Université de Fribourg, Suisse

VIRTUALDJ

Projet du cours "Multimodal Interfaces"

Auteurs:

Mathias Seuret	Pierre Vanhulst	Marc Chatton
mathias.seuret@unifr.ch	pierre.vanhulst@unifr.ch	marc.chatton@unifr.ch

Supervisé par:

Jacques Bapst, Omar Abou Khaled, Denis Lalanne, Elena Mugellini

Fribourg, Mai 2012

Table des matières

Graphs

1 Introduction

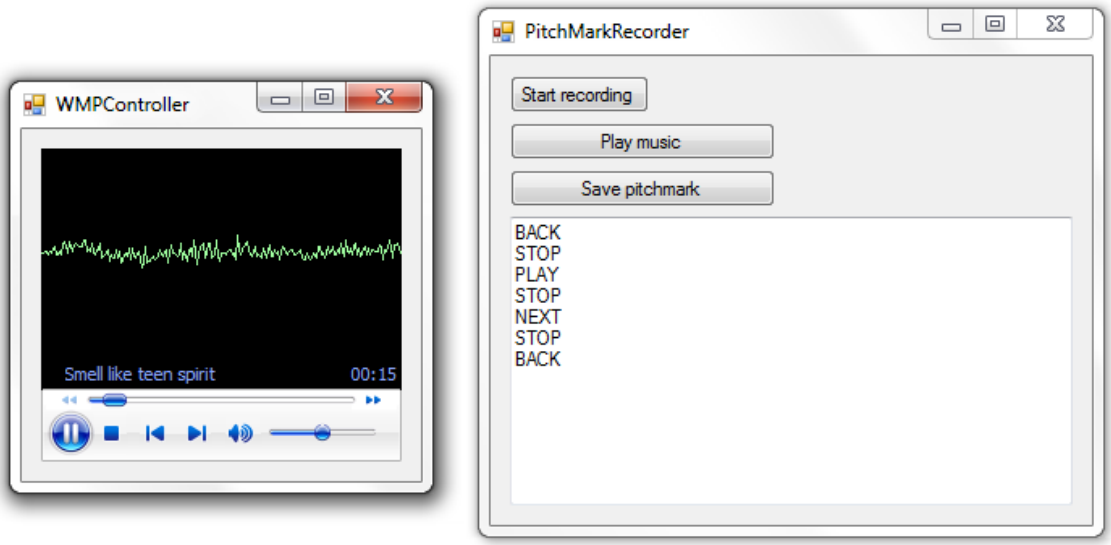
Bien que tout ordinateur personnel soit maintenant livré avec un clavier, une souris et un écran, d'autres périphériques d'entrée / sortie peuvent être utilisés pour communiquer avec. Qu'il s'agisse de webcams, de micros, d'imprimantes ou autre, ils peuvent dans certains cas remplacer avantageusement le triplet habituel.

Le cours "Multimodal interfaces", donné à l'Université de Fribourg, a pour sujet l'utilisation de plusieurs interfaces afin de réaliser une tâche. Le mélange des différentes interfaces, comme la reconnaissance de la parole ou l'utilisation d'objets divers, peut être très efficace tout en semblant naturel pour l'utilisateur.

Notre objectif pour le projet à effectuer dans le cadre de ce cours était de concevoir un logiciel permettant à une personne malvoyante de contrôler aussi aisément que possible les fonctions de base de Windows Media Player (WMP).

2 Description du projet

Le projet VirtualDJ est un lecteur audio aux fonctions basiques dont les commandes principales peuvent être activées par la voix et/ou par les gestes. Il a été conçu principalement dans le but d'aider les personnes malvoyantes mélomanes à pouvoir gérer leur lecteur depuis leur salon, sans passer par une personne voyante pour leur mettre leur chanson favorite, leur accordant ainsi un peu plus d'indépendance.



