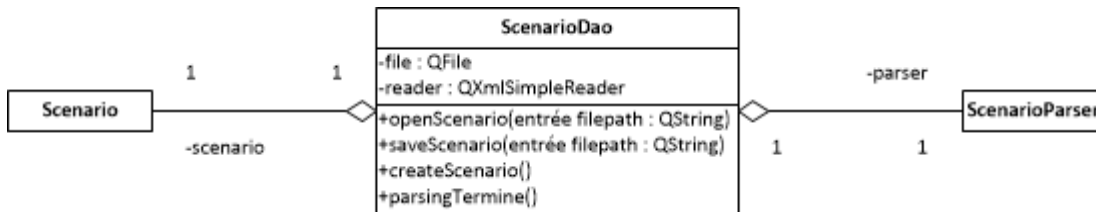


FIGURE 4.24 – Diagramme de classes de ScenarioDao

- Attributs
 - file : objet QFile permettant l'accès au fichier du scénario
 - scenario : pointeur vers l'objet Scenario
 - reader : objet QDomSimpleReader permettant l'accès en lecture au fichier XML
 - parser : objet ScenarioParser qui remplit l'objet Scenario en fonction du contenu du fichier XML
- Méthodes
 - openScenario : lit un scénario au chemin indiqué (chargement des données du fichier XML vers l'objet Scenario)
 - saveScenario : sauvegarde le scénario en cours dans un fichier au chemin indiqué (écriture des données de l'objet Scenario dans le fichier XML)
 - createScenario : crée un nouveau scénario vierge (création d'un nouvel objet Scenario)
 - parsingTermine : méthode appelée automatiquement par ScenarioParser lorsque le fichier XML est entièrement parcouru, donc lorsque l'objet Scenario est complètement chargé.

Classe BiosemiFileManager

La classe BiosemiFileManager se charge de l'accès aux fichiers contenant les signaux enregistrés par Biosemi au format BDF. Elle permet aussi de gérer le chargement ou déchargement en RAM de certains signaux afin d'optimiser la quantité de mémoire vive utilisée par le programme. Cela est indispensable, dans la mesure où un fichier BDF contenant 69 signaux de 20 minutes échantillonnés à 512 Hz pèse déjà plus de 130 Mo, et que l'on a besoin de charger en mémoire vive et simultanément plusieurs matrices carrées dont la dimension est le nombre d'échantillons : cela peut dépasser le giga de mémoire vive utilisée !

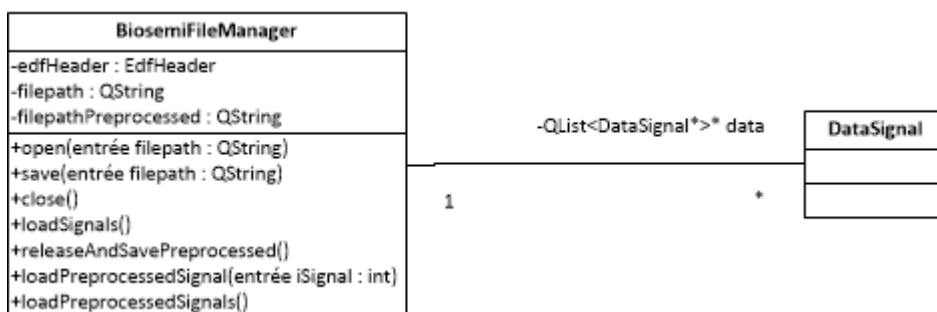


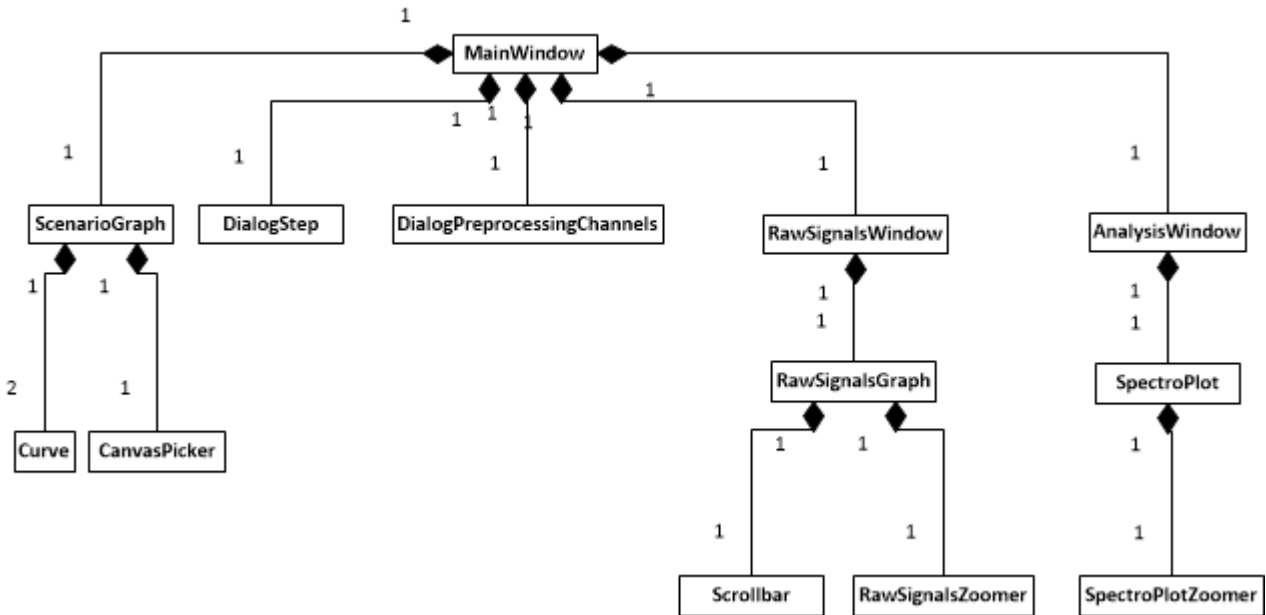
FIGURE 4.25 – Diagramme de classes de BiosemiFileManager

- Attributs
 - edfHeader : données d'entête du fichier BDF

- filepath : chemin du fichier BDF
- filepathPreprocessed : chemin du fichier contenant les données prétraitées, sauvegardées pour les retrouver lorsqu'elles ne sont plus présentes en mémoire vive (dans le cadre de l'optimisation de l'utilisation de la RAM).
- Méthodes
 - open : ouverture d'un fichier BDF
 - save : enregistrement des données prétraitées dans un fichier BDF
 - close : fermeture du fichier
 - loadSignals : chargement des données des signaux du fichier BDF, dans des objets DataSignal
 - releaseAndSavePreprocessed : déchargement RAM des signaux (prétraités) et sauvegarde dans un fichier
 - loadPreprocessedSignal : chargement d'un signal particulier, qui a été déchargé auparavant
 - loadPreprocessedSignals : même chose, mais pour tous les signaux.

Classes de la couche Présentation

La couche Présentation contient toutes les fenêtres et tous les composants d'interface graphique de l'application. Elle communique avec la couche Métier via la classe `MainWindow`, qui représente la fenêtre principale mais qui est aussi l'intermédiaire entre la couche Métier et toutes les autres fenêtres et composants graphiques.



Classe `MainWindow`

La classe `MainWindow` a deux rôles :

- elle constitue la fenêtre principale de l'interface graphique de l'application
- elle est l'intermédiaire entre la couche Métier et tous les autres composants de l'interface graphique.

Classe `ScenarioGraph`

La classe `ScenarioGraph` dérive de la classe `QwtPlot` de la bibliothèque graphique `Qwt`. `QwtPlot` est un objet permettant de représenter des courbes sur un repère gradué. `ScenarioGraph` l'étend en y incluant deux courbes, représentées par la classe `Curve`, et un objet de la classe `Canvas Picker` qui sert à manipuler les courbes à l'aide de la souris. Elle contient aussi une grille, représentée par la classe `QwtPlotGrid`.

Classe `Curve`

La classe `Curve` dérive de la classe `QwtPlotCurve`. Les données sont représentées par ses attributs `xData` et `yData` (abscisses et ordonnées des points de la courbe). Des méthodes ont été apportées pour gérer l'ajout, l'édition et la suppression de points de la courbe.

Classe `RawSignalsWindow`

La classe `RawSignalsWindow` effectue la représentation graphique des signaux originaux (ou pré-traités). La classe `RawSignalsZoomer` permet de zoomer en dessinant un rectangle sur le graphique, `ScrollBar` permet de se déplacer selon l'axe des abscisses ou des ordonnées.

Classe `AnalysisWindow`

La classe `AnalysisWindow` est la fenêtre où l'on peut configurer les paramètres de l'analyse (choisir le signal à analyser, et sur quelle bande de fréquences) et visualiser les résultats.

La classe `SpectroPlot`, qui dérive de `QwtPlot`, permet l'affichage du spectrogramme du signal.

4.3 Quelques détails sur le développement, difficultés

Dans cette section, certains points importants du développement sont abordés, en soulignant les différentes difficultés rencontrées... le but n'étant pas de détailler tout le code du programme, ce qui serait de toute manière trop long.

4.3.1 Configuration

Comme nous l'avons vu précédemment, il est possible dans le premier onglet de l'application de configurer à la fois le choix du périphérique audio et la calibration du volume sonore.

Nous ne détaillerons pas le choix du périphérique audio, qui reste quelque chose d'assez trivial. En effet, avec la bibliothèque FMOD, on peut obtenir la liste des périphériques disponibles. Ceux-ci possèdent chacun un identifiant, et c'est en donnant à FMOD l'identifiant du périphérique que l'on paramètre le choix.

Nous allons donc voir comment s'effectue la calibration du volume sonore.

Calibration sonore, présentation

Dans FMOD Ex, dont nous verrons les principales fonctions dans la section "Module d'analyse", on peut régler le volume d'un canal à l'aide de cette fonction :

```
FMOD_Channel_SetVolume(pointeur vers le canal, valeur)
```

La valeur doit être comprise entre 0 et 1. C'est donc un pourcentage, et non une valeur réelle en décibels. Si le volume est fixé à 100 %, alors le canal jouera le son au maximum de la capacité sonore que Windows aura alloué au périphérique audio. En effet, le volume peut être réglé au sein même de Windows. Si on règle le volume de notre canal, dans le logiciel, à 100 %, mais que le son de Windows est fixé à 50%, alors le son ne sera pas joué au maximum de la capacité sonore du périphérique.

Par défaut, lorsque l'on crée une étape dans un scénario, le volume des sons de l'étape est réglé à 100 %. Or, sur la machine de déploiement, le son fixé par Windows est réglé au maximum et cela ne doit pas changer. Ainsi, le son est par défaut beaucoup trop fort, d'où la nécessité d'un réglage du volume général du logiciel.

Exemple :

- Le volume du système (Windows) est réglé à 100 %.
- Le volume du logiciel est réglé à 75 % (du volume du système).

Admettons que l'on ajoute une étape au scénario, et que le volume du son y soit paramétré à 50 %. Cela signifie que le volume est en réalité réglé à 37.5 % du volume du système.

De plus, on laisse la possibilité d'enregistrer la valeur réelle (en décibels, mesurée avec un sonomètre) correspondant à telle valeur de volume logiciel. Cela est fiable du moment que le volume du système ne change pas. Cela permet de voir instantanément à combien de décibels correspond le volume d'une étape que l'on est en train de paramétrer.

Mise en oeuvre

Pour commencer la calibration, il faut cliquer sur le bouton "Begin calibration". Le clic sur ce bouton déclenche un signal, `sigBeginCalibration()`, envoyé à l'objet `MainClass`. Ce dernier appelle la méthode `beginCalibration` de l'objet `SoundManager`, ce qui fait démarrer la lecture d'un bruit blanc afin de correctement tester le volume que l'on configure. Les sliders de réglage du volume, auparavant désactivés, s'activent à ce même moment.

Lorsque le slider de réglage du volume est bougé, cela déclenche le signal `sigTryCalibration(int value)`, envoyé à l'objet `MainClass`. Le paramètre `value`, entre 0 et 100, correspond au pourcentage souhaité pour le volume logiciel. `MainClass` appelle alors la méthode `tryCalibration` de `SoundManager`, avec le paramètre `(float)value / 100`, pour donner une valeur entre 0 et 1 à `FMOD`. `SoundManager` appelle ainsi la fonction de `FMOD` permettant de modifier la valeur du volume :

```
FMOD_Channel_SetVolume(_fmodChanNoise, volume)
```

où `_fmodChanNoise` est l'objet `FMOD` représentant le canal stéréo jouant le bruit.

Il est possible de régler le volume réel avec le slider dédié à cet effet. Aucune action n'est faite à ce moment (il faut attendre que la calibration soit terminée).

Lorsque la calibration est terminée, cela déclenche le signal `sigSetCalibration`, envoyé à l'objet `MainClass`. Cela appelle la méthode `setCalibration(int relative, double real)`. Le paramètre `relative` correspond au volume logiciel (en pourcentage) que l'on a choisi, et le paramètre `real` correspond au volume réel (en décibels) que l'on a mesuré et renseigné. De là, plusieurs étapes s'enchaînent :

1. `MainClass` appelle la méthode `setSoftwareVolFactor` de `SoundManager` en lui renseignant `(float)relative / 100` pour obtenir un pourcentage entre 0 et 1.
2. `MainClass` appelle la méthode `setRealVolFactor` de `MainWindow` en lui renseignant `real`, afin que les bons volumes soient affichés par l'interface graphique au moment du paramétrage du scénario.

Ensuite, il faut effectuer les bonnes conversions à chaque réglage d'étape. Par exemple, avec un volume logiciel de 75%, si l'utilisateur fixe le volume d'un canal dans une étape, à 50%, alors il faudra donner la valeur 0.335 à `FMOD` Ex. Pour cela, on crée la fonction `getSoftwareVolOf(int relative)` :

```
return (float)((relativeVol * _softwareVolFactor) / 100);
```

Dans notre exemple, `relativeVol` correspond à 50 et `_softwareVolFactor` correspond à 75.

Ainsi, on effectue chaque réglage de volume de cette manière :

```
FMOD_Channel_SetVolume(canal, getSoftwareVolOf(volume));
```

De même, au sein de l'interface graphique, lors du réglage de l'étape, on affiche le volume réel correspondant en le multipliant par `realVolFactor`.

Par exemple, si on a réglé le volume réel à 80 dB lors de la calibration, et l'étape à 50 %, alors le volume réel affiché sera 40 dB car `realVolFactor` sera égal à $80/100$.

Sauvegarde de la calibration

Lorsqu'aucune calibration n'a été faite sur cet ordinateur, cela est indiqué. Si elle a déjà été faite, les valeurs sont affichées.

Pour cela, on utilise un procédé permettant de sauvegarder les paramètres de l'application. Il s'agit d'utiliser un registre dans la base de registres de Windows. Actuellement, ils sont enregistrés à l'adresse HKEY_CURRENT_USER

Software

FDennig

Binaural Beats Tests. Le volume logiciel est enregistré dans la clé softwareLevelCalibration, et le volume réel dans la clé realLevelCalibration. La clé isCalibrationSet indique si une calibration a été faite ou non sur cet ordinateur.

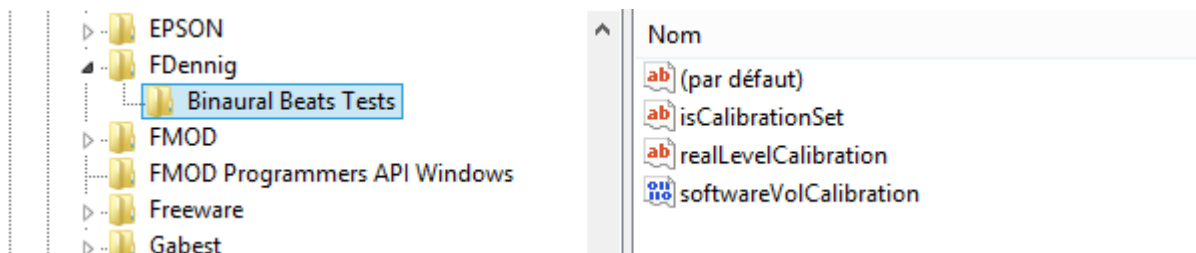


FIGURE 4.26 – Aperçu des paramètres de l'application dans la base de registres de Windows

Dans Qt 4.8, il est possible de sauvegarder et charger les paramètres avec la classe QSettings. MainClass possède l'objet QSettings _settings en tant qu'attribut, initialisé avec les paramètres "FDennig" et "Binaural Beat Tests".

4.3.2 Module d'édition de scénario

Dans cette partie je vais uniquement parler de la fonctionnalité d'affichage et d'édition graphique de scénario

Affichage (et mise à jour après édition textuelle)

L'affichage du scénario se fait par l'intermédiaire de la classe `ScenarioGraph`. Elle est dérivée de la classe `QwtPlot`, qui est un composant d'interface graphique permettant de représenter des courbes sur un repère.

Le repère est initialisé avec le temps sur l'axe des abscisses, et la fréquence sur l'axe des ordonnées. Il est quadrillé selon deux niveaux : un premier niveau représente les grands intervalles et les lignes sont donc dessinées en noir, le second niveau représente les intervalles moins importants (unité 1) et les lignes sont donc dessinées en gris. Cela se fait via un objet `QwtPlotGrid`.

Les courbes sont dessinées sur le repère grâce à la classe `Curve`, elle-même dérivée de la classe `QwtPlotCurve`. `Curve` est instanciée dans `ScenarioGraph` en deux objets : `leftFreqCurve` qui représente l'évolution de la fréquence en fonction du temps pour le canal gauche, et `rightFreqCurve` qui représente la même évolution mais pour le canal droit.

La mise à jour de l'affichage se fait par l'intermédiaire des méthodes `addStep`, `deleteStep`, `editStep`. Ces méthodes sont appelées par la classe parente, `MainWindow`, qui envoie les nouvelles données de fréquences et de durées des étapes du scénario.

Edition graphique

`ScenarioGraph` instancie la classe `CanvasPicker`, qui permet d'intercepter les actions de l'utilisateur afin de modifier les points des courbes sur le graphique. Cela se fait grâce à la méthode virtuelle `eventFilter`, héritée de la classe abstraite `QObject`.

Dans `eventFilter`, on traite les événements de l'utilisateur, que ce soit des clics de souris ou l'appui sur une touche du clavier. Dans notre cas, deux types d'événements nous intéressent particulièrement : le clic de souris, et le déplacement de la souris.

Code pour l'événement "clic de souris" :

```
case QEvent::MouseButtonPress:
{
    if (((QMouseEvent*)event)->button() == Qt::LeftButton)
    {
        select(((QMouseEvent*)event)->pos());
        _rightIsLastButtonClicked = false;
    }
    else if (((QMouseEvent*)event)->button() == Qt::RightButton)
    {
        addPoint();
        _rightIsLastButtonClicked = true;
    }
    return true;
}
```

Un clic gauche permet de sélectionner un point de la courbe. La sélection est gérée par la méthode `select`. Un clic droit permet d'ajouter un point à la courbe. L'ajout de point est géré par la méthode `addPoint` ;

Code pour l'événement "déplacement de la souris" :

```
case QEvent::MouseMove:
{
    if (!_rightIsLastButtonClicked)
        move(((QMouseEvent*)event)->pos());
    return true;
}
```

Le déplacement de la souris entraîne le possible déplacement de point de la courbe. Il faut premièrement que l'utilisateur ait effectué un clic gauche sur la courbe (pour le sélectionner). La méthode *move* teste si un point a bien été sélectionné, puis effectue le déplacement.

La sélection de point est gérée par la méthode *select*. On lui fournit une position en paramètre, et elle teste si cette position est à moins de 10 pixels de distance d'un point contenu dans la liste des points d'un objet *Curve* du graphique. Si c'est le cas, l'attribut *selectedPoint* prend la valeur de l'indice du point sélectionné, et l'attribut *selectedCurve* prend l'adresse de l'objet *Curve* sélectionné.

Le déplacement de point est géré par la méthode *move*. On lui fournit aussi une position en paramètre. On teste si l'attribut *selectedCurve* n'est pas égal à NULL. S'il ne vaut pas NULL, c'est qu'un point appartenant à une courbe a bien été sélectionné. On appelle donc la méthode *movePoint* de l'objet *Curve* sélectionné. Cette méthode *movePoint* permet de déplacer un point sur le graphique en suivant le quadrillage (aimantation sur l'axe des abscisses : il n'est pas possible de placer un point entre deux lignes verticales de la grille) et en respectant les incohérences temporelles (impossible de déplacer un point *i* et le mettre à une abscisse inférieure au point *i-1*).

Code de la méthode *movePoint* de la classe *Curve* :

```
void Curve::movePoint(int index, double x, double y)
{
    // On arrondit l'abscisse pour créer une aimantation sur chaque unité de temps
    int xValue = round(this->plot()->invTransform(this->xAxis(), x));
    // Suppression des incohérences temporelles (contraintes de précédence)
    if (index < _xDData.size() - 1)
    {
        if (xValue > _xDData.at(index + 1))
            xValue = _xDData.at(index + 1);
    }
    if (index > 0)
    {
        if (xValue < _xDData.at(index - 1))
            xValue = _xDData.at(index - 1);
    }
    else if (index == 0)
    {
        if (xValue < 0)
            xValue = 0;
    }
    _xDData[index] = xValue;
    _yData[index] = this->plot()->invTransform(this->yAxis(), y);
}
```


L'ajout de point est géré par la méthode *addPoint*. Elle appelle la méthode *appendPoint* des deux objets *Curve* pour ajouter le point à chaque courbe. En effet, les points sur les deux courbes ne sont pas placés au même endroit en ordonnée, mais le sont en abscisses : cela permet d'avoir des étapes cohérentes dans le temps.

Ces méthodes de gestion des événements, *movePoint* et *addPoint* émettent des signaux, reçus *ScenarioGraph*. Celui-ci peut ainsi mettre à jour le graphique pour éviter les incohérences. Par exemple, si un point est déplacé sur l'axe des abscisses, le point de l'autre courbe correspondant à la même étape doit le suivre sur cet axe (mais pas en ordonnée) pour garder la cohérence temporelle. De plus, *ScenarioGraph* envoie les nouvelles informations à *MainWindow*, qui elle-même envoie un signal reçu par *MainClass* qui va mettre à jour le contenu de l'objet *Scenario* concerné.

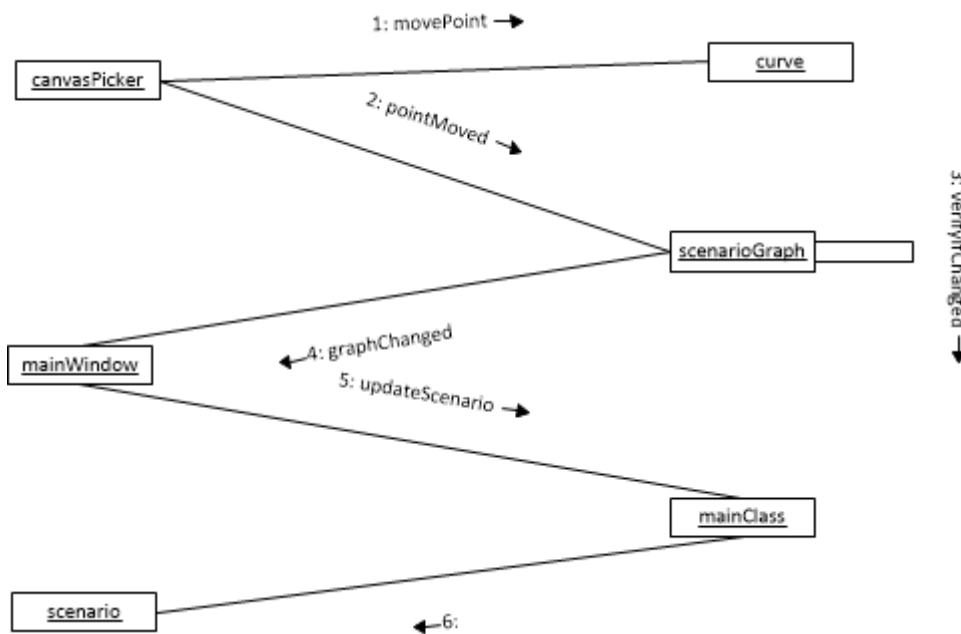


FIGURE 4.27 – Diagramme de communication pour le déplacement d'un point d'une courbe de la représentation graphique de scénario

4.3.3 Module audio, lecture de scénario

Dans cette partie sera expliqué la façon de jouer du son avec FMOD Ex, et la méthode utilisée pour faire varier le son en fonction du scénario lu.

Le son : la classe SoundManager

On utilise la bibliothèque FMOD Ex pour gérer le son. FMOD est disponible en C et en C++. Le problème avec la version C++ est qu'elle n'est pas compatible avec le compilateur MinGW, mais uniquement avec Visual C++. Il a donc fallu opter pour la version C, ce qui ne change pas grand chose, à part le style de programmation au sein de la classe SoundManager.

Avant toute chose, il faut initialiser le système audio de FMOD. On crée d'abord un objet FMOD_SYSTEM avec la fonction FMOD_System_Create. Puis on appelle ma fonction FMOD_System_Init en lui passant comme paramètres notre objet FMOD_SYSTEM, puis le nombre de canaux.

On récupère ensuite la liste des périphériques audio disponibles (pour la configuration) avec les fonctions FMOD_System_GetNumDrivers et FMOD_System_GetDriverInfo. Pour choisir le périphérique à utiliser, on utilise la fonction FMOD_SetDriver.

Pour générer les signaux sonores, il faut utiliser des oscillateurs (générateurs d'ondes). La forme d'onde par défaut est la sinusoïde : c'est ce que l'on veut. Pour créer un oscillateur, il existe la fonction FMOD_System_CreateDSPByType. Un DSP (Digital Signal Processing) est un élément de traitement du son : cela peut être un générateur ou un filtre. On peut créer soi-même ses DSP avec FMOD.

Pour ce qui est de la génération du bruit, le choix s'est porté sur la lecture en boucle d'un extrait de bruit blanc au format OGG.

Pour jouer un son ou un DSP, on utilise les fonctions FMOD_System_PlaySound ou FMOD_System_PlayDSP. Il faut leur fournir l'adresse des objets FMOD_CHANNEL correspondant aux canaux dans lesquels on veut jouer les sons.

Pour changer la fréquence d'un oscillateur, il suffit d'utiliser la fonction FMOD_DSP_SetParameter pour régler le paramètre "fréquence" à la valeur désirée. Une méthode a été créée dans SoundManager pour changer la fréquence à tout moment, cette méthode pouvant être appelée par une classe fille.

Le volume dispose lui aussi d'une méthode pour le régler à partir d'une classe fille. Il est normalisé suivant la valeur de volume logiciel fixée lors de la configuration.

La lecture de scénario : la classe ScenarioPlayer

ScenarioPlayer hérite de SoundManager.

L'objet ScenarioPlayer instancie un objet QTimer permettant de compter le temps passé. QTimer possède un signal nommé timeout(). Il est connecté au slot (réception de signal) verifyTime() de ScenarioPlayer. Dans ce slot, on incrémente les attributs elapsedTime et totalElapsedTime, et on modifie les paramètres de génération du son selon les paramètres de l'étape. De plus, si l'étape est terminée, on passe à la suivante en appelant la méthode runNextStep().

Pour modifier un paramètre, on se base sur :

- la valeur du paramètre au début de l'étape
- la valeur du paramètre à la fin de l'étape
- la valeur du paramètre à l'instant en cours
- l'instant en cours
- la durée de l'étape

Selon ces 5 facteurs, on modifie ou non le paramètre. Si modification il doit y avoir, le passage du début de l'étape à la fin de l'étape doit être linéaire.

Par exemple, si pour l'étape i , la fréquence du signal du canal gauche est 400 Hz au début de l'étape, et 500 Hz à la fin de l'étape, que la valeur actuelle est 400, que l'instant actuel est 1 après le top d'horloge de l'objet QTimer (entrée dans `verifyTime()`), et que la durée de l'étape est 2, alors la nouvelle valeur de fréquence sera 450 Hz.

Au niveau du code, cet exemple illustre la partie suivante du code de `verifyTime()` :

```
// Coefficient de pente
float coeff = _elapsedTime / (duration * 60000);

[...]

// Variation de la fréquence
if (_leftFreq < leftEndFreq)
    setLeftFreq(leftStartFreq + coef * (leftEndFreq - leftStartFreq));
```

Le coefficient de pente est égal à $\frac{\text{tempsécoulédansl'étape}}{\text{duréedel'étape}}$. On fait varier la fréquence selon ce coefficient de pente, et l'écart qu'il reste entre la fréquence actuelle et la fréquence à atteindre à la fin de l'étape.

4.3.4 Importation de fichiers BDF

L'acquisition des signaux recueillis par les électrodes est faite par le logiciel ActiView. Celui-ci enregistre les données au format BDF. Le fichier BDF induit doit être importé pour pré-traiter et analyser les signaux.

Pour gérer la lecture et l'écriture des fichiers BDF, il existe la bibliothèque EDFLib (EDF pour le format EDF, qui est l'ancienne version de BDF, mais EDFLib gère bien le format BDF). Ecrite en C et contenue dans seulement deux fichiers (un fichier header et un fichier source), elle fournit des structures de données contenant les informations d'en-tête pour le fichier et pour chaque signal, ainsi que des fonctions effectuant la lecture et l'écriture de signaux dans le fichier.