2-1 Présentation de l'interface de VirtualDJ

La particularité de ce lecteur est la reconnaissance de pitch: il s'agit d'un système permettant à l'utilisateur de mettre un "tag" à ses chansons qui permet de les retrouver en sifflant un des rythme les plus représentatif de celles-ci (typiquement le refrain de "Trololo" d'Eduar Khil). Toutes les commandes (vocales ou gestuelles) sont disponibles au chapitre 4 , triées en fonction des modalités qui les concernent.

3 Dispositifs utilisés et modalités

Partant du principe que notre projet avait pour objectif de faciliter la vie d'une personne malvoyante, la voix et les gestes étaient quasiment nos seules alternatives. Néanmoins, il y a également une interface utilisateur permettant d'accéder, via la souris, à toutes les fonctions qui ont été développées. Cette interface ne sera pas prise en compte dans ce travail.

3.1 *Dispositifs*

Voici les deux dispositifs utilisés pour la réalisation de ce projet.

3.1.1 Kinect

Pour la reconnaissance gestuelle, nous avons utilisé la Kinect, qui est le périphérique high-tech le plus vendu au monde, avec ses 10 millions d'appareils vendus.[1] Partant du principe que la Kinect était bien répandu sur le marché, notre programme aurait plus de chance de s'y faire une place.

3.1.2 Microphone

Un microphone externe était également utilisé, le système de reconnaissance vocale de la Kinect aurait pu suffire, mais dans un souci d'éloigner au maximum les périphériques d'entrée et de sortie de la voix et de rapprocher l'utilisateur du microphone, cette option a été choisie.

3.2 *Case / Care*

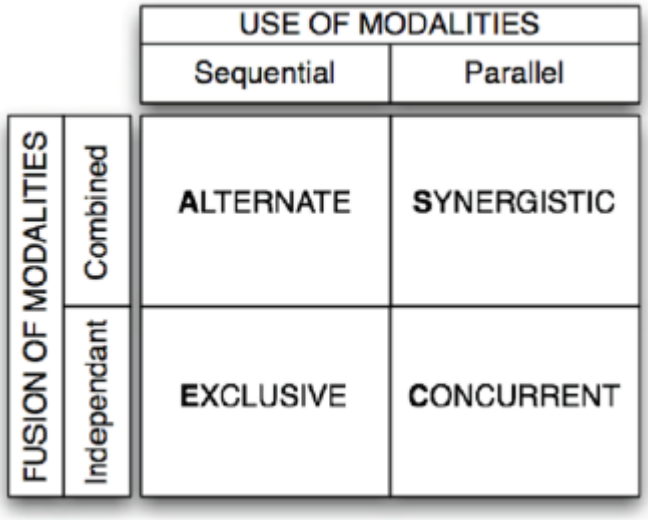
Dans un objectif de conceptualiser les différentes relations possibles entre les modalités d'entrée et de sortie, et de formaliser les interactions multimodales humains-machine

[1] http://fr.wikipedia.org/wiki/Kinect

[O. Abou Khaled, J. Bapst, D. Lalanne, E. Mugellini, 2012, MMI03-slide 20], nous allons utiliser le modèle CASE et CARE vu en cours.

3.2.1 Case:

Le modèle CASE (acronyme issu de Concurrent, Alternate, Synergistic, Exclusive) sera utilisé afin de définir les types de communications multimodales sous un aspect "machine". Ci-dessous, la figure représentant le modèle en question.



3-1 Le modèle CASE

[O. Abou Khaled, J. Bapst, D. Lalanne, E. Mugellini, 2012, MMI03- slide 23]

Notre projet possède deux des quatre caractéristiques précitées:

L'exclusivité: Quasiment toutes les tâches de notre projet peuvent être utilisées les unes après les autres, en utilisant une modalité à la fois.

La synergie: Il est possible d'activer une commande en combinant deux modalités, pour cela il faut mettre la main droite sur la tête et donner un ordre par oral.