- la fonction *edfopen_file_readonly* permet d'ouvrir un fichier en lecture et remplit la structure *edf_hdr_struct* qui contient les données d'en-tête du fichier. *edf_hdr_struct* contient un tableau de structures *edf_param_struct* qui contiennent les données d'en-tête de chaque signal.
- les fonctions *edfread_digital_samples* et *edfread_physical_samples* permettent de lire les échantillons d'un signal en mettant les données dans un tableau. La première remplit le tableau avec des entiers, correspondant aux valeurs informatisées du signal. La seconde remplit le tableau avec des réels, correspondant aux valeurs réelles dans la bonne unité physique.

Cependant, je me suis rendu compte que ces fonctions étaient vraiment trop lentes : pour lire tous les signaux, le programme pouvait mettre plusieurs minutes. Pourtant, avec les logiciels connus de visualisation de signaux au format BDF, la vitesse d'ouverture est de l'ordre de la seconde !

J'ai donc analysé le code source des fonctions de EDFLib. Premièrement, les fonctions de lecture ne permettent de charger qu'un seul signal à la fois. Ce qui veut dire qu'il faut appeler ces fonctions autant de fois qu'il y a de signaux, et donc ouvrir autant de fois le fichier, sachant qu'un accès fichier est une procédure lente. Mais il y a un deuxième problème : l'algorithme de lecture du fichier. Le fichier est lu "petit à petit" avec de nombreux appels à la directive "fgetc" pour récupérer caractère par caractère les données du fichier.

Par exemple, si on a 68 signaux contenant chacun 781 824 échantillons, on va avoir 68 appels à la fonction "edfread_..._samples", puis 781 824 appels à fgetc pour chaque signal, soit environ **53 millions d'appels à fgetc !** C'est donc normal que le processus soit excessivement long !

J'ai donc écrit une fonction permettant de charger "d'une seule traite" tous les échantillons de tous les signaux, dans un très grand tableau. Pour accéder à un échantillon précis d'un certain signal, des opérations sont faites sur le pointeur afin d'avoir la bonne adresse mémoire.

Ne voulant pas écrire cette fonction dans EDFLib, mais dans le programme, j'ai dû extraire des structures de données de EDFLib. En effet, certaines structures utilisées dans les fonctions de lecture sont des améliorations des structures disponibles dans le header, et ne sont définies que dans le fichier source. Il a donc fallu les redéfinir dans le projet. Ces structures sont indispensables pour créer sa propre fonction de lecture, car elles contiennent notamment les coefficients permettant de convertir un bit en valeur physique, le nombre de octets par échantillon...

4.3.5 Pré-traitement

Le signal acquis via EEG n'est jamais parfait : les parasites et les mouvements du sujet sont autant de perturbations que de bruit dans le signal. C'est pourquoi, avant l'analyse, on effectue un pré-traitement. Pour l'équipe de chercheurs intéressée par le projet, cette étape suit un protocole précis, qu'elle voulait automatiser dans le logiciel. Voici ce protocole :

1. Réserver les 64 premières électrodes (V1-V64) pour l'EEG
2. Utiliser 4 électrodes supplémentaires (V65-V68) pour le prétraitement, et 1 pour les événements (triggers) :
 - V65 (VEOG) est placée sous l'oeil gauche : capte les mouvements oculaires verticaux
 - V66 (LHEOG) est placée à côté de l'oeil gauche : capte les mouvements oculaires horizontaux à gauche
 - V67 (RHEOG) est placée à côté de l'oeil droit : capte les mouvements oculaires horizontaux à droite
 - V68 (NOSE) est placée sur le nez : référence
 - V69 (Status) enregistre les triggers envoyés par le logiciel
3. Supprimer l'offset sur les 68 voies. Méthode : pour chaque signal, soustraire la moyenne à chaque échantillon
4. Soustraire le signal de référence NOSE aux signaux EEG : $V_i = V_i - V68$ avec $i=1..64$
5. Garder sur V65 uniquement les mouvements oculaires verticaux : $V65 = V65 - V1$
6. Réserver V66 aux mouvements oculaires horizontaux : $V66 = V66 - V67$
7. Appliquer un filtre passe-haut butterworth d'ordre 1 à 0.3 Hz, avec un algorithme de filtrage de type zero-phase

Le pré-traitement effectué permet d'obtenir un signal lisible, mais ne supprime pas tous les artefacts. Pour ce qui est du mouvement des yeux, celui-ci n'est en fait pas supprimé sur les signaux, mais ils sont tellement visibles, et non continus, qu'il est facile de les détecter visuellement et ainsi de ne pas les prendre en compte. De plus, on peut mettre les signaux en parallèle avec les signaux V65 et V66 qui correspondent aux mouvements des yeux.

Ce protocole est très précis, et si on l'applique tel quel, il ne sera pas possible d'ouvrir n'importe quel fichier de données EEG, car le logiciel n'arrivera pas à effectuer le pré-traitement, ne trouvant pas les signaux V65 à V69. C'est pourquoi a été ajoutée la possibilité de choisir les signaux de pré-traitement, et d'activer ou non tout ou partie du pré-traitement.

La classe `PreprocessingChannels` regroupe les signaux de pré-traitement, et indique si telle partie du processus du pré-traitement est activée ou non.

Algorithme de pré-traitement (hors triggers)

```
Pour ISIGNAL de 1 à DATA.SIZE Faire
```

```
    // Calcul de la moyenne
```

```
    MOYENNE <- 0.0
```

```
    Pour IVAL de 1 à DATA[ISIGNAL].NBSAMPLES Faire
```

```
        MOYENNE <- MOYENNE + DATA[ISIGNAL][IVAL]
```

```
    Fin Pour
```

```
    MOYENNE <- MOYENNE / DATA[ISIGNAL].NBSAMPLES
```

```
Pour IVAL de 1 à DATA[ISIGNAL].NBSAMPLES Faire
```

```
    // Correction d'offset
```

```
    DATA[ISIGNAL][IVAL] <- DATA[ISIGNAL][IVAL] - MOYENNE
```

```
    // Référence Nez
```

```
    Si PREPROCESSINGCHANNELS.isNoseEnabled() Alors
```

```
        Si IVAL < PREPROCESSINGCHANNELS.NOSE.NBSAMPLES Alors
```

```
            DATA[ISIGNAL][IVAL] <- DATA[ISIGNAL][IVAL]
```

```
                - PREPROCESSINGCHANNELS.NOSE[IVAL]
```

```
    Fin Si
```

```
    // Mouvements oculaires horizontaux
```

```
    Si ISIGNAL = 1 Alors
```

```
        Si PREPROCESSINGCHANNELS.isVerticalEnabled() Alors
```

```
            Si IVAL < PREPROCESSINGCHANNELS.VERTICAL.NBSAMPLES Alors
```

```
                PREPROCESSINGCHANNELS.VERTICAL[IVAL] <- PREPROCESSINGCHANNELS.VERTICAL[IVAL]
```

```
                    - DATA[ISIGNAL][IVAL]
```

```
        Fin Si
```

```
    Fin Si
```

```
    // Mouvements oculaires verticaux
```

```
    Si ISIGNAL = 1 Alors
```

```
        Si PREPROCESSINGCHANNELS.isHorizontalEnabled() Alors
```

```
            Si IVAL < PREPROCESSINGCHANNELS.HORIZONTALLEFT.NBSAMPLES Alors
```

```
                PREPROCESSINGCHANNELS.HORIZONTALLEFT[IVAL] <- PREPROCESSINGCHANNELS.HORIZONTALLEFT[IVAL]
```

```
                    - PREPROCESSINGCHANNELS.HORIZONTALRIGHT[IVAL]
```

```
        Fin Si
```

```
    Fin Si
```

```
Fin Pour
```

```
[ Suite de la boucle, VOIR PARTIE FILTRAGE ]
```

```
Fin Pour
```

Filtrage

L'équipe souhaite que le signal passe dans un filtre passe-haut Butterworth d'ordre 1 à 0.3 Hz, avec un algorithme de filtrage zero-phase.

Avant d'expliquer ce que cela signifie, essayons de faire simplement ce qui est dit, avec le logiciel R. Le package *signal* contient des fonctions identiques à celles de Matlab pour le traitement du signal, dont une fonction permettant de modéliser un filtre Butterworth, et une permettant d'effectuer un filtrage zero-phase.

Nous allons tenter de filtrer un signal de 256 points échantillonné à 8 KHz composé de deux sinusoides, l'une de fréquence 50 Hz, l'autre de fréquence 256 Hz. Le but du filtrage est de ne laisser passer que la sinusoïde de fréquence 256 Hz.

Exemple de filtrage avec R

On définit une fréquence d'échantillonnage. Dans notre exemple : 8000 Hz.

```
fe <- 8000
```

On définit le nombre de points. Nous allons en mettre 256.

```
n <- 256
```

On crée l'axe des temps en fonction du nombre de points et de la fréquence d'échantillonnage.

```
t <- seq(1, n, 1) / fe
```

On génère deux sinusoides, l'une de fréquence 50 Hz, l'autre de fréquence 256 Hz. On crée un signal composé des deux sinusoides.

```
f1 <- 50  
f2 <- 256  
sinus1 <- sin(2 * pi * f1 * t)  
sinus2 <- sin(2 * pi * f2 * t)  
somme <- sinus1 + sinus2
```

On modélise le filtre (passe-haut, butterworth, ordre 1, fréquence de coupure 250 Hz).

```
fc <- 250  
cutoff <- fc / (fe / 2)  
filtre <- butter(1, cutoff, "high")
```

On applique le filtre sur le signal avec un algorithme zero-phase (filtfilt de Matlab).

```
f <- filtfilt(filtre, somme)
```

On affiche séparément les deux sinusoides, puis le signal (composé des deux sinusoides).

```
plot(t, sinus1, "l", col="green") #Première sinusoïde  
lines(t, sinus2, col="blue") #Deuxième sinusoïde  
x11() # permet d'ouvrir une nouvelle fenêtre  
plot(t, somme, "l", col="red") #Somme
```

On affiche enfin le signal filtré.

```
plot(t, f, "l") #Résultat filtrage
```

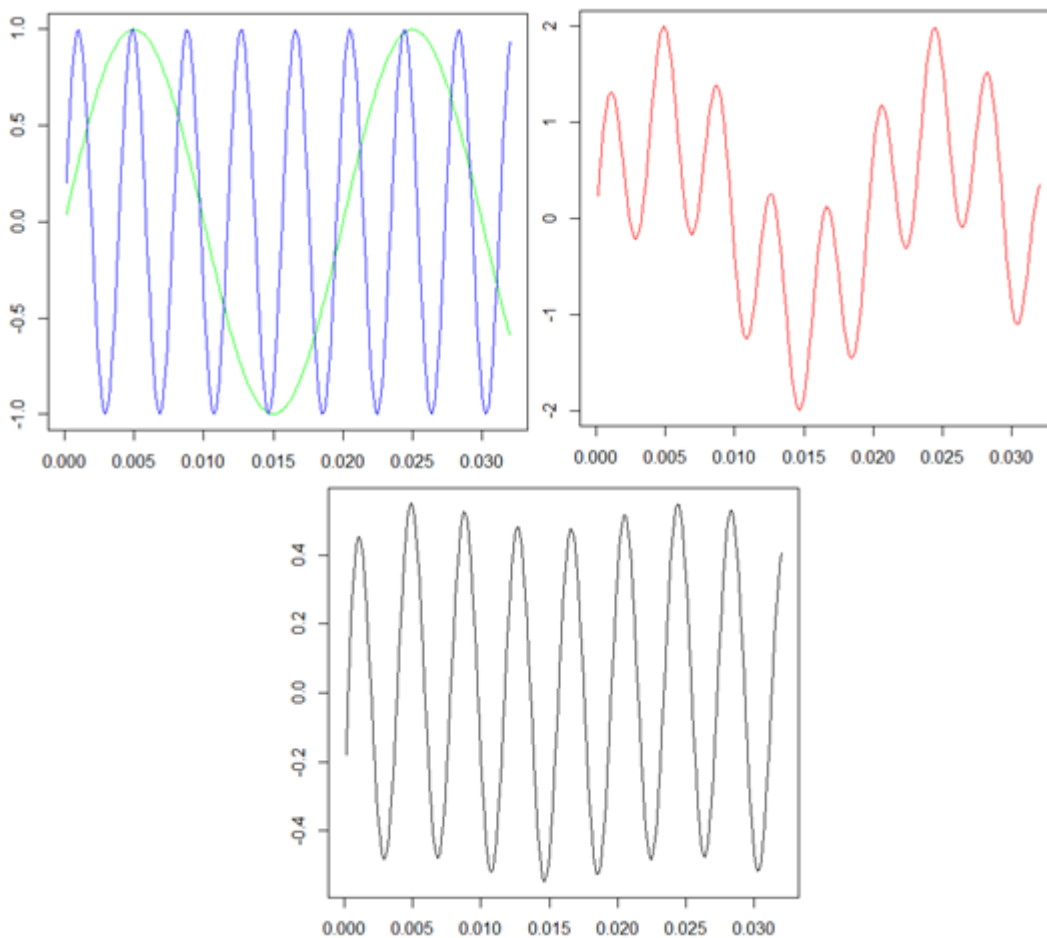


FIGURE 4.28 – Les deux sinusoïdes superposées, le signal composé des deux sinusoïdes, puis le résultat du filtrage.

On remarque sur la figure ci-dessus que le filtrage a quasi totalement supprimé la composante 50 Hz du signal. Il ne reste plus que la composante 256 Hz, légèrement altérée en amplitude.

Explications et algorithmes

Nous avons montré, avec le logiciel R, comment effectuer un filtrage zero-phase Butterworth d'ordre 1 à 250 Hz. Mais le logiciel développé dans ce projet n'utilise ni R, ni Matlab. Il doit effectuer le filtrage lui-même. Pour cela, il faut savoir comment cela fonctionne.

Pour illustrer, dans le monde réel, lorsqu'un son est émis, il rencontre de nombreux obstacles, qui ont pour conséquence de modifier les signaux sonores. Chacun de ces obstacles peut être considéré comme un filtre. Par exemple dans une salle présentant une mauvaise acoustique, certaines fréquences vont être amplifiées. Certains matériaux vont absorber plus ou moins bien telle ou telle fréquence, et renvoyer les signaux avec un spectre fréquentiel différent de celui envoyé à l'origine. Dès lors que cela arrive, on peut dire que le signal a été filtré.

Certains filtres coupent les fréquences basses, et laissent passer les fréquences hautes, selon une valeur de coupure définie. On les appelle filtres passe-haut (ou coupe-bas). Voici ce que serait un filtre passe-haut parfait (f_c est la fréquence de coupure, $f_s/2$ est la fréquence maximale, équivalente à la moitié de la fréquence d'échantillonnage) :

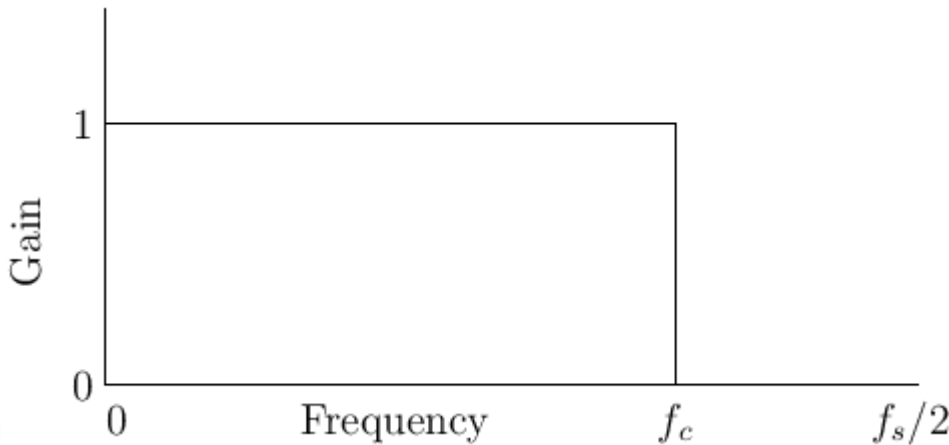


FIGURE 4.29 – Filtre passe-haut parfait

La fonction Gain représente la réponse en amplitude du filtre : si la réponse pour une certaine fréquence est égale à 1, alors après filtrage, cette fréquence ne sera pas affectée (ni amplifiée, ni atténuée). En revanche, si elle est égale à 0, la fréquence sera totalement rejetée.

Dans la réalité, un filtre possède une réponse en amplitude plus complexe : il est difficile de filtrer un signal sans affecter un minimum les fréquences. Selon le filtre utilisé, cela peut même les affecter énormément.

Par exemple, le filtre de Tchebychev de type 1 affecte les fréquences inférieures à la fréquence de coupure, tandis que le filtre de Tchebychev de type 2 affecte les fréquences supérieures.

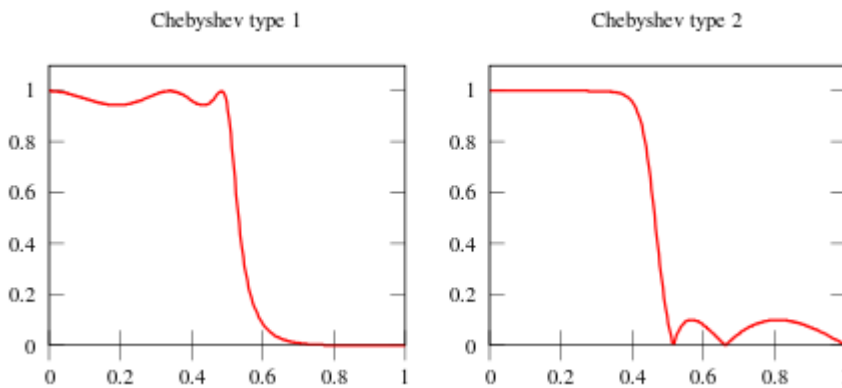


FIGURE 4.30 – Comparaison de la réponse en amplitude du filtre de Chebyshev de type 1 avec celui de type 2

Le filtre utilisé par l'équipe est Butterworth. Il n'affecte presque pas les fréquences : sa réponse en amplitude est quasi plate. En revanche, dans sa version la plus simple (ordre 1), la coupure est moins nette. Il faut utiliser un ordre élevé pour obtenir une coupure très nette. En théorie, utiliser un filtre Butterworth d'ordre très élevé est la solution idéale. Dans la pratique, surtout en électronique, cela est compliqué à réaliser, car un ordre élevé implique une combinaison de plusieurs filtres d'ordres 1 et 2.

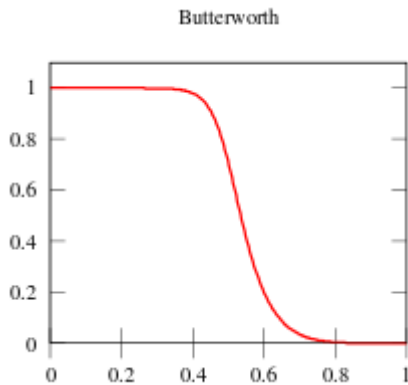


FIGURE 4.31 – Réponse en amplitude du filtre Butterworth (ordre 1)

Un filtre est caractérisé par deux vecteurs a et b . La taille de chacun de ces vecteurs est égale à $\text{ordre}+1$. L'algorithme de filtrage utilise les coefficients de ces vecteurs pour effectuer l'opération de filtrage.

Voici la formule mathématique (utilisée par Matlab dans la fonction `filter`) permettant d'effectuer un filtrage, à partir des vecteurs a et b . x est le signal d'entrée, y le signal de sortie. o est l'ordre du filtre, n est le nombre d'échantillons du signal. Les indices commencent à 0.

$$y_0 = b_0 \times x_0 \quad (4.1)$$

$$\forall i \in [0; n - 1], y_i = b_0 \times x_i + b_1 \times x_{i-1} + \dots + b_o \times x_{i-o} - a_1 \times y_{i-1} - \dots - a_o \times y_{i-o} \quad (4.2)$$

D'où l'algorithme suivant :

```
// Partie initiale
Y[0] <- B[0] * X[0]
Pour I de 1 à ORD Faire
  Y[I] <- 0
  Pour J de 0 à I Faire
    Y[I] <- Y[I] + B[J] * X[I-J]
  Pour J de 0 à I - 1 Faire
    Y[I] <- Y[I] - A[J+1] * Y[I-J-1]
Fin Pour

// Partie restante
Pour I de ORD + 1 à N - 1 Faire
  Y[I] = 0
  Pour J de 0 à ORD Faire
    Y[I] <- Y[I] + B[J] * X[I-J]
  Pour J de 0 à ORD - 1 Faire
    Y[I] <- Y[I] - A[J+1] * Y[I-J-1]
Fin Pour
```

Le protocole de pré-traitement demande un filtrage zero-phase. En effet, le fait de filtrer un signal crée une distorsion de phase, c'est-à-dire que le signal a été en quelque sorte décalé dans le temps. Le filtrage zero-phase permet de corriger ce phénomène, comme montré sur l'illustration ci-dessous (Mathworks) :

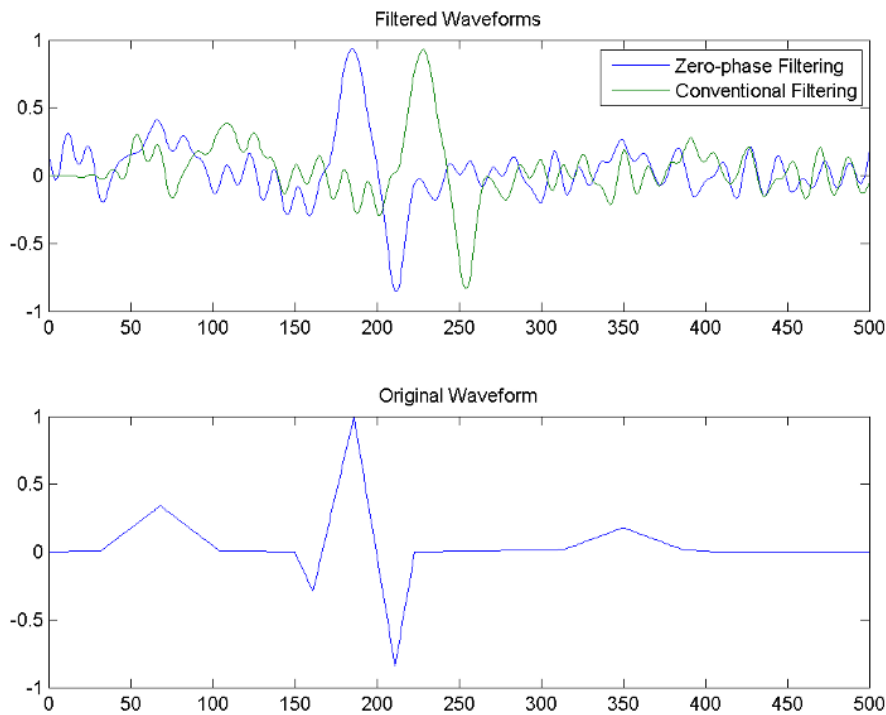


FIGURE 4.32 – Différences entre filtrage classique et filtrage zero-phase

Sur cette figure, on remarque aisément que le signal filtré avec l’algorithme de filtrage classique est décalé dans le temps par rapport au signal original, alors que ce n’est pas le cas pour celui filtré avec l’algorithme zero-phase.

Le principe de cet algorithme est de filtrer une première fois le signal, puis inverser le signal, le filtrer, et ré-inverser. Voici l’algorithme utilisé dans le logiciel :

```
// Premier filtrage
filter(ORD, A, B, N, X, Y)

// Inversion
Pour I de 0 à N - 1 Faire
    X[I] <- Y[N-I-1]

// Deuxième filtrage
filter(ORD, A, B, X, Y)

// Ré-inversion
Pour I de 0 à N - 1 Faire
    X[I] <- Y[N-I-1]
Pour I de 0 à N - 1 Faire
    Y[I] <- X[I]
```

Pour ce qui est de la modélisation du filtre, c'est-à-dire les vecteurs a et b , l'algorithme utilisé pour Butterworth est compliqué. Vu que le protocole de pré-traitement est très précis (ordre 1, passe-haut, fréquence de coupure 0.3 Hz), il est possible d'entrer directement ces vecteurs dans le code.

Voici comment les vecteurs ont été obtenus avec R.

On désire une fréquence de coupure de 0.3 Hz. Pour le filtre Butterworth, la fréquence de coupure correspond à la fréquence où la réponse en amplitude est exactement égale à $\sqrt{1/2}$.

On utilise la fonction `butter` pour obtenir les coefficients du filtre. Cette fonction prend en paramètre une fréquence de coupure normalisée entre 0 et 1 selon la loi de Nyquist et Shannon.

Cette loi affirme que la fréquence d'échantillonnage d'un signal doit être supérieure ou égale au double de la fréquence maximale du signal¹. Si cette loi n'est pas respectée, des fréquences peuvent être erronées et il y a dénaturation du signal.

La fréquence d'échantillonnage pour les signaux EEG enregistrés par l'équipe est de 512 Hz. La fréquence de coupure normalisée est donc égale à $0.3 / (512 / 2) = 0.001171875$.

Pour obtenir les coefficients du filtre, on exécute donc la commande suivante dans R (en ayant préalablement chargé le package `signal`) :

```
butter(1, 0.3 / 256, "high")
```

On obtient les valeurs suivantes :

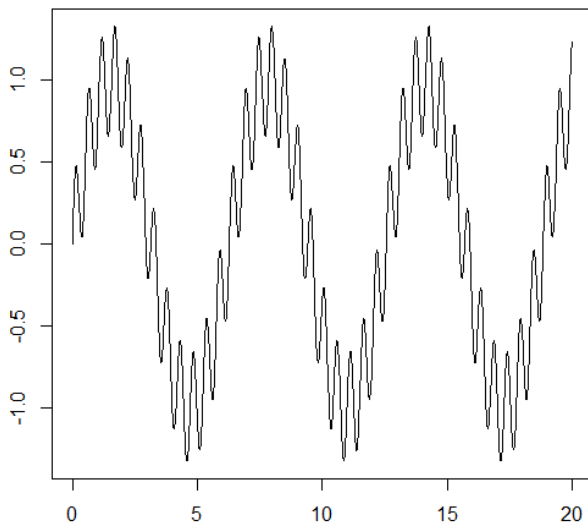
$$b = \begin{pmatrix} 0.9981626 & -0.9981626 \end{pmatrix} \quad (4.3)$$

$$a = \begin{pmatrix} 1.0000000 & -0.9963252 \end{pmatrix} \quad (4.4)$$

Inconvénients de ce filtrage

Il n'est pas recommandé d'utiliser cette technique de filtrage zero-phase pour des filtres IIR (tels que Butterworth) avec une fréquence de coupure très basse. Le risque est que le début et la fin du signal, sur une certaine durée (en général plutôt courte), les valeurs grimpent vers l'infini.

Par exemple, voici un signal échantillonné à 5 Hz, composé de deux sinusoïdes, l'une de fréquence 0.42 Hz, l'autre de fréquence 0.71 Hz.



1. C'est pourquoi la fréquence d'échantillonnage pour un signal audio doit être supérieure ou égale à 40 KHz, l'oreille humaine ne pouvant entendre que les fréquences inférieures ou égales à 20 KHz.

On désire supprimer la composante de plus basse fréquence (0.42 Hz). L'idée est donc d'appliquer un filtre passe-haut Butterworth ordre 1, de fréquence 0.6 Hz (fréquence normalisée : 0.24). Voici ce que cela donne après le filtrage zero-phase :

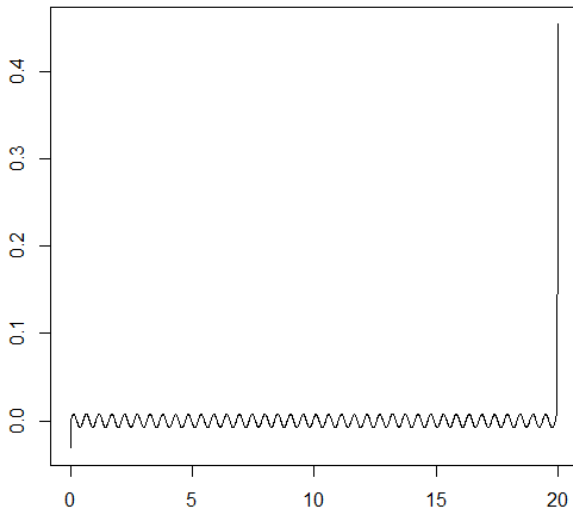


FIGURE 4.33 – Illustration du risque du filtrage zero-phase pour une fréquence de coupure trop basse

Le signal part vers l'infini en négatif au début, et en positif à la fin. Cela nuit à la bonne lecture du signal, dans une moindre mesure (il suffit de zoomer), et fausse les données sur un certain intervalle au début, et à la fin, ce qui peut être regrettable si on a besoin de ces échantillons.

Matlab, lorsque ce problème se pose, ignore purement le filtre au début et à la fin du signal. C'est donc ce que le logiciel fait. En effet, le problème se pose forcément : le protocole de pré-traitement fixe la fréquence de coupure à 0.3 Hz, ce qui est très faible !

4.3.6 Analyse temps-fréquence

Pour vérifier l'hypothèse émise au début de ce rapport, qui consiste à penser que le cerveau émettrait des ondes cérébrales de fréquence sensiblement identique à la fréquence des battements binauraux écoutés par le sujet, il faut effectuer une analyse temps-fréquence. Cela consiste à analyser le spectre des fréquences du signal, au cours du temps.

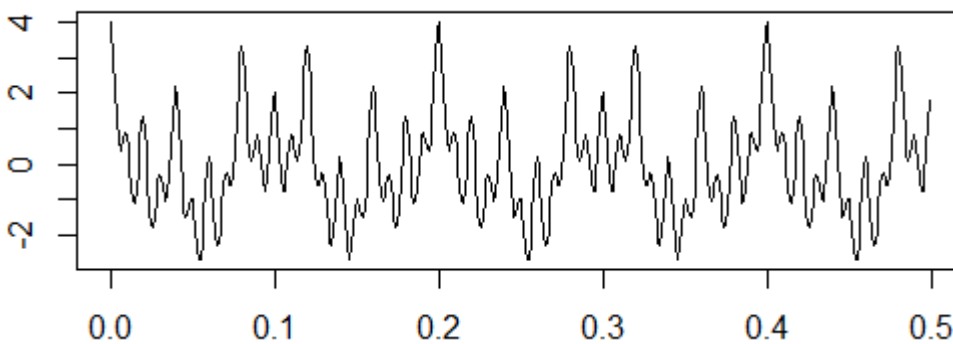
Rappels sur la transformée de Fourier et ses limites

On connaît bien l'analyse spectrale de Fourier. Mais cette méthode ne comporte pas de dimension temporelle.

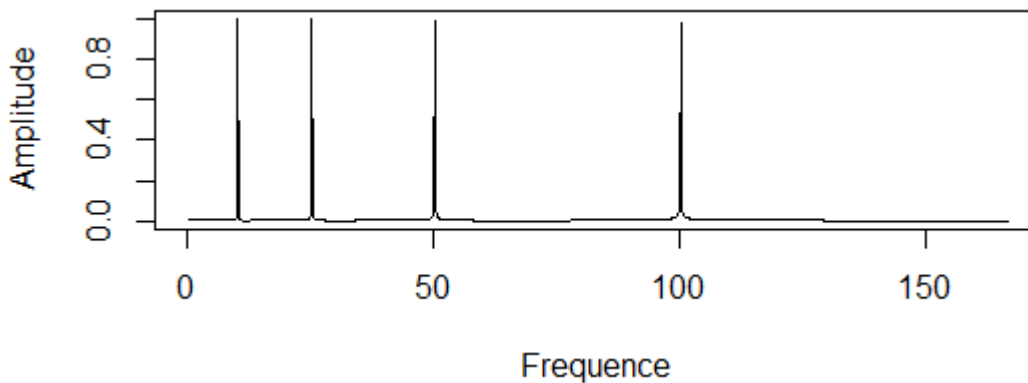
Prenons deux exemples pour illustrer ce "problème".

Soit le signal $x(t) = \cos(2\pi 10t) + \cos(2\pi 25t) + \cos(2\pi 50t) + \cos(2\pi 100t)$.

Ce signal est dit stationnaire, car il comporte les mêmes fréquences à chaque instant.

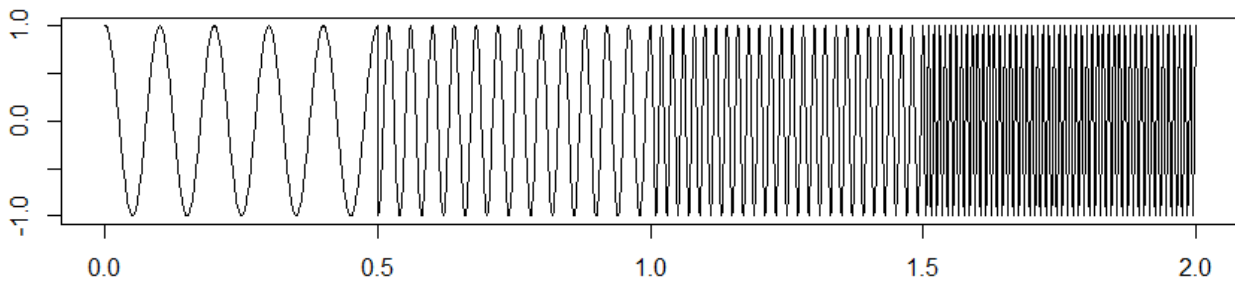


Voici son spectre fréquentiel obtenu grâce à la transformée de Fourier :



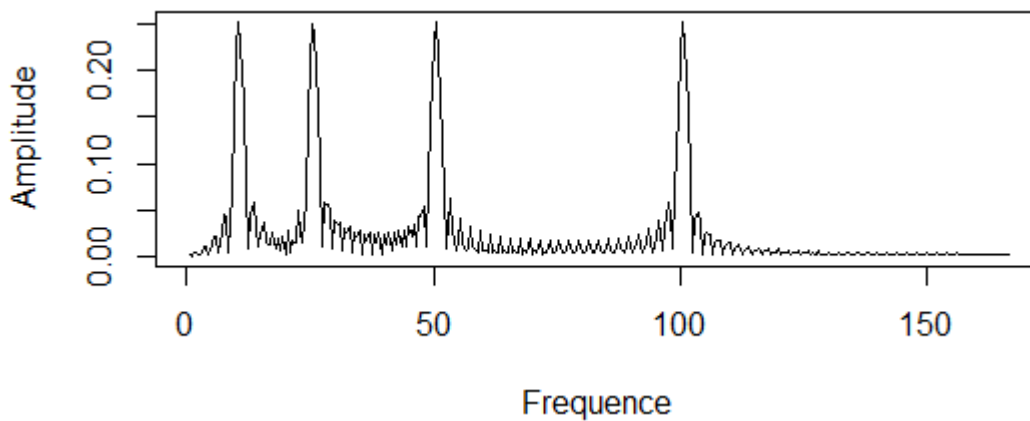
On retrouve bien les 4 fréquences du signal, à savoir 10, 25, 50 et 100 Hz.

Maintenant, prenons un signal dont la fréquence change constamment au cours du temps : un signal dit non-stationnaire.



La fréquence du signal est de 10 Hz sur la première portion, puis 25 Hz sur la deuxième, 50 Hz sur la troisième, et enfin 100 Hz sur la dernière portion.

Voici le spectre que l'on obtient si l'on effectue la transformée de Fourier de ce signal :



Les légères variations présentes entre les pics sont dues aux changements soudains de fréquence.

Selon le spectre ci-dessus, les fréquences 10, 25, 50, et 100 Hz sont présentes dans le signal. C'est vrai. Mais cela ne veut pas forcément qu'elles sont présentes tout le temps (contrairement au cas du premier signal, stationnaire, qui comportait ces 4 fréquences pendant toute la durée du signal, à chaque instant).

Si on regarde le spectre, il est impossible de savoir à quel moment une fréquence apparaît dans le signal.

La transformée de Fourier indique simplement les composantes fréquentielles présentes dans le signal. C'est tout.

C'est pourquoi la transformée de Fourier ne suffit pas dans ce cas : on a alors recours à l'analyse temps-fréquence.

Transformée de Fourier à Court-Terme (STFT) et ses problèmes

La transformée de Fourier dévoile les fréquences existant dans le signal. Mais pour savoir quelle fréquence est présente à quel moment, elle ne fonctionne pas avec les signaux non-stationnaires.

Or, en théorie, si on considère une portion du signal assez petite pour qu'elle soit stationnaire, alors on peut effectuer sa transformée de Fourier et savoir quelles fréquences sont présentes dans cette portion. C'est ce que fait STFT (Short-Term Fourier Transform) : elle divise le signal en de nombreux segments, assez petits pour être considérés comme stationnaires. Le signal est découpé selon une fonction fenêtre (w). On multiplie la fonction fenêtre par le signal (pour obtenir seulement la portion considérée) et on effectue la transformée de Fourier sur cette portion.

Ainsi, STFT permet de trouver quelles fréquences sont présentes dans le signal, et à quel moment.

Cependant, cette méthode présente un très gros défaut, qui repose sur le Principe d'Incertitude d'Heisenberg. Il dit qu'on ne peut pas savoir exactement quelles composantes fréquentielles existent à quels instants précis, mais qu'on peut savoir les intervalles de temps pour lesquels certaines bandes de fréquences existent. Il s'agit d'un problème de résolution, en rapport avec la taille de la fonction fenêtre.

La transformée de Fourier a une résolution fréquentielle parfaite : elle restitue parfaitement les fréquences. Cela est dû au fait que sa fonction fenêtre, $\exp(j\omega t)$, est de taille infinie.

A l'inverse, la fenêtre de STFT est finie (car si on utilise une fenêtre de taille infinie, cela revient à faire une transformée de Fourier classique et on ne sait pas quand telle fréquence apparaît dans le signal).

Pour résumer, pour avoir des morceaux de signal stationnaires, il faut avoir une taille de fenêtre suffisamment petite. Mais plus la taille de la fenêtre est petite, moins la résolution fréquentielle est bonne, et à l'inverse, plus la taille de la fenêtre est grande, et moins la résolution temporelle est bonne.

- Fenêtre étroite : bonne résolution temporelle mais mauvaise résolution fréquentielle
- Fenêtre large : bonne résolution fréquentielle mais mauvaise résolution temporelle

Le choix de la taille de la fenêtre est crucial, et malheureusement très difficile à faire. On est jamais sûr d'avoir des informations fréquentielles correctes, et si on veut en être sûr, on sacrifie la connaissance de leur apparition dans le temps.

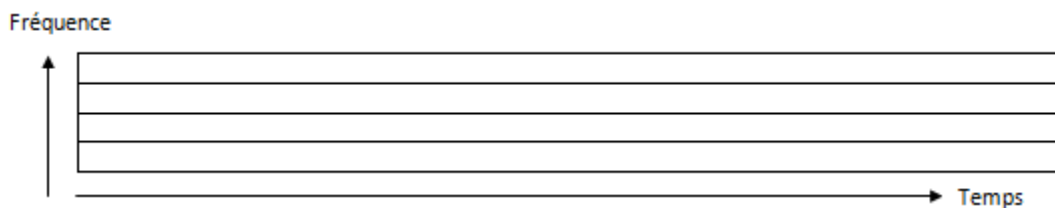


FIGURE 4.34 – Résolution de la transformée de Fourier. Les fréquences y sont parfaitement restituées, mais la notion de temps est inexistante.

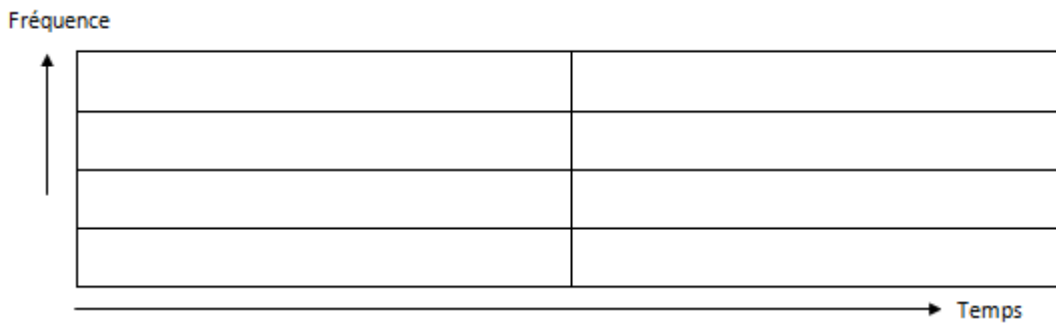


FIGURE 4.35 – Problème du paramétrage de la taille de la fenêtre pour STFT (1). STFT avec une très grande largeur de fenêtre. On peut avoir une excellente résolution fréquentielle, mais on ne saura pas **quand** telle fréquence est présente, juste si elle se trouve dans la 1ère ou la 2ème moitié du signal...

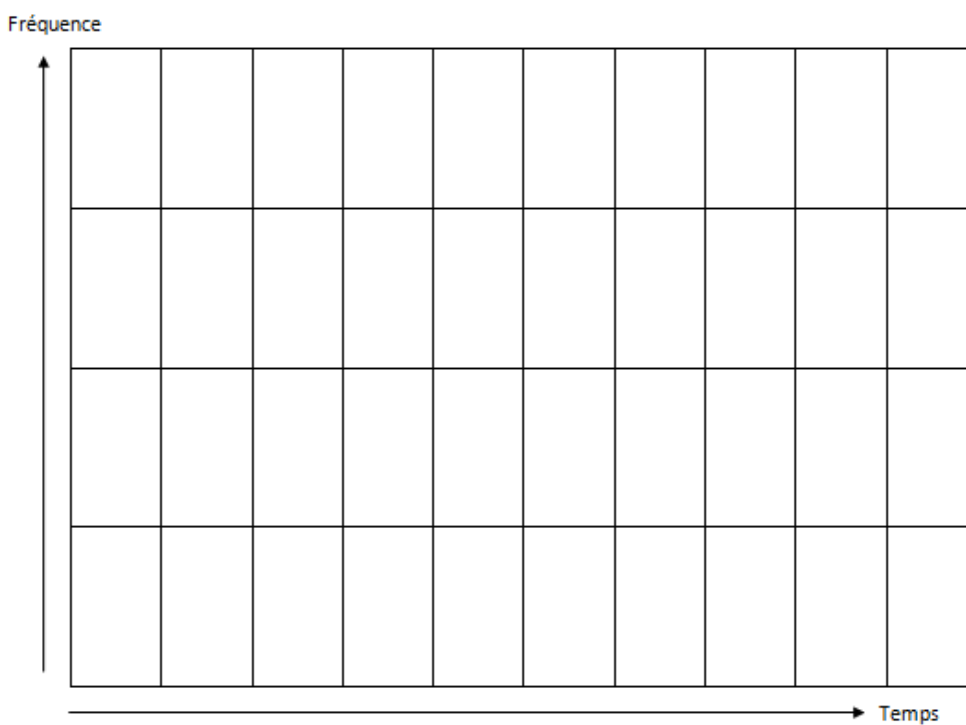


FIGURE 4.36 – Problème du paramétrage de la taille de la fenêtre pour STFT (2). Ici, la largeur de la fenêtre est plus courte. Le temps y est donc mieux représenté. Cependant, il existe maintenant une plus grande incertitude dans la fréquence.

Transformée en ondelettes : une méthode d'analyse multirésolution

Ce problème de résolution est corrigé par ce qu'on appelle l'Analyse Multi-Résolution (MRA). Voici le principe :

- La plupart des signaux comportent des composantes fréquentielles élevées sur de courtes durées : la MRA a donc une bonne résolution temporelle et une résolution fréquentielle moindre pour les hautes fréquences.
- A l'inverse, les composantes fréquentielles basses sont en général présentes sur de longues durées : la MRA a donc une bonne résolution fréquentielle et une résolution temporelle moindre pour les basses fréquences.

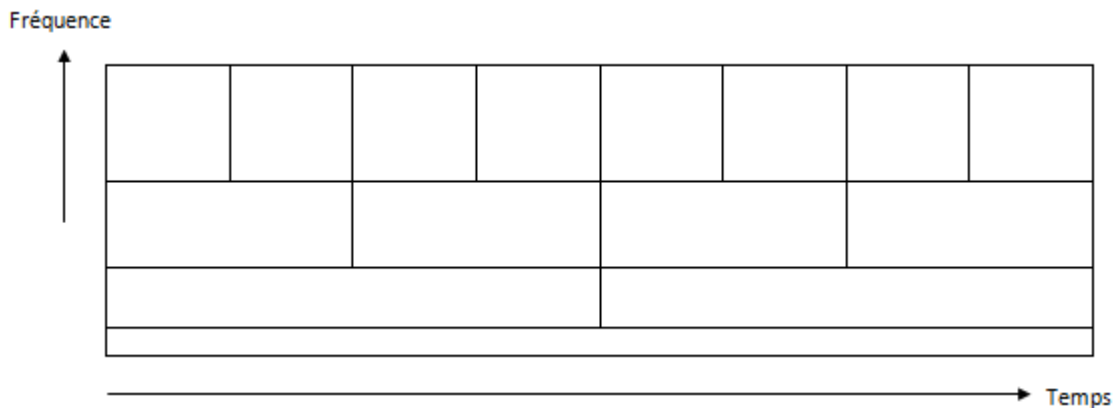


FIGURE 4.37 – Illustration du principe de multirésolution

La transformée en ondelettes est une méthode d'analyse multirésolution.

Cette transformée se fait de la même manière que STFT : on multiplie le signal par une fonction fenêtre et on effectue la transformée de Fourier sur chaque morceau du signal qui aura été découpé. Un point diffère :

- La fonction fenêtre, qu'on appelle "ondelette" n'a pas de taille fixe tout au long du processus de transformation : elle s'adapte selon la bande de fréquences concernée.

Voici le fonctionnement, en résumé, en essayant de faire le plus simple possible :

1. On choisit ce qu'on appelle une ondelette mère. Une ondelette est une onde de durée finie qu'on utilise comme fonction fenêtre. L'ondelette mère peut être dilatée ou contractée pour qu'elle soit plus ou moins longue. Les versions modifiées de l'ondelette mère constitueront les fonctions fenêtres pour la transformation du signal.
2. On fixe les valeurs d'échelles et de translations. Une échelle correspond à une bande de fréquences, plus ou moins grande selon l'intervalle entre chaque échelle. L'échelle la plus basse correspond aux fréquences les plus hautes et inversement. Une translation correspond à un intervalle de temps. Ce seront les paramètres de la transformée.
3. On se place à l'échelle $s=1$ (s comme "scale").
4. On multiplie le signal par la fonction d'ondelette pour $s=1$, et on intègre sur tous les temps. Pour normaliser l'énergie (sinon elle n'est pas la même à chaque échelle), on multiplie le tout par $1/\sqrt{s}$.
5. On décale l'ondelette vers la droite.
6. On répète les opérations 4 et 5 jusqu'à atteindre la fin du signal.
7. On incrémente s

8. On effectue les opérations 4, 5, 6 et 7 jusqu'à atteindre la valeur maximale de s (les fréquences les plus basses).

Remarque : plus s est grand, plus la fenêtre est large. En effet, une grande échelle correspond à des fréquences basses, et il faut donc une bonne résolution fréquentielle, ce qui implique une résolution temporelle faible.

Application de la transformée en ondelettes dans le programme

Il existe une bibliothèque C++ qui permet de calculer la transformée en ondelettes d'un signal. Il s'agit de CwtLib++. Son utilisation a permis de ne pas réinventer la roue en implémentant une version bancaire de la méthode, et de terminer le projet dans les temps. Cependant il a fallu la modifier pour pouvoir s'en servir de la façon la plus optimale qui soit.

Voici la structure de CwtLib++ :

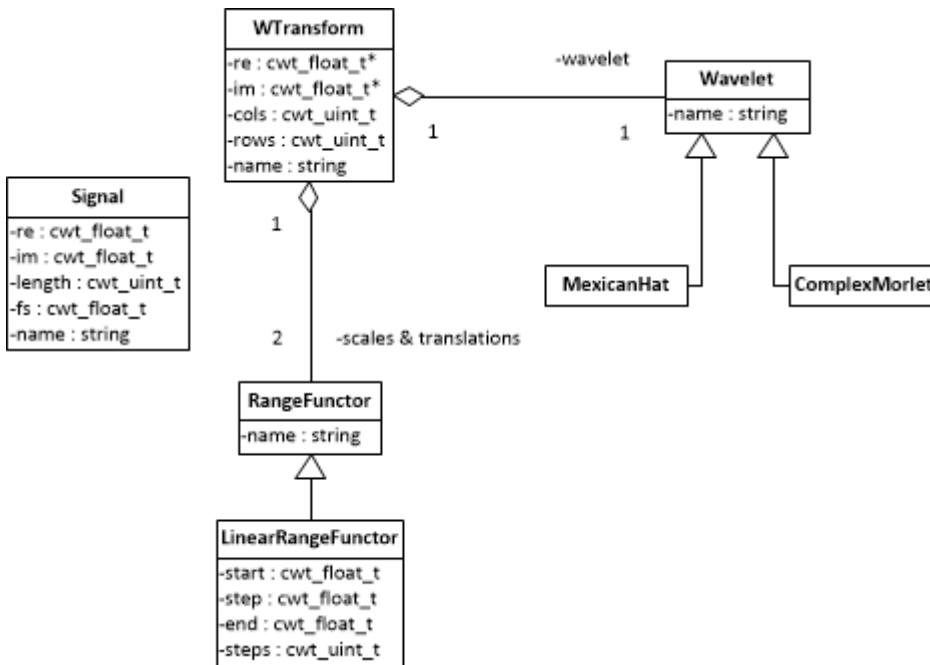


FIGURE 4.38 – Diagramme de classes de CwtLib++

Pour calculer la transformée en ondelettes d'un signal, il faut :

1. créer un objet Signal et lui fournir toutes les données du signal à analyser
2. créer un objet RangeFuncion pour les échelles
3. créer un objet RangeFuncion pour les translations
4. appeler la fonction statique cwtft de CWTalgorithm, qui crée un objet WTransform et le retourne. cwtft permet de calculer la transformée en ondelettes avec un algorithme très efficace, basé sur la transformée de Fourier rapide (FFT). Il faut lui fournir les objets représentant les échelles et les translations, ainsi que l'ondelette mère que l'on veut utiliser.

Le résultat du calcul est stocké dans l'objet WTransform. Par exemple si on veut récupérer l'amplitude à l'échelle $s = 5$ et à $\tau = 2$, alors on appelle la méthode publique mag de WTransform, avec les paramètres 5 et 2.

Le problème avec cette implémentation est que les échelles que l'on peut générer sont forcément linéaires (classe `LinearRangeFunc`) : les bandes de fréquences pour les fréquences basses sont aussi grandes que celles pour les fréquences hautes. Ce qu'il faudrait, c'est une échelle non linéaire. Pour l'ondelette mère utilisée dans le programme, l'ondelette de Morlet complexe², le meilleur facteur d'échelle est $5/2\pi$, car c'est la fréquence centrale de Morlet (dans `CwtLib++`, la fréquence centrale est enregistrée en tant qu'attribut, et est fixée à 0.8, ce qui est une valeur approchée de $5/2\pi$). La formule pour créer les échelles a à partir des fréquences f est donc $a = (5/2\pi)/f$.

Pour avoir de telles échelles, il a fallu créer une nouvelle classe dérivant de `RangeFunc`. Je l'ai appelée `ManualRangeFunc`. En effet, `LinearRangeFunc` ne prend que 3 paramètres à sa construction : la valeur de départ, la valeur de fin, l'intervalle entre chaque valeur. Les échelles sont générées automatiquement. Avec `ManualRangeFunc`, on lui fournit directement un tableau de valeurs. Comme cela, on peut calculer soi-même ses échelles, avec la méthode que l'on veut.

Dans le programme, j'ai donc créé un tableau d'échelles, calculées à partir d'un tableau de fréquences, avec le facteur $5/2\pi$, et j'ai instancié un objet `ManualRangeFunc` avec ces échelles.

Ensuite, il faut visualiser les résultats. La représentation graphique pour l'analyse temps-fréquence s'appelle spectrogramme ou scalogramme. Elle permet de représenter en abscisses le temps, en ordonnées la fréquence, et en couleur l'amplitude.

Pour faire cela, il existe dans la bibliothèque `Qwt` le widget `QwtPlotSpectrogram`. Il faut lui fournir une matrice de données, sous forme d'un objet `QwtMatrixRasterData`. On remplit cet objet avec la méthode `setValueMatrix`, en lui fournissant en paramètres un objet `QVector` contenant toutes les données à la suite, et un entier ayant pour valeur le nombre de colonnes de la matrice (il s'agit du nombre de translations).

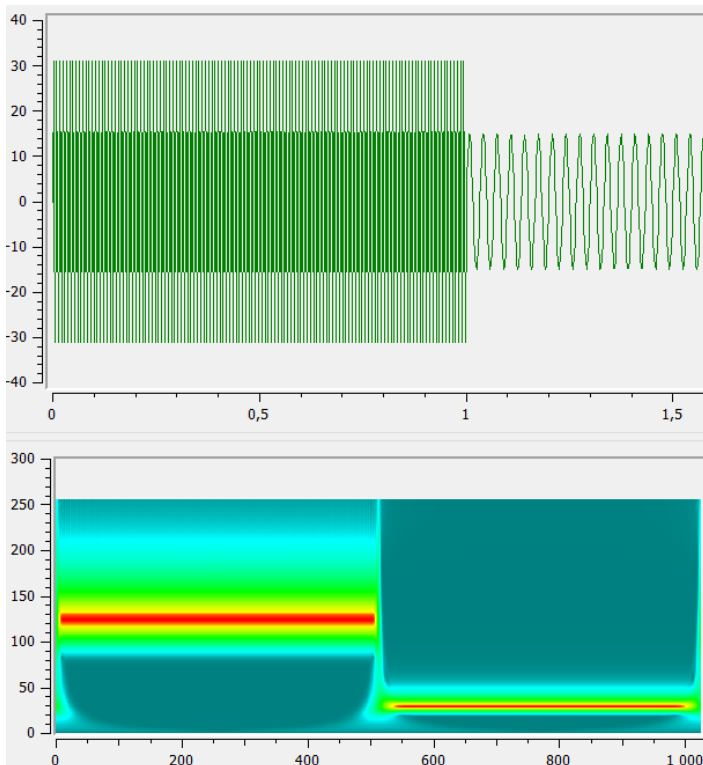


FIGURE 4.39 – Exemple de spectrogramme réalisé avec `Qt`, `CwtLib++` et `Qwt`, pour un signal non-stationnaire avec une première portion à 128 Hz puis une deuxième à 30 Hz. En haut : la représentation du signal. En bas : son spectrogramme.

2. Morlet car c'est l'ondelette mère conseillée pour l'analyse EEG. Exemple : C.Avanzo, V.Tarantino, P.Bisiacchi, G.Sparacino, A Wavelet Methodology for EEG Time-Frequency Analysis In a Time Discrimination Task

4.3.7 Gestion de la mémoire vive

Plus le développement avançait, plus la quantité de RAM utilisée par le programme grandissait. Cela ne veut pas dire qu'il y avait des fuites de mémoire. Il s'agit d'un problème bien plus complexe que de simples oublis de désallocations. Le problème, c'est le volume des données.

Dans un des fichiers BDF que l'équipe m'a fournis pour mes essais, il y a **68 signaux**, échantillonnés à **512 Hz** (donc 512 échantillons par seconde). Chaque signal a une durée **25 minutes** et 2.7 secondes. Cela fait $512 \times 25.45 \times 60 = 781824$ échantillons. Au total, si on veut les 68 signaux, cela fait plus de 53 millions d'échantillons à charger (53 164 032)! Sachant que le type de données *double* est codé sur 8 octets, cela représente **405.6 Mo**! ($53164032/1048176$ car 1 Mo = 1048176 octets).

Certes, cette quantité de mémoire reste possible à charger si la machine cible possède au moins 1 Go de mémoire vive, ce qui est monnaie courante. Mais les algorithmes de filtrage et d'analyse ont besoin de réserver de l'espace mémoire pour copier les données, et pour stocker les résultats!

C'est pour cela que j'ai décidé de procéder signal par signal lors de l'analyse, ne pas s'occuper de plusieurs signaux en même temps. En effet, un signal représente 6 Mo. L'algorithme de transformée en ondelettes effectue 2 allocations de même taille que le signal, pour la partie réelle et la partie imaginaire de chaque échantillon. Pour un signal, cela représente 12 Mo. Le résultat de l'analyse est, de plus, stocké dans une matrice dont le nombre de lignes peut être très grand. Il s'agit du nombre d'échelles que l'on veut pour la transformée. Le nombre d'échelles doit être assez grand, pour permettre une bonne résolution fréquentielle. Par exemple, pour 25 échelles, on va obtenir une matrice de 25×781824 coefficients. Cela représente 149 Mo. Soit un total de 161 Mo en comptant les données. Mais si on avait voulu analyser tous les signaux (et les garder en mémoire pour les visualiser), cela aurait représenté plus de 10 Go!

Gestion de la mémoire utilisée par le filtrage

Dans l'algorithme de pré-traitement, on filtre chaque signal. Le filtrage nécessite :

- les données initiales du signal
- l'allocation d'un tableau pour le signal filtré.

Cela fait doubler la quantité de mémoire utilisée par le signal. A un instant t , si la durée du signal ne dépasse pas 30 minutes, il ne devrait pas y avoir de problème si la mémoire disponible est supérieure ou égale à 1 Go. Il faut juste que ce "doublement" de la mémoire ne soit pas appliqué aux 68 signaux sans nettoyage préalable.

Voici la procédure :

1. On alloue le tableau pour le signal filtré.
2. On filtre. A ce moment-là, la quantité de RAM utilisée par le signal est doublée.
3. On remplit le tableau des données initiales du signal avec les données du signal filtré
4. On n'oublie pas de désallouer le tableau du signal filtré.

Il s'agit là juste de bon sens, ce n'est pas vraiment une optimisation, mais il fallait souligner le fait que le moindre oubli dans un programme peut tout faire basculer.

Gestion de la mémoire lors de la visualisation des signaux

Pour représenter un signal sur l'axe des temps, il faut fournir à `QwtPlotCurve` un tableau contenant les abscisses, et un tableau contenant les valeurs (ordonnées). L'utilité du tableau d'abscisses consiste à permettre d'affecter les bonnes valeurs de temps sur l'axe : sinon on voit les numéros des échantillons, non le temps. Il faut donc allouer un tableau de même taille que le signal (pour chaque échantillon, on divise le numéro de l'échantillon par la fréquence d'échantillon puis par 60 pour avoir le temps en minutes). Cela double ainsi la quantité de mémoire utilisée.