3.2.2 Care

Le modèle CARE (acronyme issu de Complementarity, Assignment, Redundancy et Equivalence) sera utilisé afin d'analyser les propriétés de notre interface multimodale, ce d'un point de vue "humain". Sur les quatre propriétés proposées, notre projet en possède trois:

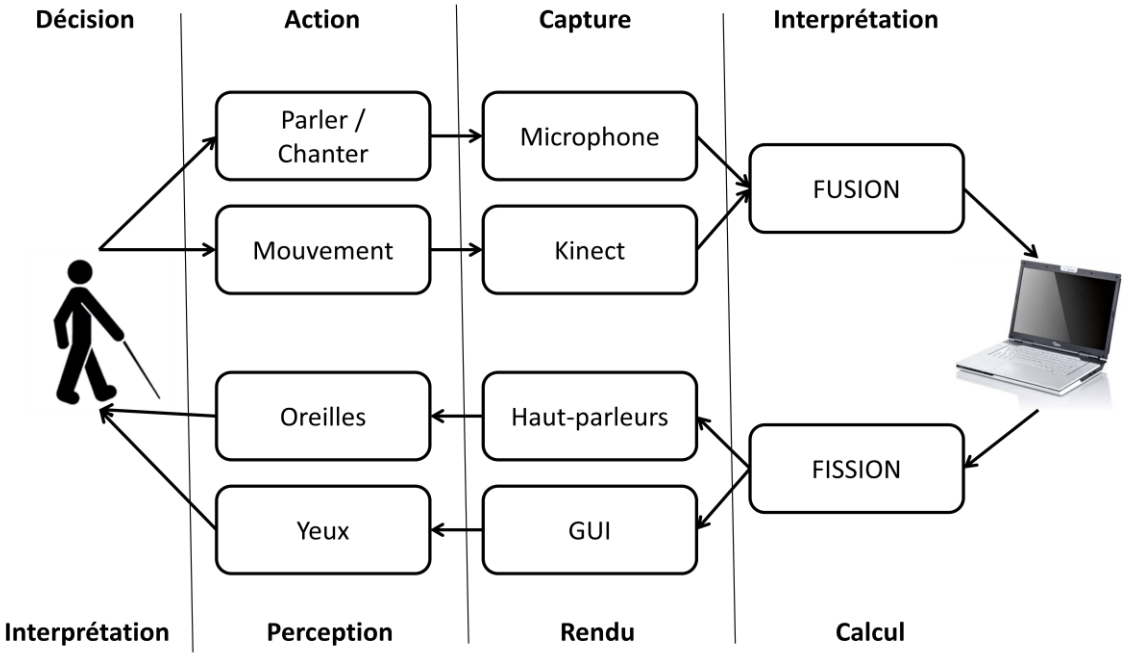
L'assignation (Assignment): Pour accéder à la fonctionnalité de reconnaissance de Pitch, il n'y a que l'option "vocale" ("Listen" et "Pitch"), de même que les fonctions "Sleep" et "Activation". Nous n'avons pas implémenter de gestes correspondants à ces commandes.

L'équivalence: Des gestes et des mots ont été implémentés afin de démarrer ou stopper une chanson, ainsi qu'avancer ou reculer dans la playlist. Les deux options donnent le même résultat.

La complémentarité (Complementarity): Cette propriété est utilisée lorsque l'utilisateur va joindre une parole à un mouvement afin de créer une commande. Le bras droite sur la tête amène à une pause, ce qui active les commandes vocales bloquées jusque-là.

3.2.3 Interactions

Le diagramme ci-dessous donne un aperçu des interactions multimodales entre l'humain et la machine, ainsi que de l'architecture de notre travail.



3-2 Interactions multimodales - Humains-Machine

Il est relativement ironique de constater qu'une des perception rendue à l'utilisateur est d'ordre visuel, notre projet étant destiné prioritairement aux personnes non-voyantes. Néanmoins, le développement de notre programme n'étant pas encore optimal, une interface utilisateur est indispensable afin de pouvoir accéder aux commandes qui ne

sont pas encore gérées par la voix ou les gestes (volume, barre de progression de la chanson, etc.).