C'est pourquoi, pour économiser la mémoire, j'ai décidé de ne pas garder tous les signaux en mémoire pour afficher la représentation graphique. Avant lancement de la fenêtre de visualisation, les données sont sauvegardées dans un fichier texte, puis le bon signal est chargé. Aussi, dès que l'on veut la représentation graphique d'un autre signal, les données du premier signal sont déchargées, et celles du nouveau signal sont chargées. Lorsque l'utilisateur choisit un signal à visualiser, un signal Qt est envoyé par l'objet RawSignalsGraph, reçu par MainClass, qui appelle la méthode loadPreProcessedSignal de BiosemiFileManager qui permet de charger le signal demandé. Ensuite MainClass envoie un signal Qt avec les données. RawSignalsGraph les reçoit. Le module d'analyse est aussi capable de recevoir ce signal Qt, car la procédure de gestion de mémoire pour les données des signaux est la même.

Enfin, le choix de la méthode d'affectation des données à la courbe (QwtPlotCurve) est crucial : la méthode setSamples effectue une copie des données (ce qui double encore la RAM), tandis que setRawSamples initialise ses points en adressant directement les données via un pointeur. Le choix s'est donc porté sur la méthode setRawSamples.

4.4 Tests réalisés

Différents types de tests ont été menés durant ce projet.

1. Des **tests unitaires**. Il s'agit de s'assurer que certaines fonctions indépendantes de la solution s'exécutent correctement. Il est souvent nécessaire d'écrire du code pour mener de tels tests. Ces tests ont été menés pendant le développement : au début pour l'ouverture et la sauvegarde de scénario, puis plus tard pour le filtrage et la transformée en ondelettes.
2. Des **tests de validation fonctionnelle**. Le but est vérifier que les fonctionnalités énoncées dans le Cahier de Spécifications sont respectées. Il s'agit de tests effectués à la souris, sur des modules spécifiques du logiciel.
3. Un **test de validation solution**. Il s'agit d'un test "grandeur nature" : le logiciel est testé dans les mêmes conditions qu'en production. Lors de cette phase, les différents cas d'utilisation sont passés en revue.
4. Des **tests de validation robustesse**. La validation robustesse permet de trouver, s'il y en a, des problèmes de stabilité du programme. Cela concerne surtout, dans notre cas, la gestion de la mémoire vive.
5. On peut aussi parler de **recette fonctionnelle**, ou UAT (User Acceptance Tests) : il s'agit de la période où le client valide le logiciel selon la satisfaction des besoins spécifiés.

4.4.1 Tests unitaires

Ouverture de scénario

Des tests ont été effectués pour tester la bonne exécution de la fonction d'ouverture de scénario.

On considère qu'un test est validé si tous les attributs du scénario sont restitués et présents dans un objet Scenario après lecture du fichier XML. Chaque étape du scénario doit aussi être restituée dans un objet Step, contenu dans l'objet Scenario.

Le détail de ces tests ne sera pas présenté ici.

Sauvegarde de scénario

De même que pour l'ouverture de scénario, la sauvegarde a été testée.

On considère qu'un test est validé si tous les attributs du scénario et de ses étapes sont correctement écrits dans le fichier XML, avec la bonne indentation.

Le détail de ces tests ne sera pas présenté ici.

Fonction de filtrage

Les fonctions de filtrage filter et filtfilt ont été testées dans un projet Qt séparé du projet du logiciel.

Pour valider les tests, le logiciel R a été utilisé : en effectuant le même filtrage avec R et avec les fonctions testées, il a été possible de comparer les deux résultats.

Premier test

Fréquence d'échantillonnage	8000 Hz
Nombre d'échantillons	256
Formule du signal	$\sin(2\pi 100t) + \sin(2\pi 800t)$
Fréquence de coupure pour le filtrage	700 Hz
Test validé	oui

TABLE 4.1 – Test Filtrage 1

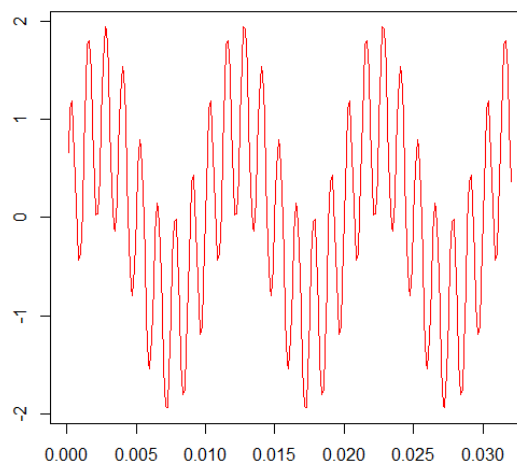


FIGURE 4.40 – Test Filtrage 1 : signal initial

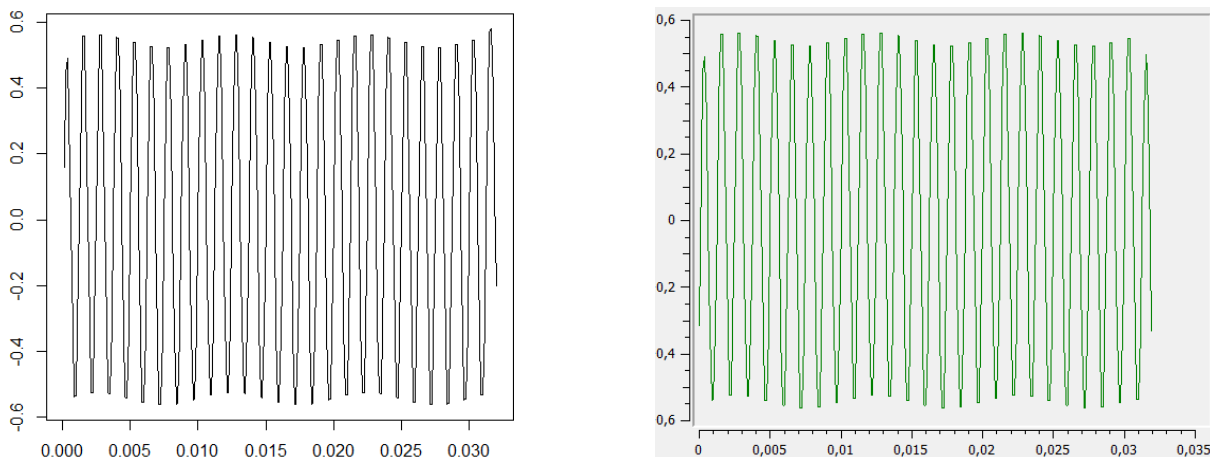


FIGURE 4.41 – Test Filtrage 1 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.

Deuxième test

Fréquence d'échantillonnage	8000 Hz
Nombre d'échantillons	256
Formule du signal	$\sin(2\pi 50t) + \sin(2\pi 256t)$
Fréquence de coupure pour le filtrage	250 Hz
Test validé	oui

TABLE 4.2 – Test Filtrage 2

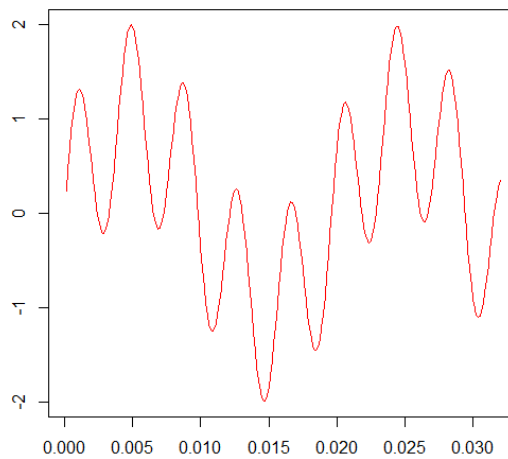


FIGURE 4.42 – Test Filtrage 2 : signal initial

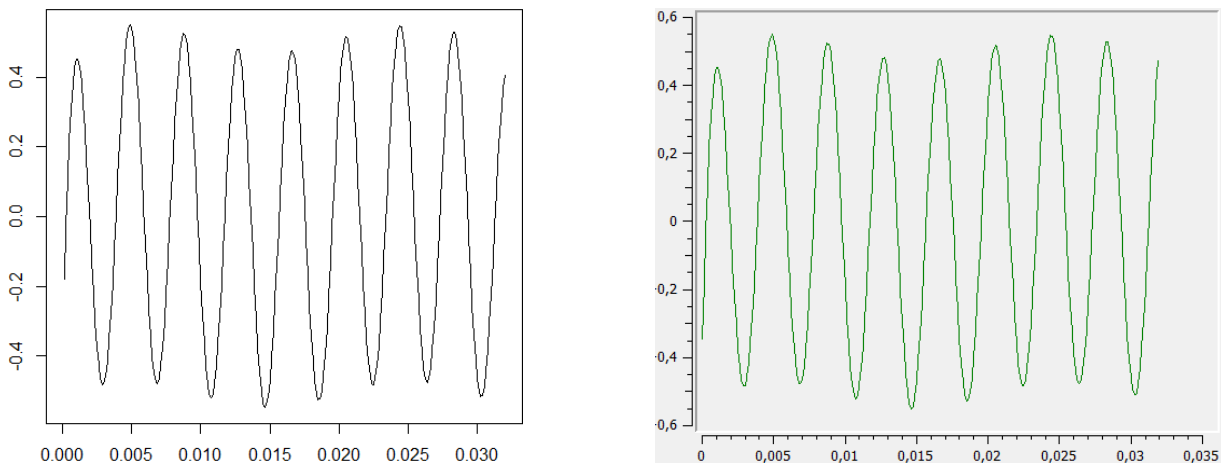


FIGURE 4.43 – Test Filtrage 2 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.

Troisième test

Fréquence d'échantillonnage	512 Hz
Nombre d'échantillons	4096
Formule du signal	$\sin(2\pi 0.1t) + \sin(2\pi 5t)$
Fréquence de coupure pour le filtrage	0.5 Hz
Test validé	oui

TABLE 4.3 – Test Filtrage 2

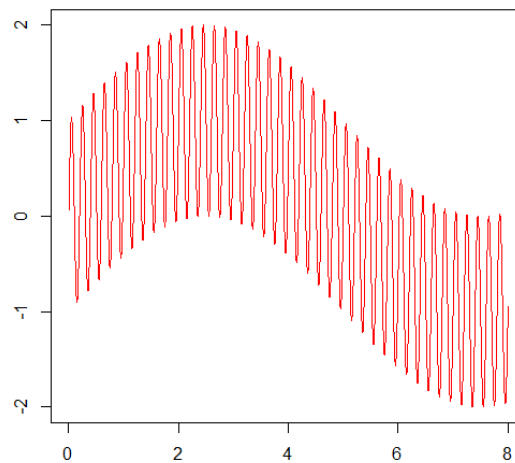


FIGURE 4.44 – Test Filtrage 3 : signal initial

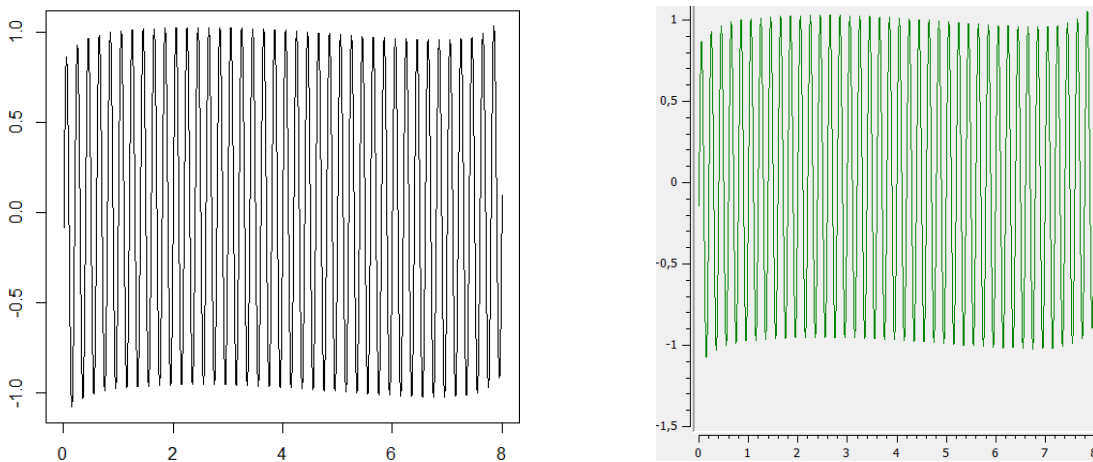


FIGURE 4.45 – Test Filtrage 3 : résultats. A gauche, résultat témoin (avec R). A droite, résultat testé.

Transformée en ondelettes

Comme pour le filtrage, l'implémentation de la transformée en ondelettes a été testée dans un projet Qt séparé.

Pour valider les tests, il a suffi de vérifier :

1. que les échelles ayant l'intensité les plus élevées du spectrogramme correspondent bien à la fréquence des composantes fréquentielles du signal initial
2. que les correspondances entre intensité d'échelle élevée et présence de composante fréquentielle dans le signal soient cohérentes dans le temps.

Premier test

Fréquence d'échantillonnage	512 Hz
Nombre d'échantillons	256
Formule du signal	De 0 à 127 : $30 \sin(2\pi 100t)$ De 128 à 255 : $15 \sin(2\pi 30t)$
Fenêtre fréquentielle de la transformée	[1 – 256]
Test validé	oui

TABLE 4.4 – Test Transformée en Ondelettes 1

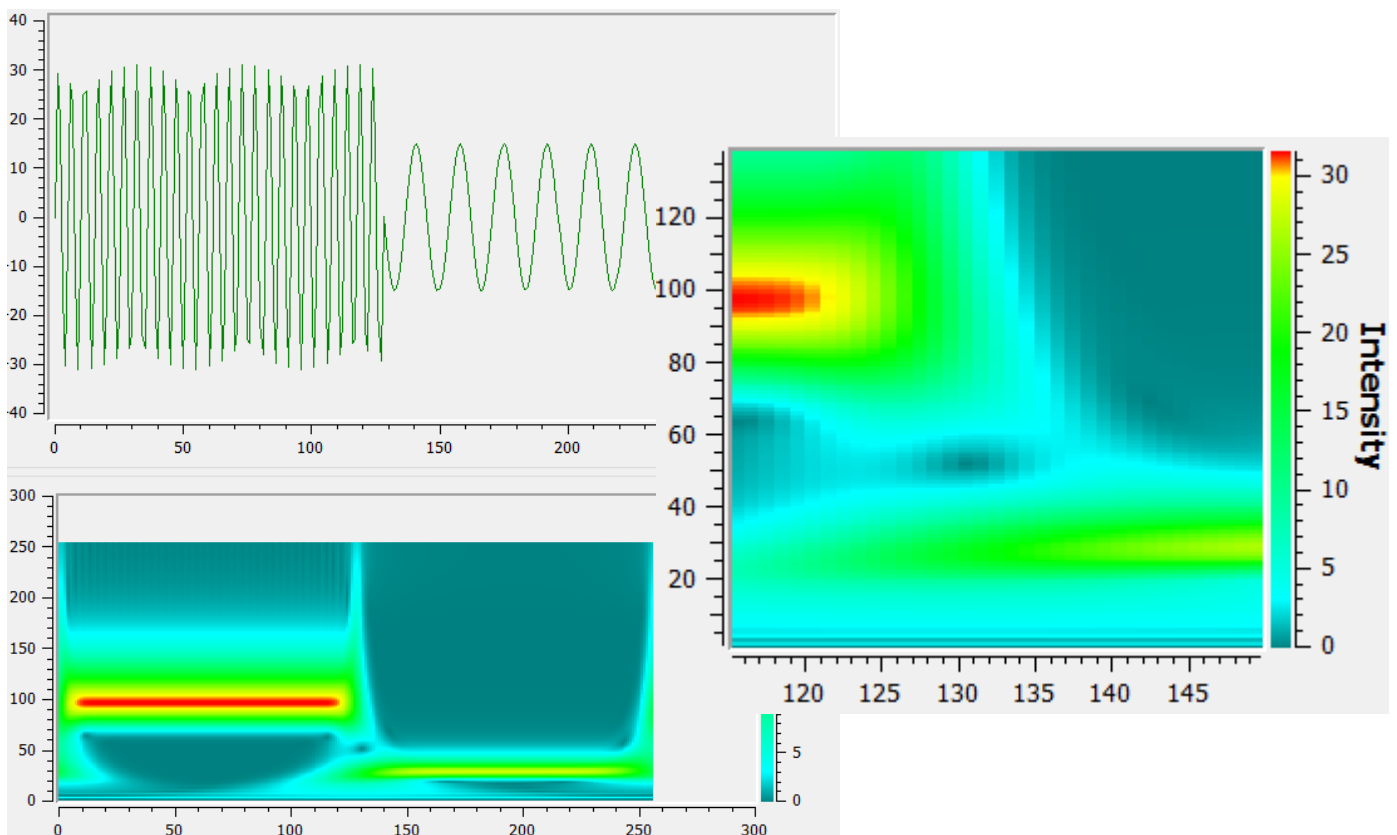


FIGURE 4.46 – Test Transformée en Ondelettes 1

Deuxième test

Fréquence d'échantillonnage	512 Hz
Nombre d'échantillons	4096
Formule du signal	De 0 à 2047 : $30 \sin(2\pi 23t)$ De 2048 à 4095 : $15 \sin(2\pi 5t)$
Fenêtre fréquentielle de la transformée	[1 – 100]
Test validé	oui

TABLE 4.5 – Test Transformée en Ondelettes 2

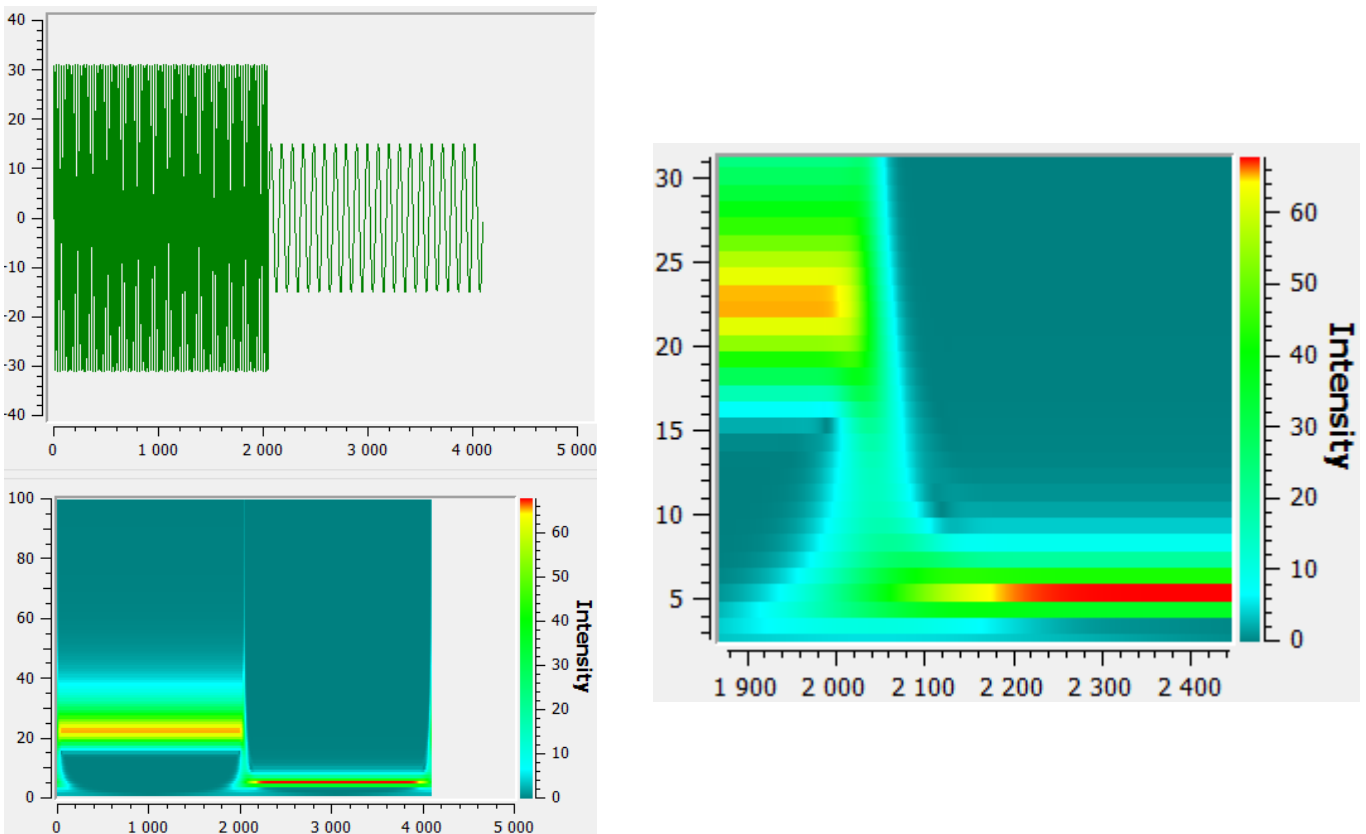


FIGURE 4.47 – Test Transformée en Ondelettes 2

Troisième test

Fréquence d'échantillonnage	512 Hz
Nombre d'échantillons	4096
Formule du signal	De 0 à 2047 : $\sin(2\pi 23t) + 5 \sin(2\pi 200t)$ De 2048 à 4095 : $\sin(2\pi 10t) + 4 \sin(2\pi 120t)$
Fenêtre fréquentielle de la transformée	[1 – 256]
Test validé	oui

TABLE 4.6 – Test Transformée en Ondelettes 3

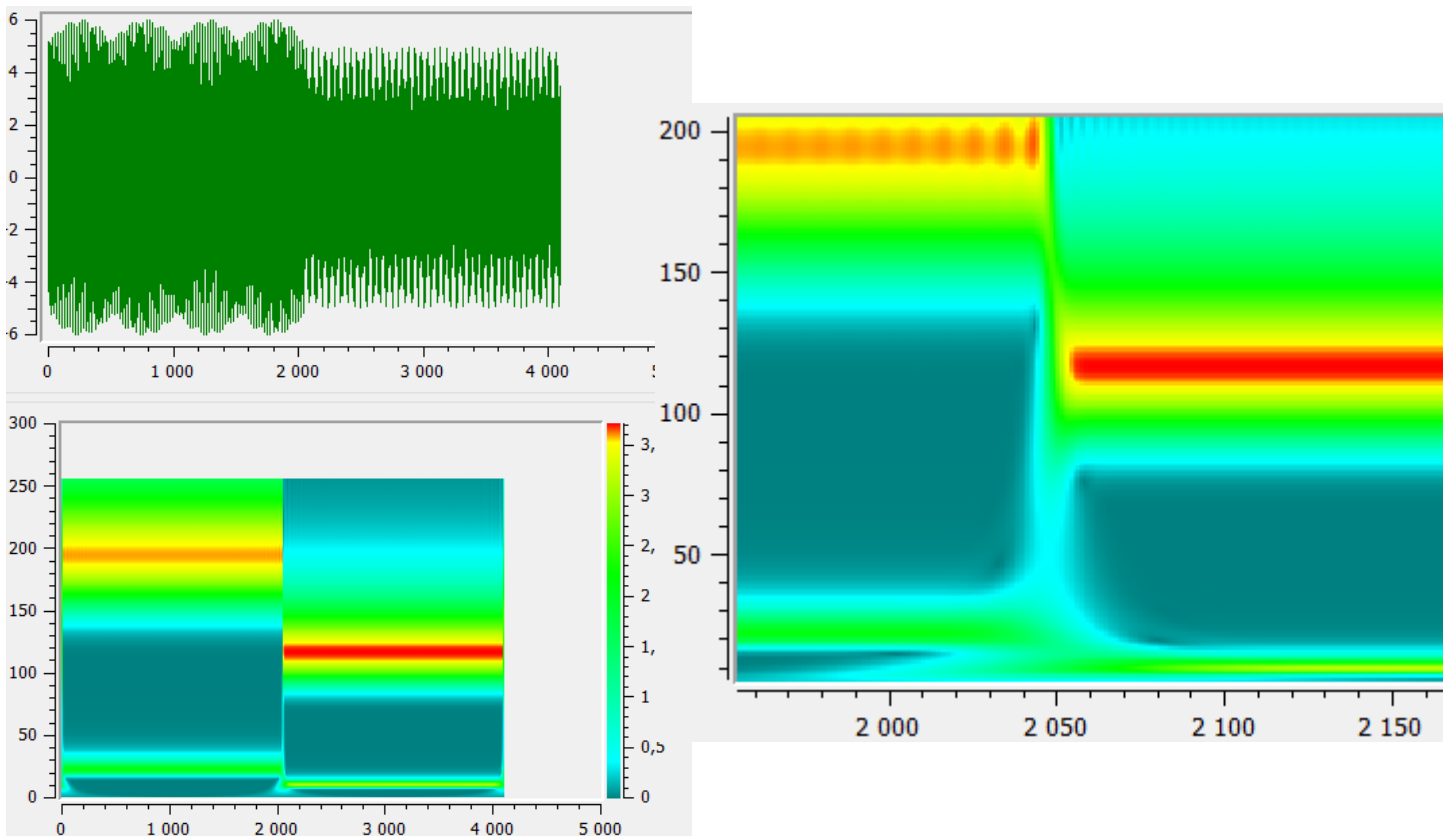


FIGURE 4.48 – Test Transformée en Ondelettes 3

4.4.2 Tests de validation fonctionnelle

Les tests de validation fonctionnelle ont été effectués à la souris, directement sur le logiciel. D'après la méthode de gestion projet utilisée, ces tests, comme les tests unitaires, n'ont pas nécessaire lieu à la fin du projet, ni même à la fin d'un sprint.

En effet, dès que le développement d'une fonctionnalité arrive à son terme, cette dernière est testée. Si elle ne satisfait pas, le développement reprend.

Ainsi, les tests fonctionnels ont été réalisés pour toutes les fonctionnalités de la solution. Il n'est pas utile de les énoncer dans ce rapport, car cela serait long et répétitif. Mais j'ai trouvé judicieux de souligner l'importance de ces tests et la façon dont ils se sont déroulés.

4.4.3 Test de validation solution

Un test de validation solution a pu être effectué. Cela aurait été appréciable d'en réaliser plusieurs, mais le temps a fait que cela n'était plus possible.

En effet, le test de validation solution, qui consiste à tester le logiciel dans des conditions réelles, a été effectué vers la fin du projet. Il s'est déroulé dans le département de pédopsychiatrie du CHRU de Bretonneau à Tours, en présence de Sylvie Roux, Nicole Bruneau et Marie Gomot.

Voici le déroulement complet du test :

1. Installation du logiciel sur le PC de stimulations
2. Exécution du logiciel
3. Paramétrage audio : calibration du volume sonore, en testant avec un casque audio
4. Préparation du cobaye : pose des électrodes, installation dans le fauteuil, pose du casque audio
5. Création d'un scénario, édition en alternant entre mode graphique et textuel
6. Ouverture du scénario créé (pour tester cette dernière fonction)
7. Sur le PC d'acquisition, lancement de l'acquisition avec ActiView
8. Lancement du scénario
9. A la fin du scénario, transfert du fichier BDF enregistré par ActiView, vers le PC de stimulations
10. Importation du fichier BDF et pré-traitement
11. Visualisation des signaux
12. Analyse des signaux

Le test s'est déroulé avec succès. En revanche, la fonctionnalité d'exportation des résultats n'était pas encore réalisée au jour du test.

4.4.4 Tests de validation robustesse

Des tests de robustesse ont été effectués, depuis le moment où la fonctionnalité d'importation posait des problèmes de performance (même si tests de robustesse et tests de performance ne sont pas la même chose, c'est en améliorant la performance qu'il a fallu vérifier la robustesse). Pour rappel, la robustesse correspond à la stabilité du logiciel. On parle surtout de gestion de la mémoire vive. Ce point a été traité dans la section "Gestion de la mémoire vive" de la partie "Quelques points sur le développement".

Pour tester la mémoire du logiciel, l'aide du Gestionnaire des tâches de Windows a été très apprécié. En effet, vu que le programme traite avec des volumes de données assez conséquents, on peut facilement voir une fuite de mémoire entre deux clics. Mais il a surtout été utile pour vérifier que les différentes optimisations ont bien été faites.

Sur Linux, j'avais l'habitude d'utiliser Valgrind, un outil très performant notamment pour la détection de fuites de mémoire. Sur Windows, Valgrind n'existe pas encore (le projet est en cours). J'ai tenté d'utiliser Microsoft Windows Performance Toolkit, qui est sensé faire le même travail, mais il est réputé comme difficile : cela me prenait trop de temps de l'appivoiser et j'ai abandonné. Les autres que l'on m'a indiqués étaient payants (Purify, Insure++...).

Ceci dit, le bilan sur le mémoire est positif, car je n'ai jamais rencontré de bug suite à une fuite de mémoire depuis la dernière validation du logiciel.

Bilan sur le planning

5.1 Planning prévisionnel

Comme vu dans le chapitre Méthode de gestion de projet, le projet a été découpé en 3 sprints, c'est-à-dire 3 périodes d'un mois chacune. Chaque sprint est un cycle comportant développement, tests, déploiement et recette. Dans les estimations des tâches, tout est compté. Ce n'est pas le plus précis, car on n'a pas de contrôle sur la durée des tests et du déploiement. Mais dans un projet de cette envergure, et avec mon expérience, les estimations de chaque tâche sont assez élevées pour permettre de faire tout cela. De plus, les créneaux alloués aux PFE ne sont pas à plein temps, et il est possible de travailler hors-créneaux.

5.1.1 Sprint 1

Le premier sprint comprend la réalisation des parties suivantes :

- la gestion de fichiers scénario
- l'édition de scénario (non graphique)
- l'affichage de scénario
- la gestion du son

5.1.2 Sprint 2

Le deuxième sprint comprend la réalisation des parties suivantes :

- l'édition de scénario (graphique)
- la communication via le port parallèle
- la gestion de fichiers BDF

5.1.3 Sprint 3

Le troisième et dernier sprint comprend la réalisation des parties suivantes :

- le pré-traitement des signaux
- l'analyse des signaux
- l'affichage des signaux
- l'exportation des données

5.1.4 Mise en production

Même si chaque sprint comporte une étape de déploiement, un laps de temps est réservé à la mise en production finale du produit, avec livraison d'un manuel utilisateur.

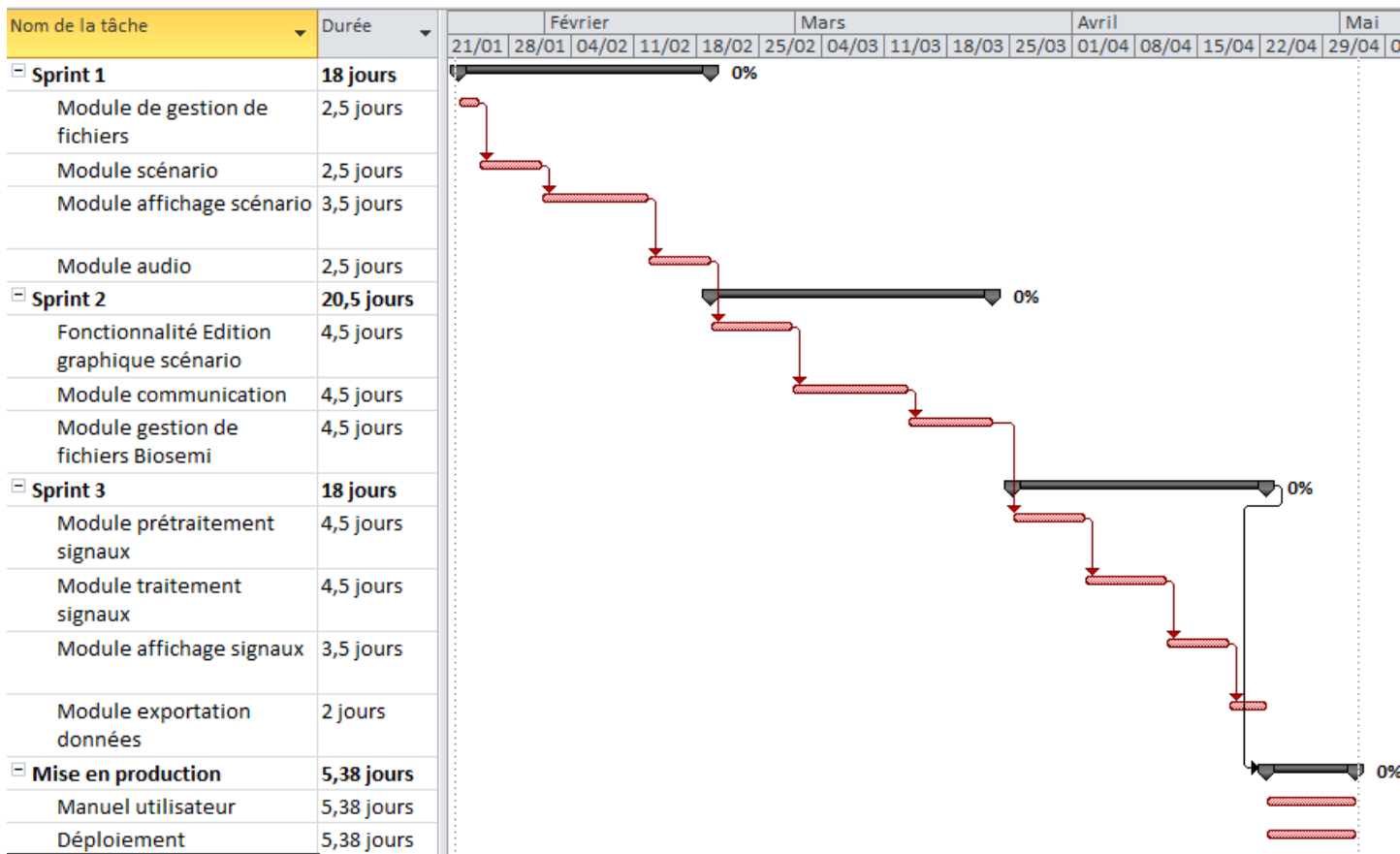


FIGURE 5.1 – Planning prévisionnel par sprint

5.2 Comparaison avec le planning effectif

	début	fin		
Gestion de fichiers scénario	13/12/2012	13/12/2012		soutenance ou réunion
Edition scénario textuelle	13/12/2012	20/12/2012		développement
Gestion de fichiers scénario	22/01/2013	22/01/2013		tests
Edition scénario textuelle	22/01/2013	22/01/2013		déploiement
Affichage scénario	22/01/2013	24/01/2013		autres
Affichage scénario	24/01/2013	24/01/2013		
Edition scénario (graphique)	24/01/2013	29/01/2013		
Audio	29/01/2013	31/01/2013		
Edition scénario (graphique)	30/01/2013	30/01/2013		
Soutenance de mi-parcours	30/01/2013	30/01/2013		
Audio	05/02/2013	07/02/2013		
Déploiement Sprint 1	07/02/2013	07/02/2013		
Envoi de triggers	12/02/2013	13/02/2013		
Importation BDF	13/02/2013	12/03/2013		
Envoi de triggers	19/02/2013	20/02/2013		
Déploiement Sprint 1 v2 (réunion)	20/02/2013	20/02/2013		
Pré-traitement (hors filtrage)	21/02/2013	22/02/2013		
Réunion	05/03/2013	05/03/2013		
Calibration audio	06/03/2013	06/03/2013		
Calibration audio	06/03/2013	06/03/2013		
Visualisation signaux	13/03/2013	13/03/2013		
Importation BDF	14/03/2013	14/03/2013		
Pré-traitement (filtrage)	14/03/2013	20/03/2013		
Visualisation signaux (zoom)	21/03/2013	21/03/2013		
Pré-traitement	21/03/2013	21/03/2013		
Visualisation signaux	21/03/2013	21/03/2013		
Analyse temps-fréquence	26/03/2013	03/04/2013		
Analyse temps-fréquence	03/04/2013	04/04/2013		
Rédaction du rapport	04/04/2013	30/05/2013		
Test Validation solution	10/04/2013	10/04/2013		
Déploiement Sprint 2,5	10/04/2013	10/04/2013		
Exportation	22/04/2013	22/04/2013		
Exportation	22/04/2013	22/04/2013		
Rédaction manuel utilisateur	29/04/2013	30/04/2013		
Déploiement Sprint 3	début mai			

FIGURE 5.2 – Planning effectif

Ce planning présente certaines différences avec le planning prévisionnel. Ceci est du à une anticipation de certaines tâches, ainsi qu'à une réévaluation des sprints.

En effet, la fonctionnalité "Gestion de fichiers scénario" a été développée au sprint 0 et testée au démarrage du sprint 1. De plus, en regardant le tableau suivant, qui expose l'évaluation des charges du Sprint 1, la charge estimée est de 6 jours (même sans la tâche qui a été enlevée) alors que la charge réalisée s'élève à 9 jours. Cela ne veut pas dire que j'ai mis 9 jours à faire ce que j'avais estimé pouvoir réaliser en 6 jours, au contraire : j'ai réalisé pendant le sprint 9 jours de charge, alors que j'avais estimé ne pouvoir en effectuer que 6. C'est pourquoi la tâche de réalisation de la fonctionnalité "Edition scénario (graphique)" a été ajoutée au sprint. On voit alors, toujours sur le tableau suivant, que le travail était sous-estimé.

	points	charge estimée (j/h)	charge effective (j/h)	Remarques
Gestion de fichiers scénario	4	2,5	2	déjà réalisé au sprint 0
Edition Scénario (non graphique)	4	2,5	3	
Affichage scénario	6	3,5	3	
Audio	4	2,5	6	
Total	18	11	14	
Total sprint 1 (sans sprint 0)	10	6	9	
<i>Ajout tâche :</i>				
Edition Scénario (graphique)	7	4,5	4	
Total Sprint 1 ajusté	17	10,5	13	

FIGURE 5.3 – Tableau d'évaluation du Sprint 1

C'est pourquoi les deux sprints suivants ont été rééquilibrés. Les tâches "Pré-traitement" et "Analyse" ont été intégrées au Sprint 2. Ce dernier est donc plus conséquent, et le Sprint 3 est ainsi soulagé, car plus court que les deux premiers (rédaction du rapport, deux semaines de pause pédagogique, projet d'option en parallèle...).

Evolution

6.1 Améliorations possibles

Le logiciel comporte toutes les fonctionnalités décrites dans le cahier de spécifications. Cependant, il reste possible de les améliorer, et d'en apporter de nouvelles.

6.1.1 Stimulations auditives multiples

Pour l'instant, le logiciel ne peut délivrer à la fois qu'une stimulation auditive par canal (gauche, droit) en plus du bruit en stéréo. L'idéal serait de pouvoir jouer autant de stimulations que l'on souhaite, simultanément.

Par exemple, on pourrait vouloir tester l'entraînement cérébral des battements binauraux en créant deux battements binauraux différents :

- l'un de fréquence 4 Hz, par exemple généré par un signal de fréquence 400 Hz à gauche, et un signal de 404 Hz à droite
- l'autre de fréquence 12 Hz, par exemple généré par un signal de fréquence 650 Hz à gauche, et un signal de 662 Hz à droite

On pourrait créer un scénario jouant tous ces signaux, ce qui permettrait de percevoir deux battements binauraux différents. Le concept serait de, au cours du temps, modifier l'amplitude des signaux afin de percevoir plus ou moins distinctement chaque type de battements binauraux. Par exemple :

1. Pendant 5 minutes : battements binauraux de 4 Hz d'amplitude 75 %, battements binauraux de 12 Hz d'amplitude 25 %
2. Pendant 10 minutes : diminution de l'amplitude des battements binauraux 4 Hz, augmentation de l'amplitude des battements binauraux 12 Hz
3. Pendant 10 minutes : battements binauraux de 4 Hz d'amplitude 10 %, battements binauraux de 12 Hz d'amplitude 90 %

L'intérêt d'un tel concept serait d'avoir des stimulations plus proches de la vraie forme des ondes cérébrales, ce qui pourrait peut-être amplifier les effets sur le cerveau.

Au niveau du logiciel, pour l'utilisateur, cela se traduirait par une démultiplication de l'onglet Scenario. Il y aurait plusieurs sous-onglets permettant de créer une évolution de chaque stimulation.

Au niveau de la modélisation, il faudrait adapter la classe Scenario, pour qu'elle contienne une liste d'objets de la classe qu'on pourrait appeler Stimulation, elle-même contenant une liste d'objets Step. Il faudrait également adapter la gestion du son pour prendre en compte la lecture de toutes les stimulations à la fois.

6.1.2 Paramétrage de l'analyse

Pour le moment, le logiciel effectue l'analyse temps-fréquence avec la transformée en ondelettes sur l'intervalle de fréquences [1-25] (Hz). L'ondelette mère utilisée est Morlet.

Les utilisateurs pourraient avoir le désir de choisir la plage de fréquences à analyser, et l'ondelette mère.

6.1.3 Edition graphique pour tous les paramètres

Il est possible de planifier dans les domaines temporel et fréquentiel les stimulations du scénario. Mais il n'est pas possible de régler l'amplitude des signaux, et du bruit, directement sur le graphe. Cela est une fonctionnalité à apporter.

Cela supposerait plusieurs vues graphiques dans l'onglet Scenario.

6.1.4 Acquisition directe

Une autre possibilité serait de permettre l'acquisition des signaux EEG directement via le logiciel. Il serait ainsi inutile de passer par ActiView.

Cela demanderait une excellente gestion de la mémoire, car l'acquisition implique d'enregistrer une soixantaine de signaux en mémoire vive (à la base) pendant une durée indéterminée.

6.2 Nouvelles perspectives pour le projet

L'équipe de chercheurs a montré un réel intérêt pour le logiciel, non plus comme une solution de tests pour les battements binauraux, mais comme une solution de tests de stimulations/analyse EEG à part entière.

L'idée serait d'avoir un logiciel complet, permettant de jouer des stimulations de toutes sortes afin d'avoir des scénarios de tests pour l'étude de phénomènes divers et variés, et d'analyser dans le domaine temps-fréquence les signaux EEG enregistrés.

Si un tel projet était mené à terme, il pourrait faire partie des solutions phares dans le domaine de l'étude électroencéphalographique.

Conclusion

Ce projet a été pour moi très enrichissant dans de nombreux domaines et m'a permis de me travailler sur des thèmes qui m'intéressaient mais pour lesquels je n'avais pas de temps à consacrer. En effet, toujours attiré par les sciences cognitives d'une part, et par l'acoustique et le traitement du son d'autre part, j'ai pu relier l'informatique à ces deux thématiques et j'en suis vraiment ravi.

Les compétences que j'ai acquises ou renforcées sont nombreuses :

- La *spécification*. Ce point a été principalement bien abordé, même si les modifications entre chaque sprint du projet n'ont pas été annexées au Cahier de Spécifications. J'ai essayé d'être au plus proche de ce qui pourra être attendu en entreprise, même si je n'ai pas vraiment idée de la forme que cela prendra.
- La *gestion de projet*. Depuis le projet d'ingénierie logiciel de l'année dernière, dans lequel j'ai joué le rôle de Chef de projet, j'ai pris goût à ce métier. Le fait de travailler seul cette année et devoir néanmoins mener une politique de conduite de projet permet de tester de nouvelles choses que je n'aurais pas essayé si j'avais eu la responsabilité d'une équipe.
- La *modélisation et l'architecture*. J'ai toujours apprécié structurer des programmes avant de les développer, et c'est pourquoi un certain nombre de diagrammes UML (ou non) apparaît dans ce rapport. Grâce à ce projet j'ai remis le nez dans des livres et sites de modélisation, ce qui m'a beaucoup apporté.
- Le *développement en C++*. Déjà à l'aise dans ce langage, j'ai vraiment eu l'occasion cette année de conforter mes compétences. Ma maîtrise du framework Qt s'est nettement améliorée et je consulte de plus en plus rarement les forums d'entraide, du moins pour l'aspect Qt pur.
- L'*acoustique*. N'aimant pas implémenter des concepts que je ne comprends pas, j'ai été contraint de me documenter sur l'acoustique, domaine qui m'intéresse particulièrement.
- Le *traitement du signal*. N'effectuant pas cette option, j'ai l'impression de l'avoir quand même suivie. Pour comprendre le concept du filtrage du signal et de la transformée en ondelettes, il m'a fallu savoir ce qu'était réellement un signal, une transformée, un filtre... Finalement, j'ai acquis de nombreuses notions dans ce domaine, même si je suis loin de pouvoir prétendre le maîtriser.

Le sujet initial, que j'avais proposé avant le début de l'année, a été modifié, mais je ne regrette pas de m'être orienté dans cette direction. Ce sujet est plus juste, possède un esprit plus scientifique et m'a fait explorer plus de notions. J'ai pu utiliser du matériel électroencéphalographique, et cela grâce à l'équipe Autisme de l'UMR INSERM du CHRU de Bretonneau, Sylvie Roux, Marie Gomot, Nicole Bruneau, ainsi qu'à mon encadrant Pascal Makris qui a cru en mon idée et a usé de ses contacts pour me faire rencontrer ces chercheuses. Je remercie donc toutes ces personnes, ainsi que le jury de validation des sujets de PFE qui a accepté mon sujet et m'a permis de réaliser ce projet.

Enfin, je ne doute pas que le projet sera repris lors de futurs PFE, celui-ci donnant des possibilités d'ouverture dans le domaine du logiciel de stimulations auditives et d'analyse du biopotential électrique. C'est pourquoi j'ai essayé de l'implémenter en restant le plus souple possible, afin de laisser un code clair et réutilisable. Je me tiendrai à la disposition de quiconque demandera des informations pour la reprise du projet.

Références

- Qt [<http://qt.digia.com>]
- Biosemi [<http://www.biosemi.com>]
- FMODE [<http://www.fmod.org>]
- Qwt [<http://qwt.sourceforge.net>]
- EDFLib [<http://www.teuniz.net/edflib>]
- CwtLib [<http://sourceforge.net/projects/cwtlib/>]

- S. Blanco, R. Quian Quiroga, O.A. Rosso, S. Kochen, Time-frequency analysis of electroencephalogram series, 1995.
- H. Swildens, Concepts of EEG processing : from power spectrum to bispectrum, fractals, entropies and all that, 2006.
- T. Malina, A. Folkers, U.G. Hofmann, Real-time EEG processing based on Wavelet Transformation, 2002.
- C. D’Avanzo, V. Tarantino, P. Bisiacchi, G. Sparacino, A wavelet Methodology for EEG Time-frequency Analysis in a Time Discrimination Task, 2009.
- Y. Meyer, S. Jaffard, O. Rioul, L’analyse par ondelettes, 1987.
- R. George, S. J. Cox, An Introduction to the Analysis of Brain Waves, 2011.

- Dan Russell : Acoustics and Vibration Animations [<http://www.acs.psu.edu/drussell/demos.html>]
- Mathworks (explications sur les algorithmes de Matlab) [<http://www.mathworks.fr>]
- Wavelets and Time-Frequency Analysis [<http://mudasir.hubpages.com/hub/wavelets1>]

- F. Holmes Atwater, Accessing Anomalous States of Consciousness with a Binaural Beat Technology, 1997.
- F. Holmes Atwater, Binaural Beats and Regulation of Arousal Levels, 2009.
- G. Oster, Auditory Beats in the Brain.
- J. D. Lane, J. E. Owens, G. R. Marsh, Binaural Auditory Beats Affect Vigilance Performance and Mood, 1998.
- J. Gunnell, An Illustration of Binaural Beats, 2009.
- TL. Huang, C. Charyton, A Comprehensive Review of the Psychological Effects of Brainwave Entrainment, 2008.
- H. Wahbeh, C. Calabrese, H. Zwickey, Binaural Beat Technology in Humans : a Pilot Study to Assess Psychologic and Physiologic Effects, 2007.

Manuel utilisateur

Ce document est la notice d'utilisation du logiciel de tests des battements binauraux réalisé par François Dennig dans le cadre de son Projet de Fin d'Etudes à Polytech Tours, durant l'année universitaire 2012-2013.

B.1 Installation

Aucune installation n'est requise pour le programme. Il suffit de le copier n'importe où sur votre disque dur, et de l'exécuter. Attention à bien copier tout le dossier, et non pas seulement l'exécutable ! En effet, les fichiers DLL associés sont importants pour bonne exécution du logiciel.

B.2 Lancement pour la première fois

Si c'est la première fois que vous lancez le logiciel sur votre ordinateur, quelques paramètres sont à faire. Ceux-ci se font sur l'onglet de configuration de la fenêtre principale. Cet onglet est le premier à s'afficher lors du lancement du logiciel.

B.2.1 Test et choix du périphérique audio

Il est possible que votre ordinateur possède plusieurs cartes son. Pour le savoir, cliquez sur le bouton *Scan*. Une liste déroulante se remplit, et vous pouvez choisir ce qui vous convient.

Il est important de tester la balance de votre périphérique. Pour ce faire, cliquez sur le bouton *Test Left Ear*. Si tout se passe bien, vous entendrez un son de bruit blanc dans l'oreille gauche. Si vous entendez aussi du son dans l'oreille droite, c'est que votre périphérique est mal configuré. Cliquez une nouvelle fois sur le bouton pour arrêter le son. Répétez la procédure avec le bouton *Test Right Ear* pour tester l'oreille droite.

Une fois que vous avez choisi et testé votre périphérique, cliquez sur le premier bouton *It's OK*.

B.2.2 Calibration du volume sonore

Si vous utilisez un appareil externe pour gérer le son dans votre pièce d'expérimentations, vous avez sûrement pour consigne de ne pas modifier le volume général du système Windows. C'est pourquoi vous pouvez régler le volume directement dans le logiciel.

Pour commencer la calibration, cliquez sur le bouton *Begin calibration*. Un bruit blanc est joué à travers les deux haut-parleurs. Réglez le volume à votre convenance à l'aide du bouton coulissant correspondant à *Software digital value*.

Vous pouvez aussi utiliser un sonomètre pour mesurer le volume réel à la sortie de vos haut-parleurs. Dans ce cas, fixez la valeur de votre sonomètre dans la boîte *Real value*. Ainsi, le logiciel vous indiquera le volume réel à chaque fois que vous planifierez un son lors de vos tests.

Une fois que vous êtes satisfait, cliquez sur le deuxième bouton *It's OK*.

Les valeurs de calibration que vous avez rentrées sont maintenant enregistrées dans la base de registres de Windows, et vous n'aurez plus à effectuer cette étape. En revanche, vous devrez toujours choisir le périphérique audio au démarrage du logiciel.

B.3 Création d'un scénario

Pour créer un scénario, cliquez sur le bouton *Créer un scénario* en haut de la fenêtre. L'onglet Description devient l'onglet en cours.

B.3.1 Description d'un scénario

Vous pouvez donner un titre et une description au scénario, dans les deux zones de texte prévues à cet effet dans l'onglet Description. Vous n'avez pas à cliquer sur un bouton pour valider.

B.3.2 Edition d'un scénario

Pour éditer un scénario, entrez dans l'onglet Scenario.

Ajout d'une étape

Pour ajouter une étape, vous avez deux méthodes à votre disposition.

Ajout d'une étape via la boîte de dialogue Cliquez sur le bouton *Add step* à gauche de la fenêtre. La boîte de dialogue *Step properties* apparaît. C'est ici que vous allez entrer tous les paramètres de l'étape :

- *Left canal frequency* : correspond à la fréquence de l'onde sonore jouée dans le canal gauche. La valeur *From* indique la valeur de la fréquence au départ de l'étape, alors que la valeur *To* indique la valeur de la fréquence à la fin de l'étape. Si ces deux valeurs sont différentes, alors la fréquence suivra une pente linéaire entre ces deux valeurs tout au long de l'étape.
- *Right canal frequency* : correspond à la fréquence de l'onde sonore jouée dans le canal droit. Même remarque pour les valeurs.
- *Amplitude* : correspond à l'amplitude, ou au volume sonore des ondes. Elles sont définies en pourcentage du volume logiciel fixé au lancement du logiciel. Si une calibration sur le volume réel a été faite, vous pouvez voir la valeur correspondant au volume réel.
- *Noise* : correspond à l'amplitude, ou au volume sonore du bruit blanc. Il est par défaut fixé à 0 % mais vous êtes libre de fixer la valeur de votre choix.
- *Duration* : correspond à la durée de l'étape.

Cliquez sur le bouton *OK* lorsque vous avez paramétré l'étape. Celle-ci s'affiche normalement sur le graphique et dans la zone de gauche.

Ajout d'une étape via le graphique Pour ajouter une étape via le graphique, il suffit d'exercer un clic droit dessus. Pour modifier ses paramètres, référez-vous au paragraphe *Modification d'une étape via le graphique*.

Modification d'une étape

Modification d'une étape via la boîte de dialogue Cliquez simplement une fois sur l'intitulé de l'étape, dans la liste des étapes à gauche de la fenêtre. La boîte de dialogue s'ouvre. Le principe est le même que dans le paragraphe *Ajout d'une étape via la boîte de dialogue*.

Modification d'une étape via le graphique Effectuez un clic maintenu sur un des points (extrémités) de l'étape, et déplacez-le :

- dans le sens des abscisses pour modifier la durée de l'étape (ou celle de l'étape précédente)
- dans le sens des ordonnées pour modifier la fréquence de l'onde de l'un des canaux.

Vous ne pouvez pas modifier l'amplitude lorsque vous modifiez une étape via le graphique. Pour cela, référez-vous au paragraphe *Modification d'une étape via la boîte de dialogue*.

Suppression d'une étape

Pour supprimer une étape, cliquez sur le bouton *Delete* à droite de l'intitulé de l'étape désirée, dans la liste des étapes à gauche de la fenêtre.

B.4 Chargement/Ouverture d'un scénario

Pour charger/ouvrir un scénario existant, cliquez sur le bouton *Open a scenario* en haut de la fenêtre principale. L'onglet Description devient alors l'onglet en cours.

B.5 Démarrage du scénario

Les actions suivantes sont à effectuer dans l'onglet *Control*.

B.5.1 Activation du triggering

Si vous voulez voir les repères de changement d'étape lors de la visualisation et de l'analyse, vous devez activer l'envoi de triggings au matériel Biosemi. Pour ce faire, cochez la case *Enable EEG Triggering*.

B.5.2 Démarrage

Pour faire démarrer le scénario, cliquez sur le bouton *Run*. Laissez ensuite le scénario jouer, jusqu'à la fin. Si toutefois vous désirez tout arrêter, cliquez sur le bouton *Stop*.

B.6 Importation BDF et pré-traitement

Les actions suivantes sont à effectuer dans l'onglet Analysis.

Pour importer un fichier de données d'acquisition Biosemi BDF, cliquez sur le bouton *Import Biosemi out file*. Lorsque vous avez choisi votre fichier, une barre de chargement apparaît et défile, jusqu'à aboutir sur une boîte de dialogue de configuration des canaux de pré-traitement.

B.6.1 Configuration des canaux de pré-traitement

Cochez les cases correspondant aux canaux de pré-traitement que vous avez utilisé pour cette acquisition, et si besoin, changez la référence de l'électrode associée. Normalement, tout devrait être OK.

B.6.2 Pré-traitement

Passé la boîte de dialogue de configuration des canaux de pré-traitement, une nouvelle barre de chargement s'affiche et défile pendant que le pré-traitement s'effectue. Lorsque la barre disparaît, le pré-traitement est terminé.

B.7 Visualisation des signaux

Pour visualiser les signaux que vous avez préalablement chargés au format BDF, et éventuellement pré-traités, cliquez sur le bouton *Show raw signals* en bas de l'onglet.

Le signal du premier canal s'affiche automatiquement.

B.7.1 Changement du canal

Pour changer le canal, choisissez-le dans la liste déroulante en haut à gauche de la fenêtre de visualisation.

B.7.2 Zoom

Pour zoomer, dessinez un rectangle à l'aide de votre souris sur la zone que vous souhaitez voir grossir. Pour annuler le zoom, effectuez un simple clic droit sur le graphique.

B.8 Analyse temps-fréquence

Pour configurer et effectuer une analyse temps-fréquence d'un signal, cliquez sur le bouton *Analyse* de l'onglet Analysis.

Une nouvelle fenêtre apparaît dans laquelle vous devez configurer l'analyse.

B.8.1 Configuration de l'analyse

Choisissez le canal dont vous souhaitez analyser le signal dans la liste à gauche de la fenêtre.

Puis choisissez la bande de fréquences que vous souhaitez analyser. **Attention** : ne choisissez pas un intervalle trop grand!!! L'intervalle par défaut est [1-25] Hz. Cela est normalement suffisant pour l'étude des battements binauraux. Si vous souhaitez toutefois augmenter agrandir cette fenêtre, assurez-vous que votre ordinateur est équipé de suffisamment de mémoire vive, où le logiciel risquerait de crasher.

B.8.2 Démarrage de l'analyse

Pour lancer l'analyse, cliquez sur le bouton *Begin Analysis*. Une barre de chargement s'affiche. Lorsqu'elle disparaît, cela signifie que l'analyse est terminée. Attention cependant, l'affichage de la représentation graphique peut durer une ou deux secondes de plus.

B.8.3 Interprétation de l'analyse : échelles de couleur, zoom

Pour mieux interpréter l'analyse, vous pouvez modifier les valeurs d'échelle de couleur afin de plus ou moins dégager certaines intensités de fréquence.

Vous pouvez aussi zoomer l'image en dessinant un rectangle avec votre souris sur la zone souhaitée. Annulez le zoom en effectuant un clic droit. Ces procédures peuvent être plus ou moins réactives suivant la bande de fréquences choisie pour l'analyse et la capacité en mémoire vive de l'ordinateur.

B.8.4 Exportation des résultats

Vous pouvez exporter les résultats sous forme d'une matrice au format txt, en cliquant sur le bouton *Export*.

Cahier de Spécifications

Département Informatique

Cahier de spécification système & plan de développement

Projet :	Outil de test pour la synchronisation cérébrale par battements binauraux	
Emetteur :	François Dennig	Coordonnées : EPU-DI
Date d'émission :	8 janvier 2013	

Validation

Nom	Date	Valide (O/N)	Commentaires
-----	------	--------------	--------------

Pascal Makris : 06/12/2012 ; O ; Ok première version

Pascal Makris : 08/01/2013 ; O ; Validé

Historique des modifications

Version	Date	Description de la modification
---------	------	--------------------------------

00 : 12/2012 ; Version initiale

01 : 12/2012 ; Corrections et précisions

02 : 01/2013 ; Partie Architecture modifiée et complétée.

Table des matières

Cahier de spécification système	6
1.1 Introduction	6
1.2 Contexte de la réalisation	6
1.2.1 Contexte	6
1.2.2 Objectifs	8
1.2.3 Bases méthodologiques	8
1.2.4 Licence	9
1.3 Description générale	10
1.3.1 Environnement du projet	10
1.3.2 Caractéristiques des utilisateurs	10
1.3.3 fonctionnalités et structure générale du système	11
1.3.4 Contraintes de développement, d'exploitation et de maintenance	12
1.4 Description des interfaces externes du logiciel	13
1.4.1 Interfaces matériel/logiciel	13
1.4.2 Interfaces logiciel/logiciel	13
1.4.3 Interfaces homme/machine	15
1.5 Architecture générale du système	21
1.5.1 Description des composants	22
1.6 Description des fonctionnalités	24
1.6.1 Fonction [Ouvrir scénario]	24
1.6.2 Fonction [Créer scénario vierge]	24
1.6.3 Fonction [Sauvegarder scénario]	25
1.6.4 Fonction [Configurer scénario]	25
1.6.5 Fonction [Ajouter étape]	25
1.6.6 Fonction [Configurer étape]	26
1.6.7 Fonction [Activer triggering Biosemi]	26
1.6.8 Fonction [Lancer scénario]	27
1.6.9 Fonction [Visualiser avancement scénario]	27
1.6.10 Fonction [Choisir les canaux]	28
1.6.11 Fonction [Choisir la gamme de fréquences]	28
1.6.12 Fonction [Lancer l'analyse]	29
1.6.13 Fonction [Visualiser signaux bruts]	29
1.6.14 Fonction [Visualiser signaux transformés]	30
1.6.15 Fonction [Exporter les données]	30
1.7 Utilisation des fonctionnalités au cours du temps	31
1.8 Conditions de fonctionnement	32
1.8.1 Performances	32
1.8.2 Capacités	32
1.8.3 Contrôlabilité	32
1.8.4 Sécurité	32

Plan de développement	34
2.1 Méthode de planification adoptée	34
2.2 Calcul du ratio 'Durée totale de travail / Complexité totale'	34
2.3 Découpage du projet en tâches	35
2.3.1 Tâches de spécifications	35
2.3.2 Tâches d'analyse et de modélisation	35
2.3.3 Tâches de développement	36
2.3.4 Tâches de mise en production	37
2.3.5 Tâches annexes	37
2.4 Planning	38
2.4.1 Chronologie générale du projet	38
2.4.2 Planning détaillé du projet à partir du sprint 1	39
A Références	40

Cahier de spécification système

1.1 Introduction

Ce document détaille les spécifications du projet de fin d'étude de François Dennig, étudiant au département informatique de Polytech' Tours.

Le projet concerne aussi directement :

- Pascal Makris, encadrant école
- Nicole Bruneau, Marie Gomot (CR1 INSERM) et Sylvie Roux (IR), équipe "Autisme" de l'UMR INSERM U930, Université François Rabelais de Tours (Centre Universitaire de Pédopsychiatrie, CHRU Bretonneau Tours)
- Nicolas Ragot, responsable des Projets de Fin d'Etude pour l'année universitaire 2012-2013.

Ce document n'est valable qu'après validation des acteurs cités précédemment. Il représente un engagement de la part de chacune des parties dans la prévision des aspects du projet qui pourront y être développés.

1.2 Contexte de la réalisation

1.2.1 Contexte

Le projet vise la recherche en neurophysiologie. Le produit final devra servir d'outil de test pour établir les effets des battements binauraux sur le cerveau et sera utilisé par des chercheurs.

Les battements binauraux

Imaginons deux oscillateurs. Le premier génère un signal sinusoïdal, le second génère un signal sinusoïdal de fréquence légèrement différente à celle du premier. Lors de l'association des sorties de ces deux oscillateurs, il se produit en sortie une modulation d'amplitude provoquant un battement dont la période est égale à la différence de fréquence entre les deux signaux.

Lorsque cette association est faite avant la sortie audio, on qualifie le phénomène de battements monoraux.

En revanche, lorsque l'on attribue une sortie par oscillateur, et que l'on écoute la sortie de chaque oscillateur respectivement dans l'oreille gauche, et dans l'oreille droite, alors le battement est généré à l'intérieur du cerveau. On qualifie ce phénomène de battements binauraux.

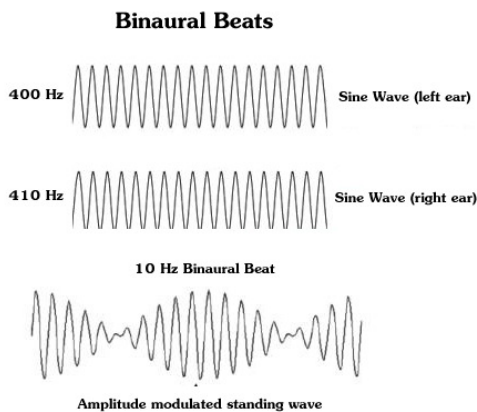


FIGURE 1.1 – Formation des battements binauraux (Atwater)

Observations sur la formation et la perception des battements binauraux

Les battements binauraux peuvent ne pas être formés selon les caractéristiques du signal. Ils peuvent aussi ne pas être perçus selon les personnes.

La meilleure fréquence de transport (fréquence de base des signaux) pour entendre les battements est de 440 Hz.

Pour une fréquence de transport inférieure à 90 Hz, il est possible de confondre les battements binauraux avec les signaux utilisés pour former ces battements.

Les battements binauraux sont perçus même si l'un des signaux est en dessous du seuil de l'audibilité humaine.

Les battements binauraux ne se forment que si les deux sons sont séparés de moins de 26Hz, voire 30Hz. Ce maximum diminue au fur et à mesure que la fréquence de transport s'éloigne des 440Hz.

Les personnes atteintes de la maladie de Parkinson ne peuvent pas entendre les battements binauraux. Au cours de leur traitement, s'il évolue positivement, les patients pourront les percevoir de plus en plus.

Lors de la période d'ovulation, les femmes ne peuvent pas entendre les battements binauraux (évolution de la perception tout au long du cycle menstruel).

L'âge n'influence pas la perception des battements binauraux, dans la mesure où une personne dont l'audition est partiellement réduite à cause du vieillissement peut toujours distinguer les sons entre 90Hz et 1KHz.

Les battements monoraux deviennent inaudibles lors de l'ajout de bruit. A l'inverse, La perception des battements binauraux est amplifiée par le bruit. En effet, le bruit masque les deux sons projetés, ce qui rend la perception des battements intra-auraux plus aisée.

Effets des battements binauraux

Les effets des battements binauraux n'ont jamais été clairement prouvés. Quelques études ont été réalisées, dont la plupart proviennent de l'Institut Monroe (institut privé vendant des séances de développement de soi). Les résultats des différentes études sont contradictoires et les bases de plusieurs d'entre elles sont inconnues ou insuffisantes.

Ceci étant dit, l'effet qui est sensé se produire est l'altération des états de conscience par l'entraînement des ondes cérébrales. En l'occurrence, une exposition à des battements binauraux dont la fréquence fait partie d'une certaine gamme de fréquence entraînerait la modification de la fréquence des ondes cérébrales, se calant dans la gamme appliquée. De ce fait, l'état de conscience d'un individu étant lié à la gamme de fréquence de ses ondes cérébrales, le fait de modifier ces dernières permettrait d'invoquer un état modifié de la conscience, et pourrait provoquer des effets tels que la relaxation, l'endormissement ou encore l'augmentation de l'attention, la concentration...

Un certain commerce s'est créé à partir de ce phénomène, et il se vend des CD d'écoute de ces battements binauraux. De rares centres médicaux les utilisent. Mais rien n'a été clairement établi quant à l'efficacité des effets sur l'état de conscience et les ondes cérébrales.

1.2.2 Objectifs

Le but du projet est de mettre en place une solution logicielle donnant la possibilité d'effectuer des tests qui, à long terme, devront permettre aux chercheurs d'établir les effets neurophysiologiques de l'écoute de battements binauraux sur l'humain.

Le système devra s'interfacer avec le matériel EEG de l'équipe de recherche, Biosemi. Celui-ci aura pour rôle d'acquérir les signaux émis par le cerveau des sujets de test.

Côté logiciel, on peut souligner trois grands aspects :

- paramétrage et génération des signaux sonores à émettre via la sortie audio
- envoi de signaux (triggers) au matériel EEG Biosemi via le port parallèle
- analyse des signaux enregistrés par le matériel EEG Biosemi via le fichier généré par ce dernier. Cette analyse n'est pas temps-réel.

1.2.3 Bases méthodologiques

Méthode de gestion de projet

La méthode utilisée est une méthode hybride, associant à la gestion de projet classique certaines spécificités des méthodes agiles. Voici les particularités de cette méthode :

- Le concept de cycle en V est conservé, mais on introduit plusieurs cycles dans le projet, appelés sprints. Chaque sprint comporte les étapes suivantes :
 - Développement
 - Tests
 - Déploiement
 - Recette
- A l'issue de chaque sprint, une version (ne comportant que certaines fonctionnalités) doit être fournie et vérifiée. Les phases de spécifications, analyse et modélisation sont réalisées avant les sprints (on appelle aussi cette partie Sprint 0).
- L'estimation des tâches se fait en évaluant leur complexité : chiffre arbitraire permettant de classer les tâches par ordre de difficulté. On évalue ensuite la complexité totale du projet et on calcule la charge de chaque tâche en jours/homme à partir de leur complexité (voir chapitre Plan de développement).

Langages et bibliothèques

- Le logiciel sera réalisé avec le langage C++. Une partie de code en langage C pourra être utilisée pour la communication via le port parallèle.
- Le framework Qt sera utilisé, notamment pour l'interface graphique.
- L'extension Qwt pour Qt servira à l'implémentation des visualisations graphiques.
- La bibliothèque FMOD Ex sera utilisée pour la gestion audio.
- La bibliothèque EDFLib sera utilisée pour la gestion de fichiers Biosemi.
- Les outils permettant le traitement du signal ne sont pas encore définis.

Méthode de traitement du signal

Il est possible de traiter un signal de multiples façons.

La technique la plus célèbre est la transformée de Fourier. Elle permet d'obtenir une représentation du spectre de fréquences d'un signal. Elle est très largement utilisée dans de nombreux domaines.

Cependant, cette méthode ne permet pas d'avoir l'évolution des fréquences en fonction du temps, la dimension temporelle est perdue. Elle n'est donc pas adaptée à l'analyse de signaux non-stationnaires. La transformée de Fourier à fenêtre glissante a été créée afin de pallier à ce problème : on effectue une transformée de Fourier à intervalles de temps très petits et réguliers. Cela permet d'ajouter la dimension temporelle à l'analyse.

Mais il existe une méthode plus rapide. Il s'agit de la transformée en ondelettes. Une ondelette est une onde très courte (aussi appelée vague) qui a un début et une fin. Le principe de la méthode est de représenter le signal comme une somme pondérée d'ondelettes, pouvant être dilatées ou translatées.

Pour effectuer l'analyse des signaux EEG, la transformée en ondelettes sera utilisée. Les choix de l'algorithme et de l'ondelette mère seront détaillés dans le Cahier d'Analyse.

1.2.4 Licence

L'utilisation des bibliothèques tierces citées précédemment sont soumises au respect de leurs licences respectives. La licence de l'application finale en est donc impactée. Voici les licences de ces bibliothèques :

Qt	Qwt	FMOD Ex	EDFLib
LGPL	Qwt licence	FMOD Non-Commercial licence	BSD licence permissive

TABLE 1.1 – Licences des bibliothèques tierces

- **LGPL** ("Lesser General Public Licence"), contrairement à GPL, permet d'utiliser la bibliothèque libre en question au sein d'un logiciel propriétaire. Ainsi, le développeur n'a pas besoin de publier le code source de son application. Cependant, s'il modifie le code source de la bibliothèque, il devra publier les modifications. Il faut citer l'utilisation de Qt.
- **Qwt Licence** est une LGPL modifiée. Les clauses modifiées ne concernent pas notre application, à part le fait que dériver des widgets de Qwt ne constitue pas une modification de la bibliothèque Qwt et permet donc toujours d'utiliser la bibliothèque dans une application propriétaire sans avoir à citer la Qwt Licence. Toutefois il faut citer l'utilisation de Qwt.
- **FMOD Non-Commercial Licence** : elle permet d'utiliser librement la bibliothèque, quelle que soit la licence finale, tant que le produit n'est pas vendu.
- **EDFLib** utilise une licence personnalisée correspondant à une licence de type BSD très permissive : il est possible d'utiliser et de modifier la bibliothèque même dans le cadre de projets commerciaux à sources fermées.

1.3 Description générale

1.3.1 Environnement du projet

Le système sera mis en place au sein du service d'explorations fonctionnelles du bâtiment de pédo-psychiatrie au CHRU Bretonneau. Le logiciel à développer sera installé sur un poste Windows et devra s'interfacer avec Biosemi (matériel d'acquisition EEG). Le poste et le matériel sont déjà disponibles et en fonctionnement dans le service. En outre, des électrodes et un casque stéréophonique (ou des oreillettes) seront utilisés.

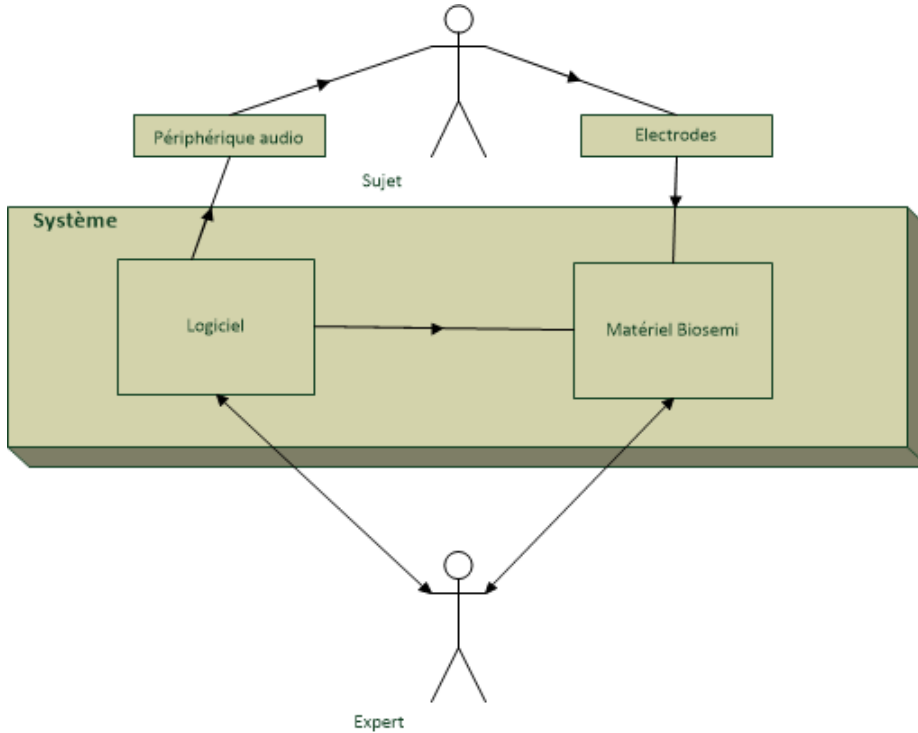


FIGURE 1.2 – Environnement du projet

1.3.2 Caractéristiques des utilisateurs

Deux types d'utilisateurs utiliseront le système.

- Type "Expert" : il s'agit des chercheurs en neurophysiologie qui établiront les tests et interagiront activement avec le système. Ils ont une bonne expérience du matériel Biosemi. Leur connaissance de l'informatique est variable suivant les utilisateurs.
- Type "Sujet" : il s'agit des sujets de tests. Ils n'interagiront que passivement avec le système, c'est-à-dire qu'ils n'auront pas la main dessus.

1.3.3 fonctionnalités et structure générale du système

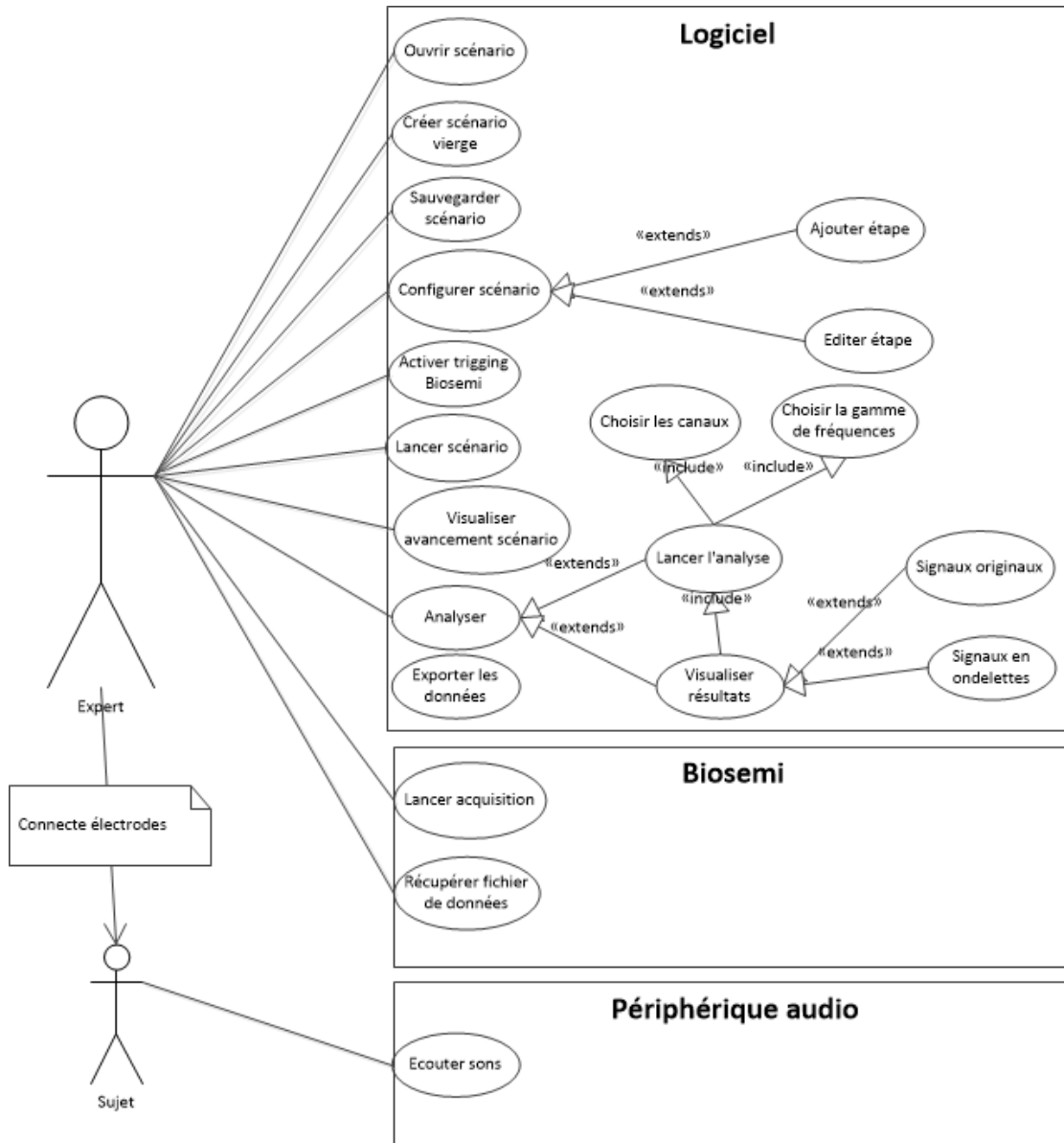


FIGURE 1.3 – Diagramme général des cas d'utilisation

Comme l'indique le diagramme ci-dessous, voici les différentes interactions que les utilisateurs auront avec le système :

- L'expert pourra effectuer les actions suivantes :
 - avec le logiciel :
 - ouvrir un fichier scénario déjà configuré
 - créer un nouveau scénario vierge qu'il lui faudra configurer à partir de zéro
 - configurer un scénario :
 - ajouter des étapes

- configurer des étapes (fréquences et amplitude des ondes pendant l'étape, et durée de l'étape)
- activer le triggering vers Biosemi
- lancer le scénario
- visualiser l'avancement du scénario (barre de progression, messages)
- choisir les paramètres d'analyse (canaux, gammes de fréquences)
- lancer l'analyse
- visualiser les résultats de l'analyse (signal brut, signal transformé dans le domaine temps-fréquence)
- avec Biosemi :
 - lancer l'acquisition
 - récupérer le fichier de données résultat
- Le sujet portera des électrodes et un casque audio posé par l'expert. Tout ce qu'il aura à faire est d'écouter les sons générés tout au long du scénario.

1.3.4 Contraintes de développement, d'exploitation et de maintenance

Contraintes de développement

- La communication avec Biosemi via le port parallèle se fait obligatoirement en langage C. Les signaux à envoyer doivent respecter le protocole existant pour les triggers.
- La soutenance de projet se déroulant le 5 mai 2013, le produit doit impérativement être livré avant cette date.

1.4 Description des interfaces externes du logiciel

1.4.1 Interfaces matériel/logiciel

Génération de sons

Les signaux sonores doivent sortir de la carte son de l'ordinateur, à destination d'un casque stéréophonique ou d'oreillettes.

Liaison avec le matériel EEG

Le matériel EEG à interfacier est Biosemi. Il faut lui envoyer des signaux de trigger pendant la génération de sons. Cette liaison se fait via le port parallèle de l'ordinateur. Il n'existe pas de protocole pour transmettre via ce port. Il faut affecter les valeurs aux broches manuellement. Seulement cela n'est pas possible directement sur un environnement Win32, que ce soit en C ou en ASM. L'OS bloque l'accès au port car il est considéré comme un port matériel et non de communication. Il faut donc utiliser un pilote. On utilisera le pilote Inpout32.dll qu'on intégrera dans le projet. On choisira le mode SPP (Standard Parallel Port) pour pouvoir avoir 8 bits de données en sortie.

Prise en charge du format de fichiers de Biosemi

Le matériel EEG Biosemi génère en sortie des fichiers d'un certain format. Le logiciel doit être en mesure de convertir ces fichiers en un format exploitable, permettant l'analyse des signaux EEG enregistrés (toujours au sein du logiciel).

1.4.2 Interfaces logiciel/logiciel

Concept de scénario, spécification d'un fichier scénario

Un scénario est une séquence de battements binauraux dont les paramètres évoluent au cours du temps. Voici un exemple simple de scénario, comportant ici 2 étapes :

"Les ondes à émettre sont des sinusoïdes pures. La première étape dure 5 minutes : un son de fréquence 440Hz est envoyé sur le canal gauche, un son de fréquence 444Hz est envoyé sur le canal droit. La deuxième dure 3 minutes : un son dont la fréquence passe linéairement de 440Hz à 438Hz sur le canal gauche, un son dont la fréquence est de 444Hz est envoyé sur le canal droit."

Les experts qui utiliseront le logiciel manipuleront des fichiers scénario. Il s'agit de fichiers XML contenant toutes les informations sur le scénario. Ils sont générés automatiquement par le système lors des opérations de création et sauvegarde.

Voici la structure que doit avoir un fichier scénario (voir page suivante).

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <title>ExempleScenario1</title>
  <description>Ceci est un exemple de scenario</description>
  <steps>
    <step id="1">
      <leftFreq>
        <from>440</from>
        <to>440</to>
      </leftFreq>
      <rightFreq>
        <from>444</from>
        <to>444</to>
      </rightFreq>
      <amplitude>
        <from>100</from>
        <to>100</to>
      </amplitude>
      <duration>5</duration>
    </step>
    <step id="2">
      <leftFreq>
        <from>440</from>
        <to>440</to>
      </leftFreq>
      <rightFreq>
        <from>444</from>
        <to>442</to>
      </rightFreq>
      <amplitude>
        <from>100</from>
        <to>100</to>
      </amplitude>
      <duration>10</duration>
    </step>
  </steps>
</scenario>
```

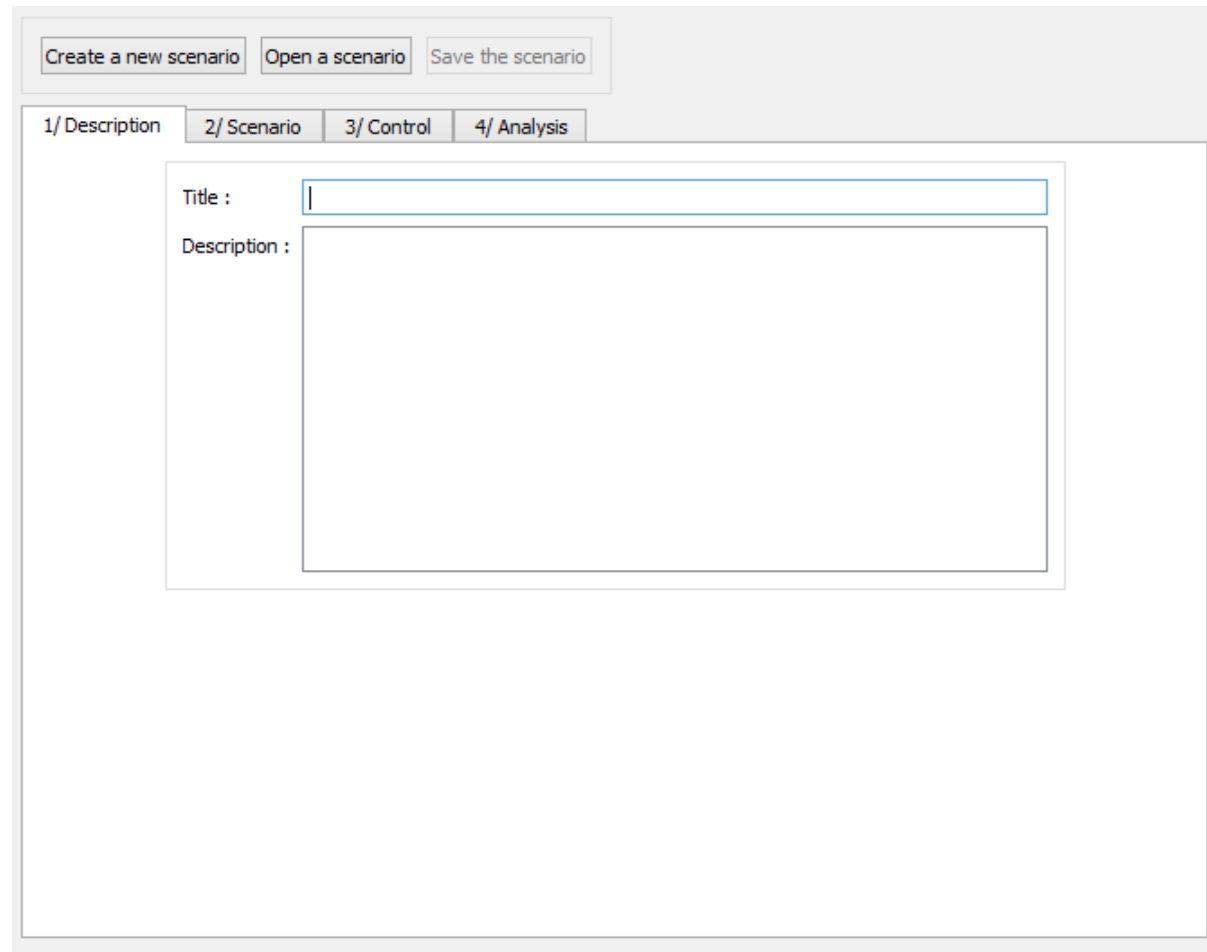
1.4.3 Interfaces homme/machine

Le logiciel tend à une navigation guidée afin de faciliter sa prise en main. Chaque étape du processus de test qu'un expert voudra réaliser est résumée dans un onglet numéroté de la fenêtre principale. Cela évite que l'expert ne se perde dans des menus. Voici ces différents onglets :

1. Description du scénario
2. Conception du scénario
3. Contrôle du scénario (paramétrage du triggering, lancement, visualisation de l'avancement)
4. Analyse

Description du scénario

Cet onglet permet de définir le titre du scénario et d'y entrer un texte pour le décrire.



The screenshot shows a software window with a title bar containing three buttons: "Create a new scenario", "Open a scenario", and "Save the scenario". Below the title bar is a tabbed interface with four tabs: "1/ Description", "2/ Scenario", "3/ Control", and "4/ Analysis". The "1/ Description" tab is active. Inside this tab, there is a form with two fields: "Title :" followed by a single-line text input field, and "Description :" followed by a larger multi-line text area.

FIGURE 1.4 – IHM de l'onglet n°1

Conception du scénario

Dans cet onglet, l'expert crée son scénario en ajoutant des étapes. Ceci se fait en cliquant sur le bouton "Add Step". Une fenêtre s'affiche ensuite, permettant d'éditer les paramètres des ondes sonores pour l'étape (voir page suivante). Lorsqu'elle est validée, la représentation graphique de l'évolution de la fréquence des ondes sonores au cours du temps est mise à jour (cadre de droite). L'étape est ajoutée dans le cadre de gauche. Il est possible de modifier une étape en y effectuant un clic gauche, et d'intervertir les différentes étapes entre elles.

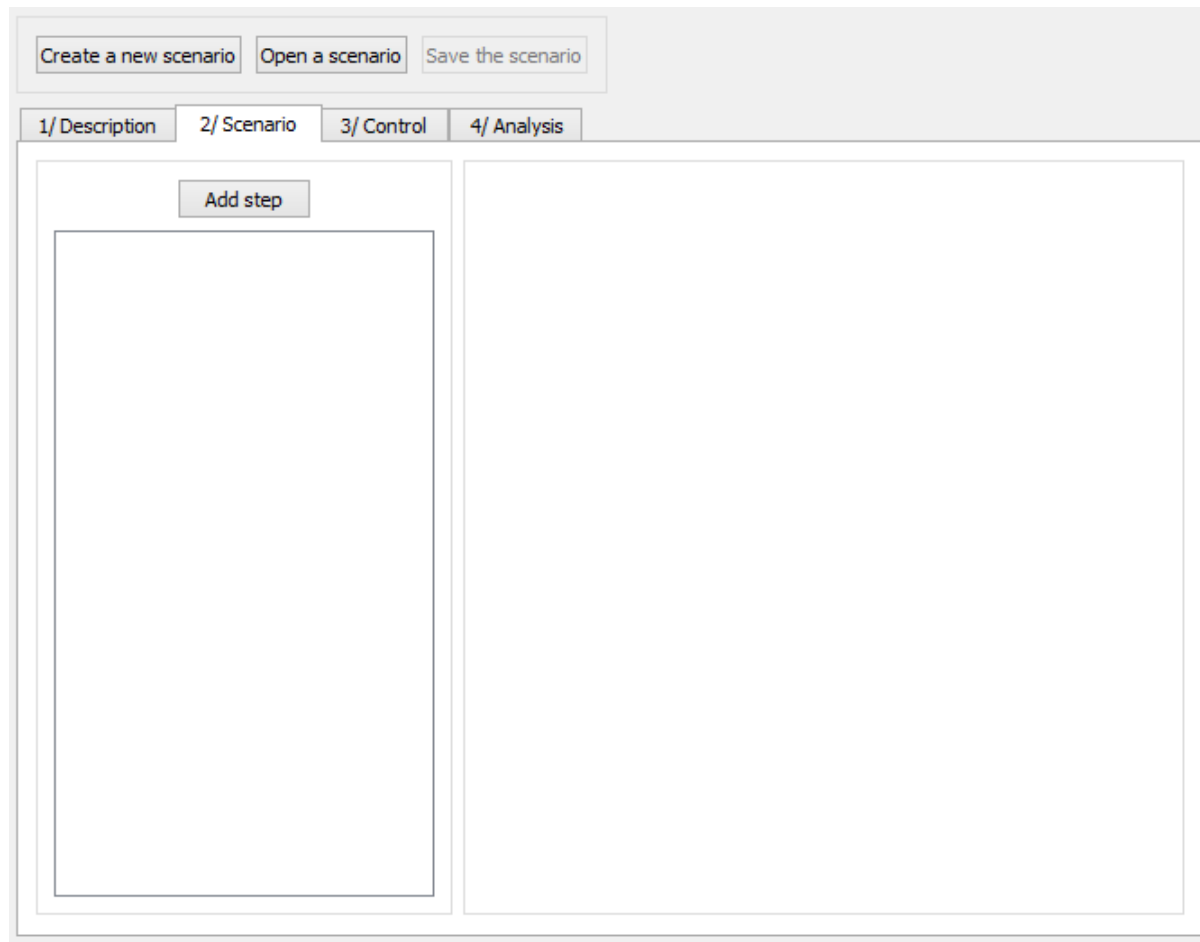
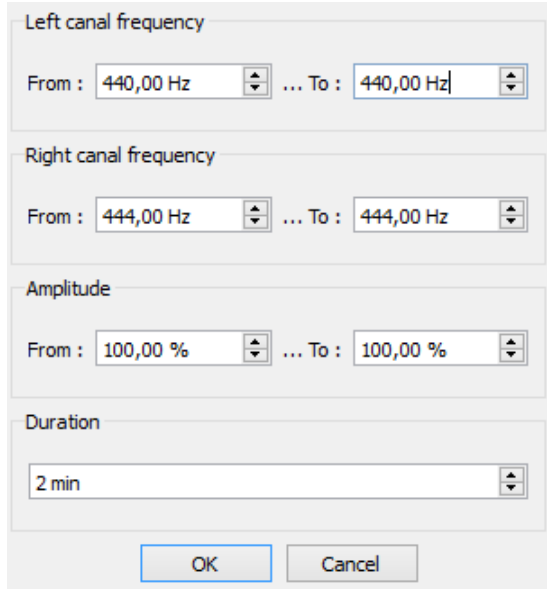


FIGURE 1.5 – IHM de l'onglet n°2

La fenêtre d'ajout ou de modification d'étape est constituée comme ceci (de haut en bas) :

- Réglage de la fréquence d'onde en Hertz pour le canal gauche, pour le début de l'étape et la fin de l'étape.
- Réglage de la fréquence d'onde en Hertz pour le canal droit, pour le début de l'étape et la fin de l'étape.
- Réglage de l'amplitude du signal, pour le début de l'étape et la fin de l'étape.
- (non inclus dans l'IHM pour l'instant) Réglage de l'amplitude du bruit
- Réglage de la durée de l'étape, en minutes.



Left canal frequency

From : 440,00 Hz ... To : 440,00 Hz

Right canal frequency

From : 444,00 Hz ... To : 444,00 Hz

Amplitude

From : 100,00 % ... To : 100,00 %

Duration

2 min

OK Cancel

FIGURE 1.6 – IHM de la fenêtre d'ajout d'étape de scénario

Contrôle du scénario

Cet onglet sert principalement à lancer le scénario. Cela comprend la génération du son et sa lecture via le périphérique, ainsi que le triggering vers Biosemi si celui-ci est activé (case à cocher pour activer ou non le triggering). Une barre de progression permet de visualiser l'avancement du scénario, et une zone de texte affiche les messages correspondant aux différentes actions effectuées (changement d'étape, envoi de trigger, erreurs éventuelles...). Il est aussi possible de tout arrêter si un problème a été rencontré lors du test.

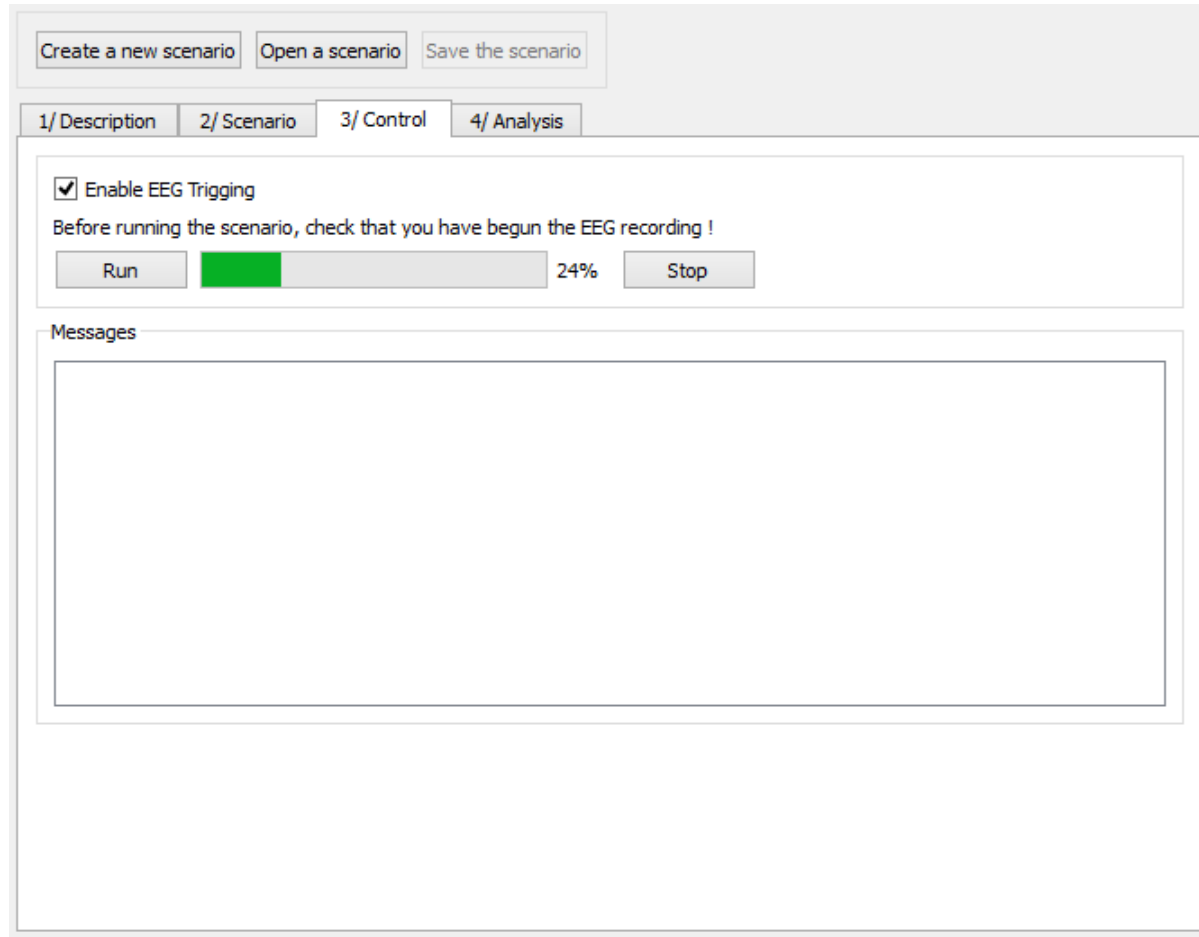
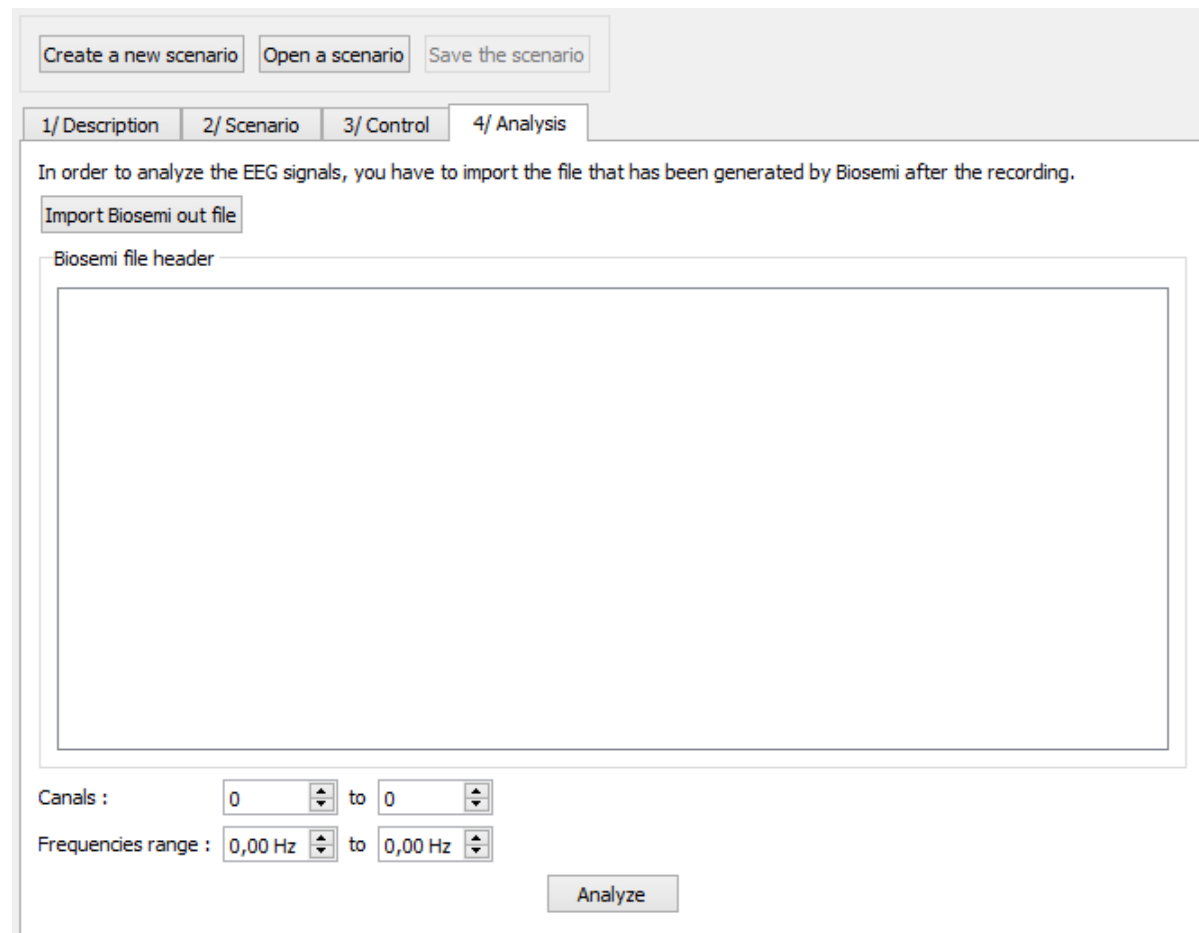


FIGURE 1.7 – IHM de l'onglet n°3

Analyse

Cet onglet est à utiliser pour effectuer l'analyse des signaux enregistrés par le matériel EEG. Un premier bouton sert à importer le fichier généré par Biosemi. On peut ensuite visualiser les données fournies par l'entête du fichier. On peut ensuite choisir les différentes voies à analyser (correspondant aux signaux captés par chaque électrode), ainsi que la gamme de fréquence sur laquelle on veut se concentrer.

Remarque importante : sur la maquette ci-dessous, le choix des canaux se fait en choisissant un intervalle de numéros de canaux. La maquette finale comportera en réalité un bouton dont l'action est d'ouvrir une fenêtre affichant la liste des noms des canaux disponibles (références des électrodes de Biosemi, par exemple Fp1, AF3, AF7...) et qu'il faudra sélectionner.



Create a new scenario Open a scenario Save the scenario

1/ Description 2/ Scenario 3/ Control 4/ Analysis

In order to analyze the EEG signals, you have to import the file that has been generated by Biosemi after the recording.

Import Biosemi out file

Biosemi file header

Canals : 0 to 0

Frequencies range : 0,00 Hz to 0,00 Hz

Analyze

FIGURE 1.8 – IHM de l'onglet n°4

Un clic sur le bouton "Analyze" ouvre une nouvelle fenêtre. En haut de cette fenêtre s'affiche une barre de progression indiquant l'avancement de l'analyse (qui peut prendre du temps). On peut ensuite visualiser les signaux bruts, ainsi que les résultats de l'analyse (évolution de l'énergie de la gamme de fréquence en fonction du temps). Sur chaque graphique, des barres verticales représentent les triggers qui ont été envoyés. Enfin, le dernier bouton permet d'exporter les données.

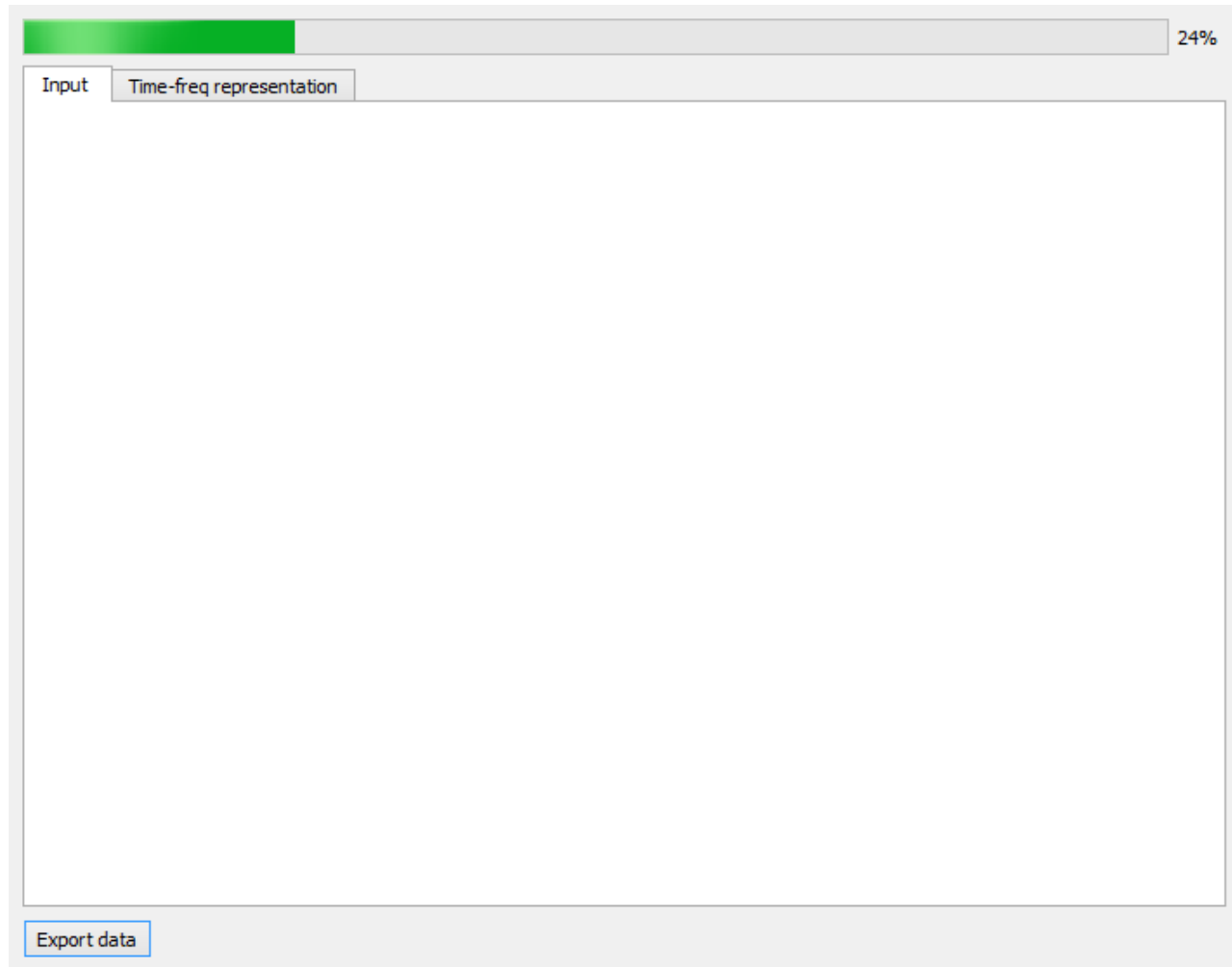


FIGURE 1.9 – IHM de la fenêtre de résultats d'analyse

1.5 Architecture générale du système

Cette section présente une première réflexion sur la structure interne du programme, avec les principaux composants. Il ne s'agit pas là de l'analyse complète du projet avec la totalité des classes, attributs et méthodes, mais permet d'avoir une première vision de l'architecture générale qui sera implémentée.

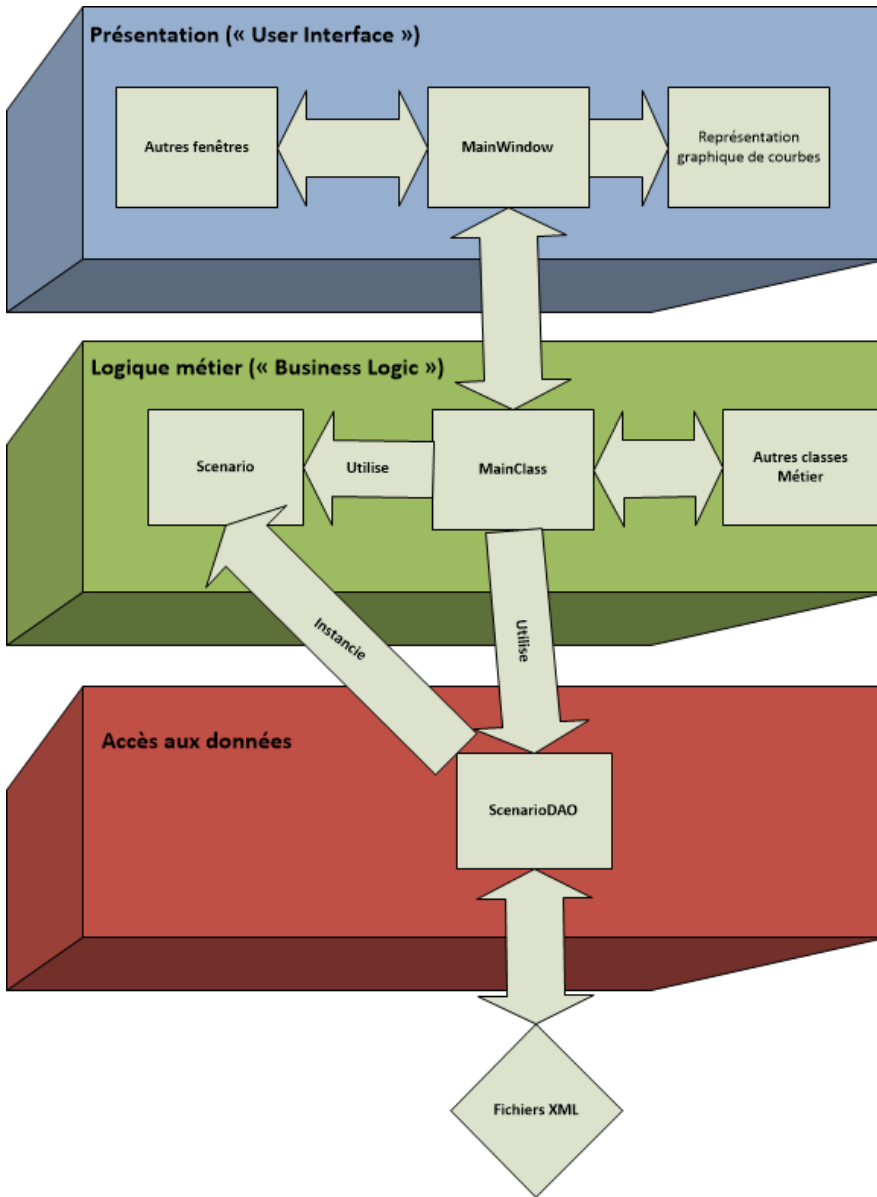


FIGURE 1.10 – Architecture générale du logiciel. Basée sur le modèle 3-Tiers.

Le choix s'est porté sur une architecture 3-tiers. Celle-ci est donc répartie sur trois couches :

- la couche Présentation (ou "User Interface"). Il s'agit de l'interface graphique. C'est la couche arrivant directement à l'utilisateur : c'est elle qui lui transmet les informations venant des couches supérieures, et c'est elle qui intercepte en premier les actions de l'utilisateur.
- la couche Logique Métier (ou "Business Logic"). Partie fonctionnelle du logiciel, elle traite les données et les transmet aux deux autres couches.
- la couche Accès aux Données (ou "Data Access"). C'est la partie qui accède aux données persistantes. Dans notre cas, ces données sont stockées dans des fichiers XML. En général cette couche est plus importante lorsqu'il s'agit de s'interfacer avec des bases de données.

1.5.1 Description des composants

Couche Présentation

- **MainWindow** est la fenêtre principale de l'application.
- **StepDialog** est la boîte de dialogue permettant de configurer une étape d'un scénario.
- **AnalysisWindow** est la fenêtre permettant d'afficher le résultat d'une analyse.
- **ScenarioGraphManager** permet de dessiner la représentation graphique d'un scénario.
- **AnalysisGraphManager** permet de dessiner la représentation graphique du résultat d'une analyse.

Couche Logique Métier

- **MainClass** est la classe de démarrage de l'application. C'est elle qui coordonne tous les objets. Elle joue le rôle de médiateur, comme nous le verrons dans la partie suivante.
- **SoundManager** prend en charge la gestion du son.
- **TriggingManager** est destinée à la communication avec Biosemi (envoi de triggers) via le port parallèle.
- **BiosemiFileManager** prend en charge la lecture de fichiers BDF/EDF (fichiers Biosemi) et leur conversion en données lisibles pour leur utilisation future.
- **Analyzer** se charge de l'analyse du signal.
- **AnalysisExporter** prend en charge l'exportation des données d'analyse.
- **Scenario** représente un scénario. Il s'agit de la représentation en objets C++ des données contenues dans les fichiers XML.

Couche Accès aux Données

- **ScenarioDAO** pour "Scenario Data Access Object". Cette classe extrait les données à partir du fichier XML et les convertit en objets de type Scenario.

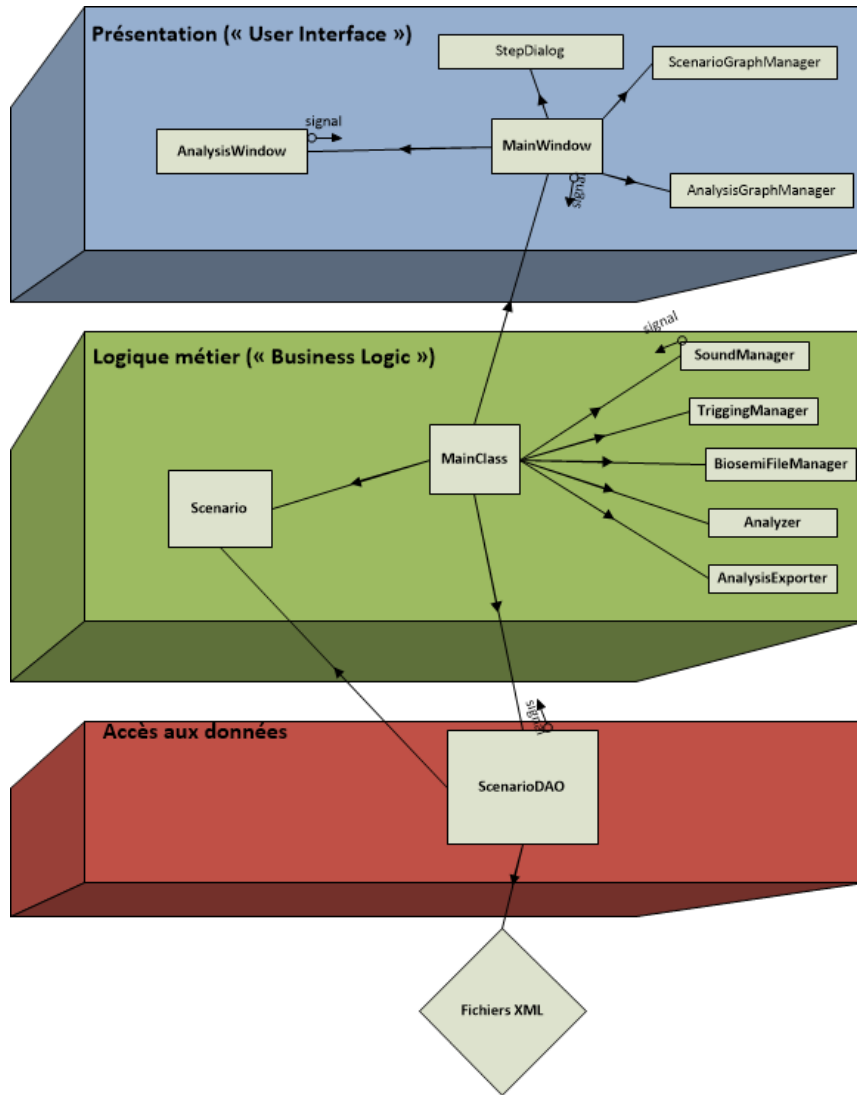


FIGURE 1.11 – Architecture générale du logiciel avec sens et types de communication. Les flèches simples représentent des appels directs ou instanciations. Les petites flèches représentent la capacité d’un objet à émettre un signal, que d’autres objets peuvent recevoir, grâce au mécanisme des signaux et des slots de Qt (implémentation du patron de conception Observable)

On utilise le mécanisme des signaux et des slots afin d’effectuer une synchronisation entre les objets : ceux-ci peuvent s’envoyer des messages sans se connaître. C’est le principe du patron de conception "Observable". De plus, certains objets ne communiquent pas directement entre eux, ils doivent passer par un intermédiaire, un médiateur, `MainClass`, qui s’occupe de rediriger les messages après avoir analysé la situation. C’est le principe du patron de conception "Mediator".

1.6 Description des fonctionnalités

Cette section liste et décrit l'ensemble des fonctionnalités du système en exposant l'interaction entre les différents composants. Il ne s'agit là que d'une pré-analyse, qui sera complétée dans le cahier d'analyse.

1.6.1 Fonction [Ouvrir scénario]

Identification de la fonction

L'expert doit pouvoir ouvrir un fichier représentant un scénario créé auparavant. L'action peut être réalisée à tout moment, et consiste en la sélection d'un fichier xml dans l'arborescence des dossiers de l'ordinateur. On a recours à cette fonctionnalité en cliquant sur le bouton "Open scenario" dans la partie haute de la fenêtre principale.

Description de la fonction

- **Entrées-Sorties**
 - Entrée : aucune
 - Sortie : fichier XML. Ce fichier sera parsé afin de créer un objet de type Scenario.
- **Composants en interaction**
 - Objet Scenario : le résultat du parsing du fichier XML sera stocké dans un objet de type Scenario.
- **Traitement de la fonction**
 1. Affichage de la fenêtre de dialogue d'ouverture de fichier XML
 2. Sélection du fichier XML
 3. Parsing du fichier
 4. Création d'un objet de type Scenario
- **Gestion des erreurs**
 - Exception dans le cas de balises non reconnues dans le cadre des scénarios
 - Exception dans le cas de valeurs non valides

1.6.2 Fonction [Créer scénario vierge]

Identification de la fonction

L'expert doit pouvoir créer un nouveau scénario vide. Il accède à cette fonction par l'intermédiaire du bouton "Create new scenario" dans la partie haute de la fenêtre principale. Il pourra par la suite enregistrer son scénario via la fonction [Sauvegarder scénario].

Description de la fonction

- **Entrées-Sorties**
 - Entrée : aucune
 - Sortie : aucune
- **Composants en interaction**
 - Objet Scenario : cet objet sera réinitialisé.
- **Traitement de la fonction**
 1. Si un scénario est déjà ouvert, demande de confirmation si ce dernier n'a pas été sauvegardé, afin d'éviter de perdre le travail en cours.
 2. Réinitialisation de l'objet Scenario

1.6.3 Fonction [Sauvegarder scénario]

Identification de la fonction

L'expert doit pouvoir sauvegarder le scénario qu'il a créé ou modifié. La sauvegarde consiste en la création ou la mise à jour d'un fichier XML représentant un scénario. L'action peut être réalisée lorsqu'un scénario est ouvert, et consiste en la sélection d'un emplacement dans l'arborescence des dossiers de l'ordinateur, pour la sauvegarde du fichier XML. On a recours à cette fonctionnalité en cliquant sur le bouton "Save scenario" dans la partie haute de la fenêtre principale.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : objet de type Scenario.
 - Sortie : aucune
- **Composants en interaction**
 - Objet Scenario : la fonction analysera l'objet Scenario afin de générer le fichier XML correspondant.
- **Traitement de la fonction**
 1. Affichage de la fenêtre de dialogue d'enregistrement de fichier XML
 2. Génération du fichier XML

1.6.4 Fonction [Configurer scénario]

Identification de la fonction

L'expert doit pouvoir configurer un scénario, c'est-à-dire :

- ajouter des étapes au scénario
- éditer les paramètres d'onde pour chaque étape du scénario

Pour ce faire, il faudra entrer dans l'onglet 2 de la fenêtre principale.

Les trois sous-fonctionnalités citées ci-dessus seront précisées dans les fonctions [Ajouter étape] et [Configurer étape].

1.6.5 Fonction [Ajouter étape]

Identification de la fonction

L'expert doit ajouter insérer au moins une étape dans son scénario afin qu'il soit utilisable. Cela se fait à partir du bouton "Add Step" de l'onglet 3 de la fenêtre principale.

Description de la fonction

- **Entrées-Sorties**
 - Entrée : aucune.
 - Sortie : objet de type Step. Cet objet sera ajouté à une liste, contenue dans l'objet Scenario.
- **Composants en interaction**
 - Objet Step : création d'un objet de type Step.
 - Objet Scenario : ajout de cet objet Step la liste d'étapes de l'objet Scenario.
- **Traitement de la fonction**
 1. Affichage de la fenêtre d'édition d'étape
 2. Edition de l'étape (voir fonction [Editer étape])
 3. Création d'un objet de type Step
 4. Ajout de l'objet Step à la liste d'étapes de l'objet Scenario

1.6.6 Fonction [Configurer étape]

Identification de la fonction

L'expert doit pouvoir éditer les paramètres des étapes de son scénario, que ce soit lors de l'ajout d'une nouvelle étape ou lors de la modification d'une étape existante. Pour entrer dans le dernier cas, il faut cliquer sur une étape dans la liste des étapes sur la partie gauche de l'onglet 3 de la fenêtre principale. La fenêtre d'édition comporte plusieurs boîtes numériques permettant de renseigner les différents paramètres de l'étape (durée, fréquences des ondes...).

Description de la fonction

- **Entrées-Sorties**
 - Entrées :
 - Durée de l'étape en minutes
 - Fréquences sur les canaux gauche et droit, au début et à la fin de l'étape (4 valeurs au total)
 - Amplitude du signal, au début et à la fin de l'étape (2 valeurs au total)
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Step : édition d'un objet de type Step
- **Traitement de la fonction**
 1. Edition des différents paramètres
 2. Edition d'un objet de type Step
- **Gestion des erreurs**
 - Exceptions dans le cas de valeurs non valides pour les paramètres renseignés

1.6.7 Fonction [Activer triggering Biosemi]

Identification de la fonction

L'expert doit pouvoir spécifier au logiciel s'il souhaite activer l'envoi de signaux (triggers) au matériel Biosemi. Cela se fait à partir de l'onglet 3 de la fenêtre principale (**à compléter**).

Description de la fonction

- **Entrées-Sorties**
 - Entrées : booléen spécifiant la volonté d'envoyer des triggers.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet TriggeringManager : édition de l'objet TriggeringManager, qui décrit les actions à effectuer sur le matériel Biosemi et qui se charge de la communication via le port parallèle.
- **Traitement de la fonction**
 - Edition de l'objet TriggeringManager

1.6.8 Fonction [Lancer scénario]

Identification de la fonction

En appuyant sur le bouton "Run", l'expert peut démarrer le scénario, c'est-à-dire le démarrage de l'envoi des triggers et de la génération/lecture sonore.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Scenario : lecture du scénario et de ses étapes afin que les objets PTLManager et SoundManager effectuent les actions appropriées au moment opportun.
 - Objet PTLManager : communication avec le port parallèle pour l'envoi des triggers.
 - Objet SoundManager : génération et lecture du son en temps réel.
- **Traitement de la fonction** En parallèle :
 - Lecture du scénario
 - Envoi des triggers
 - Génération du son
 - Lecture du son
- **Gestion des erreurs**
 - Exception dans le cas de l'indisponibilité du périphérique audio (pilote non installé...)
 - Exception dans le cas de l'indisponibilité du matériel Biosemi
 - Exception dans le cas d'un dysfonctionnement ou d'une perte de liaison avec le matériel Biosemi

1.6.9 Fonction [Visualiser avancement scénario]

Identification de la fonction

Pendant le déroulement d'un scénario, le logiciel doit afficher des messages à chaque événement : nouveau trigger envoyé, erreur de communication, erreur de périphérique audio... De plus, une barre de progression permettra de connaître l'avancement du scénario en fonction du temps.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Scenario.
 - Objet PTLManager.
 - Objet SoundManager.
- **Traitement de la fonction** En parallèle :
 - Mise à jour de la barre de progression
 - Affichage des messages

1.6.10 Fonction [Choisir les canaux]

Identification de la fonction

L'expert doit pouvoir choisir le ou les canaux dont il veut analyser le signal.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : indices des canaux de début et de fin (exemple : 5 - 45 pour analyser le signal des canaux 5 à 45, ou encore 6-6 pour n'analyser que le canal 6).
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
- **Traitement de la fonction**
 - Renseignement des paramètres à l'objet Analyzer.

1.6.11 Fonction [Choisir la gamme de fréquences]

Identification de la fonction

L'expert doit pouvoir choisir l'intervalle de fréquences sur lesquelles il veut analyser le signal.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : les fréquences de début et de fin (exemple : 0.5 - 5 pour ne traiter que l'intervalle entre 0.5Hz et 5Hz).
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
- **Traitement de la fonction**
 - Renseignement des paramètres à l'objet Analyzer.

1.6.12 Fonction [Lancer l'analyse]

Identification de la fonction

L'appui sur le bouton "Analyse" déclenche le traitement du signal ou des signaux. Pour chaque canal, et pour la gamme de fréquences choisies, le signal est transformé avec un algorithme de transformée rapide en ondelettes. Cela permet de visualiser l'évolution de l'énergie de la gamme de fréquences en fonction du temps.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
- **Traitement de la fonction** Pour chaque signal :
 1. Prétraitement
 2. Transformation en ondelettes
- **Gestion des erreurs**
 - Exception dans le cas de variables erronées (fréquences, canaux).

1.6.13 Fonction [Visualiser signaux bruts]

Identification de la fonction

L'expert doit pouvoir visualiser les signaux tels qu'ils ont été enregistrés.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
 - Objet AnalysisGraphManager.
- **Traitement de la fonction** Pour chaque signal :
 - Affichage de la courbe du signal

1.6.14 Fonction [Visualiser signaux transformés]

Identification de la fonction

L'expert doit pouvoir visualiser les signaux transformés, c'est-à-dire dans le domaine temps-fréquence (évolution de l'énergie d'une gamme de fréquences en fonction du temps).

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
 - Objet AnalysisGraphManager
- **Traitement de la fonction** Pour chaque signal
 - Affichage de la courbe du signal transformé

1.6.15 Fonction [Exporter les données]

Identification de la fonction

L'expert doit pouvoir exporter les données d'analyse.

Description de la fonction

- **Entrées-Sorties**
 - Entrées : aucune.
 - Sortie : aucune.
- **Composants en interaction**
 - Objet Analyzer.
 - Objet AnalysisExporter.
- **Traitement de la fonction** Pour chaque signal :
 - enregistrement des données brutes
 - enregistrement des données transformées

1.7 Utilisation des fonctionnalités au cours du temps

Le diagramme d'activité suivant permet de mieux visualiser comment le système sera utilisé par l'expert. Il ne recouvre pas tous les scénarios possibles mais est assez explicite pour permettre la bonne compréhension générale des étapes d'utilisation du logiciel.

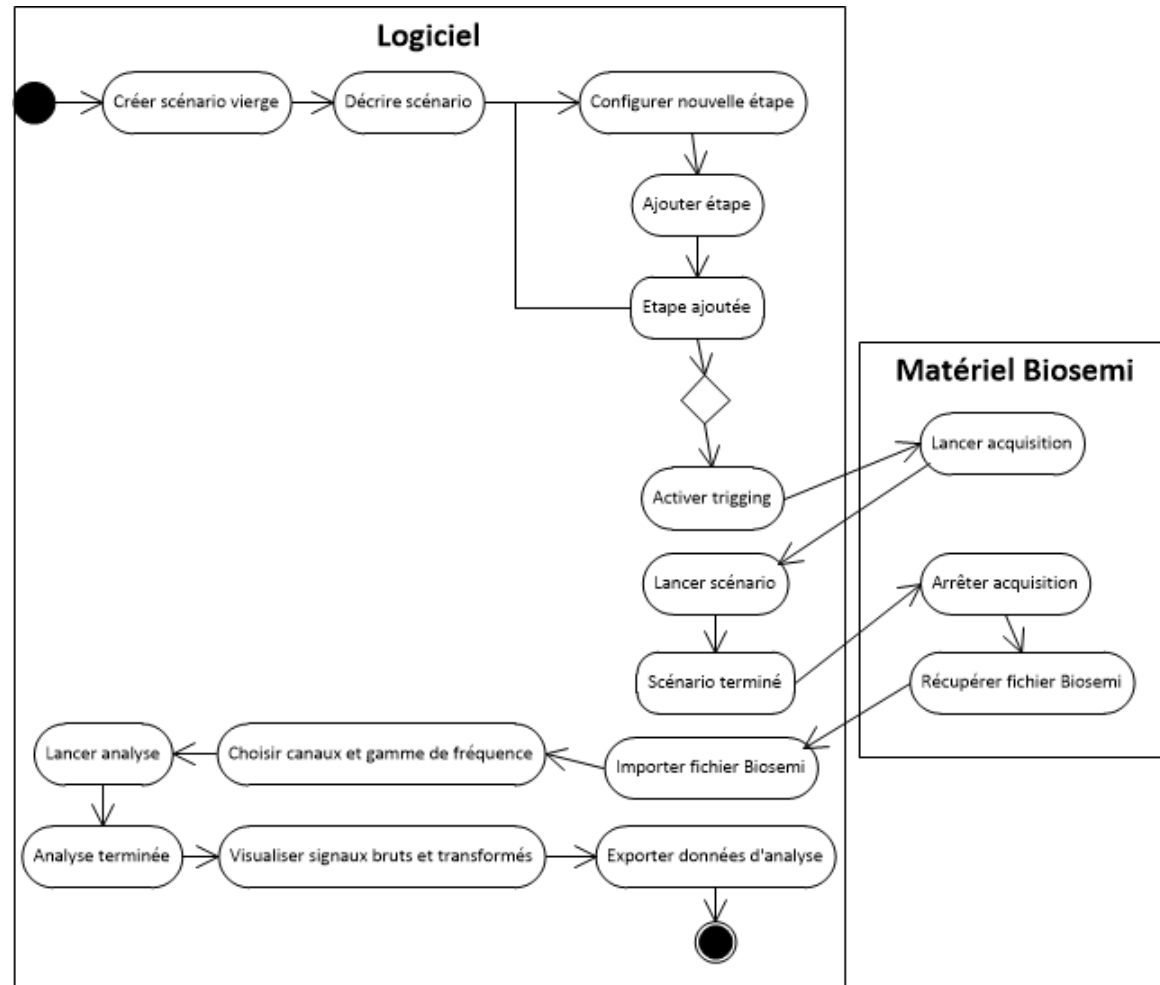


FIGURE 1.12 – Diagramme d'activité détaillant un scénario complet d'utilisation du système.

1.8 Conditions de fonctionnement

Pour utiliser le logiciel dans de bonnes conditions, il est impératif d'être muni d'un casque stéréophonique ou d'oreillettes, de bonne qualité.

Il n'est pas obligatoire d'utiliser de matériel EEG pour faire fonctionner le logiciel. Si aucun matériel n'est utilisé, il faut désactiver le triggering. De plus le module d'analyse ne pourra être utilisé.

Le matériel EEG ciblé est Biosemi et aucun autre. Dans le cas d'une utilisation avec ce matériel, il faut établir une liaison par câble PTL (port série) et activer le triggering. Il est aussi important de veiller à lancer l'acquisition en même temps que le scénario.

1.8.1 Performances

Pour toutes les fonctionnalités correspondant aux onglets 1 à 4 (tous les onglets sauf le dernier qui correspond à l'analyse), le temps de réponse doit être le plus court possible (quasi-instantané).

Pour les fonctionnalités correspondant au dernier onglet, le temps de réponse peut être plus long, notamment lors de l'étape de traitement du signal (pouvant durer plusieurs dizaines de secondes). Pour l'étape d'importation de fichiers Biosemi, on acceptera un temps de réponse de l'ordre de quelques secondes.

1.8.2 Capacités

Le logiciel pourra prendre en compte au maximum 64 voies d'acquisition lors du paramétrage du matériel Biosemi.

1.8.3 Contrôlabilité

L'onglet "Contrôle" permettra de visualiser les différents événements au cours de l'exécution du scénario. Les messages qui y seront affichés seront de plus stockés dans un fichier de log.

1.8.4 Sécurité

Aucune sécurité du point de vue confidentialité (contrôle d'accès des utilisateurs) n'est prévue pour le logiciel.

Plan de développement

2.1 Méthode de planification adoptée

La technique adoptée est la suivante :

1. Découpage du projet en tâches. Il peut s'agir de tâches de spécification, de rédaction, de modélisation, de développement ou de production. Chacune des tâches de développement est en général une fonctionnalité du logiciel.
2. Estimation de la complexité de chaque tâche. La complexité est un indicateur arbitraire, utilisé dans les méthodes agiles lors de séances de Planning Poker par exemple, où les membres de l'équipe donnent rapidement leur sentiment sur la complexité de la tâche concernée.
3. Calcul de la complexité totale du projet. On effectue la somme de toutes les complexités.
4. Calcul de la durée totale de travail en jours. Ce calcul est simple car la fin du projet est fixée.
5. Calcul du ratio 'Durée totale de travail / Complexité totale'.
6. Calcul des estimations de durée de travail pour chaque tâche en jours/homme, à partir du ratio précédent.
7. Répartition des tâches sur 3 périodes ou sprints (hors périodes de spécification et d'analyse). La complexité de chaque sprint doit être environ égale. Un sprint est un cycle logiciel en lui-même, incluant des phases de développement, de tests et de déploiement.

Remarque : les calculs de complexité et de charge ne sont faits que sur la période du 22/01 au 22/04, correspondant à la durée totale des 3 sprints. Les phases de spécification et d'analyse se déroulent avant cette période et ne sont pas estimées. La mise en production aura lieu à l'issue cette période, le projet se terminant le 03/05.

2.2 Calcul du ratio 'Durée totale de travail / Complexité totale'

Comme nous le verrons à la section suivante, des points de complexité ont été attribués à chaque tâche. La somme de ces points donne la complexité totale du projet. Nous avons aussi besoin de la durée totale de travail (à raison de 3 jours par semaine). Nous excluons de ces calculs les phases de spécification et d'analyse.

Complexité totale = 62 points

Durée totale de travail = 39 jours

Ratio = $39 / 62 = 0,63$ jour / point

Grâce à ce ratio, l'estimation en jours/homme de la charge de chaque tâche pourra être indiquée dans la section suivante.

Par exemple, une tâche de complexité 6 aura une charge de $6 * 0,63 = 3,5$ jours/homme environ.

2.3 Découpage du projet en tâches

2.3.1 Tâches de spécifications

Tâche 'Documentation générale'

Description Recherches bibliographiques au sujet des battements binauraux

Tâche 'Documentation technique'

Description Recherches au sujet de l'analyse des signaux, Biosemi, la génération de sons. Recherches de bibliothèques pour la prise en charge de ces derniers, ainsi que pour l'interface graphique.

Tâche 'Documentation légale'

Description Recherches sur les licences utilisées par les bibliothèques.

Tâche 'Première modélisation de l'architecture générale'

Description Modélisation de l'architecture générale du système. Etablissement d'un diagramme général des objets afin de distinguer les principales classes du programme avec certaines interactions.

Livable Le diagramme doit être livré avec le Cahier de Spécifications.

Tâche 'Rédaction du Cahier de Spécifications, version 1'

Description Rédaction de la première version du Cahier de Spécifications. De nombreuses études sont faites pendant cette rédaction. En outre, il est nécessaire de réaliser plusieurs rencontres avec l'équipe.

Livable Cette version doit être fournie à l'ensemble de l'équipe ainsi qu'au Responsable des PFE le 3 décembre au plus tard.

Tâche 'Finalisation du Cahier de Spécifications'

Description Rédaction de la version finale du Cahier de Spécifications.

Livable Cette version doit être fournie 7 janvier au plus tard.

Tâche 'Réalisation de l'Interface Homme-Machine'

Description Conception et mise en place de l'IHM avec Qt Creator.

Livable Doit être terminée en même temps que le Cahier de Spécifications. Possibilités de modification.

2.3.2 Tâches d'analyse et de modélisation

Tâche 'Modélisation de l'architecture du logiciel'

Description Modélisation des classes du programme ainsi que leurs interactions. Diagramme des classes et diagrammes des objets.

Livable Les diagrammes doivent être livrés en même temps que le Cahier d'Analyse.

Tâche 'Rédaction du Cahier d'Analyse Fonctionnelle'

Description Rédaction du Cahier d'Analyse Fonctionnelle.

Livable Le cahier doit être fourni à l'ensemble de l'équipe ainsi qu'au responsable des PFE le 22 janvier.

2.3.3 Tâches de développement

Tâche 'Développement du module de gestion de fichiers'

Description Développement du gestionnaire de fichiers XML (lecture / écriture). Lors de la lecture, enregistrement des données du fichier dans un objet de type Scenario. Lors de l'écriture, processus inverse. Parsing XML.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 4 pts. => Charge : 2,5 jours/homme.

Tâche 'Développement du module scénario'

Description Prise en charge des modifications effectuées dans l'onglet 'Description' au sein de l'objet Scenario en cours. Prise en charge des modifications effectuées dans l'onglet 'Scenario' au sein de l'objet Scenario en cours : gérer les créations d'objets Step et leur intégration au sein de l'objet Scenario.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 4 pts. => Charge : 2,5 jours/homme.

Tâche 'Développement du module d'affichage de scénario'

Description Création / mise à jour de la courbe des fréquences des canaux gauche et droit en fonction du temps.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 6 pts. => Charge : 3,5 jours/homme.

Tâche 'Développement du module audio'

Description Génération et modification du son en temps réel en fonction des paramètres du scénario changeant au cours du temps.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 4 pts. => Charge : 2,5 jours/homme.

Tâche 'Développement de la fonctionnalité d'édition graphique de scénario'

Description Interaction entre la souris et la représentation graphique du scénario entraînant l'édition de ce dernier.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 7 pts. => Charge : 4,5 jours/homme.

Tâche 'Développement du module de communication'

Description Gestion de la communication entre le logiciel et le matériel via le port Biosemi (envoi de signaux).

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 7 pts. => Charge : 4,5 jours/homme.

Tâche 'Développement du module de gestion de fichiers Biosemi'

Description Importation de fichiers BDF (Biosemi Data File) au sein du logiciel pour analyse.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 7 pts. => Charge : 4,5 jours/homme.

Tâche 'Développement du module de prétraitement des signaux'

Description Suppression du bruit.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 7 pts. => Charge : 4,5 jours/homme.

Tâche 'Développement du module de traitement des signaux'

Description Transformation du signal en ondelettes sur la bande de fréquences indiquée. Applicable à plusieurs signaux dans le cas de l'analyse de plusieurs canaux

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 7 pts. => Charge : 4,5 jours/homme.

Tâche 'Développement du module d'affichage des signaux'

Description Représentation graphique du signal original (ou des signaux originaux). Représentation graphique du signal transformé (ou des signaux transformés).

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 6 pts. => Charge : 3,5 jours/homme.

Tâche 'Développement du module d'exportation de données'

Description Enregistrement des signaux originaux et transformés dans des fichiers XSL.

Cycle de vie Tests fonctionnels effectués parallèlement au développement.

Estimation de charge Complexité : 3 pts. => Charge : 2 jours/homme.

2.3.4 Tâches de mise en production

Tâche 'Rédaction du manuel utilisateur'

Description Rédaction du manuel utilisateur, décrivant toutes les étapes à réaliser pour utiliser correctement le logiciel. Il doit être écrit dans un langage compréhensible et le plus clair possible, dans un registre en accord avec la classe d'utilisateurs ciblée.

Tâche 'Déploiement'

Description Installation du logiciel sur la machine cible. Interfaçage avec le matériel.

2.3.5 Tâches annexes

Tâche 'Rédaction du rapport'

Description Rédaction du rapport de PFE.

Livrable Ce document doit être fourni à l'ensemble de l'équipe ainsi qu'au responsable de PFE avant le 3 mai.

Tâche 'Etablissement de la bibliographie'

Description Rassemblement de tous les articles et documents utilisés pour la réalisation de ce projet.

Livrable La bibliographie doit être fournie avec le rapport.

2.4 Planning

2.4.1 Chronologie générale du projet

La figure suivante représente l'avancement du projet dans ses grandes phases :

1. Phase de Spécifications, à l'issue de laquelle le Cahier de Spécifications doit être livré.
2. Phase d'Analyse et de Modélisation, à l'issue de laquelle le Cahier d'Analyse Fonctionnelle doit être livré.
3. Phase de Développement, à l'issue duquel le produit doit être terminé.
4. Phase de Mise en Production, à l'issue de laquelle le produit doit être livré et installé. Un manuel d'utilisation devra être fourni.

De plus, le rapport de PFE sera livré au terme de la dernière phase.



FIGURE 2.13 – Chronologie générale du projet

2.4.2 Planning détaillé du projet à partir du sprint 1

La figure suivante est un diagramme de Gantt, présentant l'ordonnancement des tâches à partir du sprint 1, jusqu'à la fin du projet.

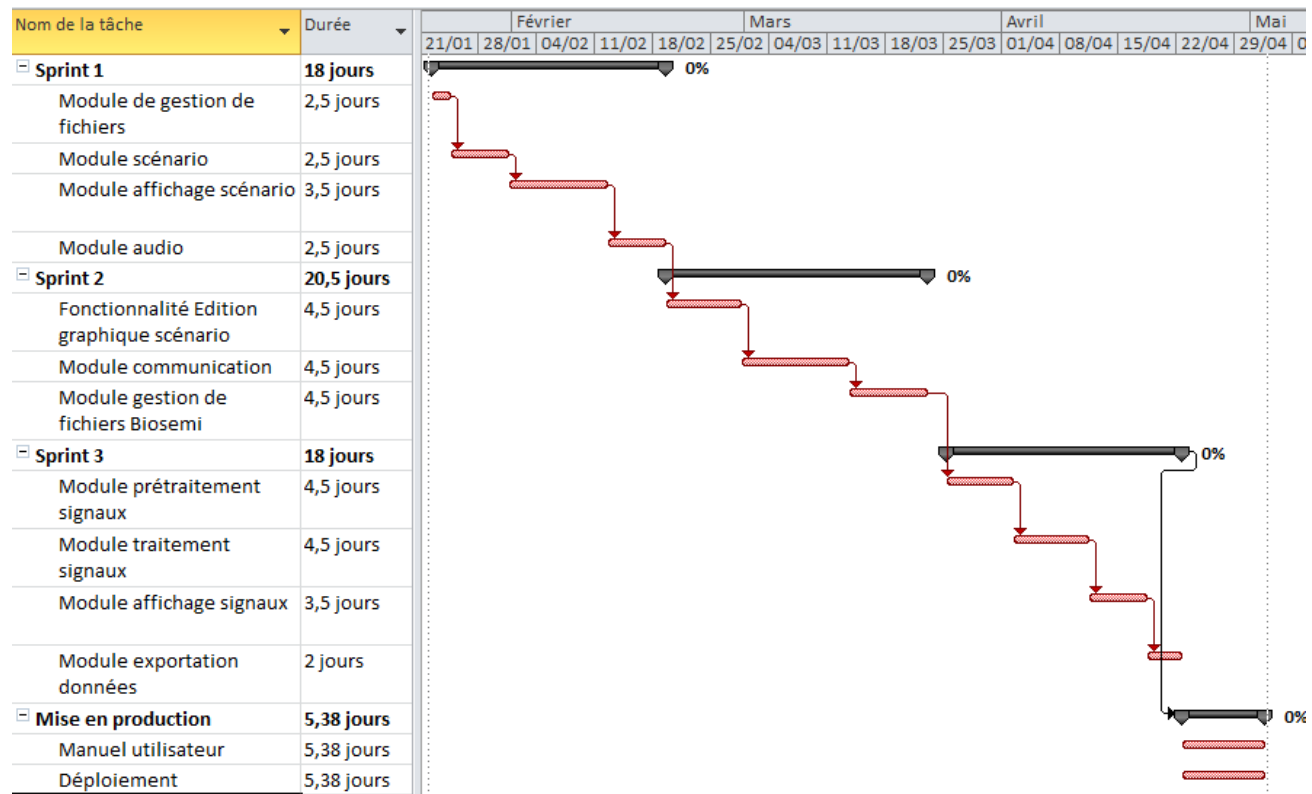


FIGURE 2.14 – Planning à partir du sprint 1

Références

- Framework Qt [<http://qt.digia.com>]
- Bibliothèque FMOD [<http://www.fmod.org>]
- Bibliothèque Qwt [<http://qwt.sourceforge.net>]
- Bibliothèque EDFLib [<http://www.teuniz.net/edflib>]
- Bibliothèque Blitzwave [<http://oschulz.github.com/blitzwave>]
- Bibliothèque GSL [<http://www.gnu.org/software/gsl>]
- Bibliothèque Wavelet1D [<http://code.google.com/p/wavelet1d>]

- Licence GPL [<http://www.gnu.org/licenses/gpl.html>]
- Licence LGPL [<http://www.gnu.org/copyleft/lesser.html>]
- Licence Qwt [<http://qwt.sourceforge.net/qwtlicense.html>]
- Licences FMOD [<http://www.fmod.org/fmod-sales.html>]

- H. Schwilden, Concepts of EEG processing : from power spectrum to bispectrum, fractals, entropies and all that, 2006.
- T. Malina, A. Folkers, U.G. Hofmann, Real-time EEG processing based on Wavelet Transformation, 2002.
- C. DŠAvanzo, V. Tarantino, P. Bisiacchi, G. Sparacino, A wavelet Methodology for EEG Time-frequency Analysis in a Time Discrimination Task, 2009.

- F. Holmes Atwater, Accessing Anomalous States of Consciousness with a Binaural Beat Technology, 1997.
- F. Holmes Atwater, Binaural Beats and Regulation of Arousal Levels, 2009.
- J. D. Lane, J. E. Owens, G. R. Marsh, Binaural Auditory Beats Affect Vigilance Performance and Mood, 1998.
- TL. Huang, C. Charyton, A Comprehensive Review of the Psychological Effects of Brainwave Entrainment, 2008.

Solution de tests pour l'étude des battements binauraux

Département Informatique
5^e année
2012 - 2013

Rapport de Projet de Fin d'Etudes

Résumé : Projet de Fin d'Etudes sur la réalisation d'un logiciel de test des effets neurophysiologiques de l'écoute de battements binauraux. Le programme permet d'effectuer la génération des stimulations auditives en temps réel suivant un scénario préalablement configuré, puis l'analyse de l'activité électrique du cerveau, mesuré par le matériel d'EEG Biosemi, grâce à la transformée en ondelettes.

Mots clefs : EEG, électroencéphalographie, transformée en ondelettes, traitement du signal, activité électrique, cerveau, biopotential, Biosemi, C++, Qt, Qwt

Abstract: Final Project Assignment about development of a software that tests neurophysiological effects of the hearing of binaural beats. The program generates real-time auditive stimulations set in a configurable scenario, and analyses the brain electrical activity, measured by Biosemi EEG hardware, using wavelet transform.

Keywords: EEG, electroencephalography, wavelet transform, signal processing, electrical activity, brain, biopotential, Biosemi, C++, Qt, Qwt

Encadrant
Pascal MAKRIS
pascal.makris@univ-tours.fr

Université François-Rabelais, Tours

Étudiant
François DENNIG
francois.dennig@etu.univ-tours.fr

DI5 2012 - 2013