4 Réalisation

4.1 *Langage et environnement de développement*

Utilisant la Kinect pour détecter les mouvements, nous n'avions pas d'autres choix que de nous tourner vers la technologie .NET pour le développement de notre projet, ainsi que le langage C# reflétant au mieux cette architecture. Nous avons choisi l'environnement de développement de "Visual Studio 2010", étant disponible gratuitement sur le site de MSDN de l'Université de Fribourg[2] et disposant de tous les outils nécessaires à la réalisation de ce projet.

4.2 *Reconnaissance vocale*

4.2.1 Concept

La reconnaissance vocale est un outil qui se démocratise depuis quelques années. Avec l'avènement des téléphones de nouvelles génération, ainsi que l'intégration de cette technologie par Windows dans ses derniers système d'exploitation, elle est amenée dans un avenir proche à s'inviter chez nous, pour régler son thermostat, préchauffer son four ou commander tous types d'appareils ménager par la voix.

Nous avons pu voir dans ce projet à quel point il était rapide de mettre en place un système de reconnaissance vocal plus ou moins fiable avec l'ajout d'une simple librairie. Les résultats sont cependant dépendants des mots choisis, et le taux d'erreurs augmentent proportionnellement au nombre de mots auquel on a attribué une tâche. Pour limiter ce risque, nous avons choisis ces mots en regard de certains critères : qu'ils possèdent un maximum de consonnes "dures", qu'ils possèdent un minimum de syllabes, qu'aucun ne possède une consonance identique et que leur prononciation ne soit pas trop péjorée par un accent francophone.

Les tests utilisateurs effectués ont aussi prouvé que les erreurs variaient grandement d'un locuteur à l'autre, l'élocution et la portée de la voix étant des facteurs déterminants.

[2] http://student.unifr.ch/software/fr/msdnaa.php

Afin d'optimiser la reconnaissance, nous avons également opté pour un micro externe, et non pas celui intégré à la Kinect. Nous avons ainsi pu le placer au plus près de l'utilisateur, et constaté des performances plus stables.

4.2.2 Guide

Nous commencerons par vous donner la liste des commandes disponibles via la reconnaissance vocale.

- "Play!": utilisée pour lancer la chanson.
- "Stop": utilisée pour stopper la chanson.
- "Next": utilisée pour avancer d'une chanson dans la liste de lecture.
- "Back": utilisée pour reculer d'une chanson dans la liste de lecture.
- "Listen": utilisée pour enregistrer un pitch.
- "Pitch": utilisée pour la reconnaissance d'une chanson via un sifflement.
- "Sleep": utilisée pour bloquer l'accès à toutes les commandes vocales.
- "Activation": utilisée pour débloquer l'accès à toutes les commandes vocales.

Toutes ces commandes, hormis le "STOP", nécessitent un silence complet du lecteur pour être utilisé (ce code sera décrit dans la partie 4.2.3).

A noter que les deux derniers appels (Sleep et Activation) ont été implémentés après le test utilisateurs. Nous avons remarqué que lorsque des explications étaient données aux testeurs, le système reconnaissait des mots non-désirés durant ces conversations. Ainsi, le mot "Sleep" bloque l'accès aux fonctions ci-dessus, et le système de reconnaissance de ces mots n'est réactivable que par le mot "Activation" (celui-ci n'a malheureusement pas une bonne moyenne de reconnaissance, au même titre que "Listen").

4.2.3 Implémentation

Voici les parties importantes du code:

On instancie le "speech engine" et on l'initialise sous la forme d'un contructeur

```
recognitionEngine = new SpeechRecognitionEngine();
```

Il faut également créer un nouvel "objet grammaire" qui permettra au "speech engine" de savoir quel mot écouter.

```
private Grammar CreateGrammarObject()
        {
            Choices commandChoices = new Choices("Sleep", "Activation", "Next",
"Play", "Back", "Stop", "Again", "Listen", "Pitch");
            GrammarBuilder grammarBuilder = new GrammarBuilder("GO");
            grammarBuilder.Append(commandChoices);
            Grammar g = new Grammar(grammarBuilder);
            return g;
        }
```

Les mots "capturés", nous pouvons maintenant placé un niveau minimum de confiance pour lequel le mot sera accepté:

```
if (word.Confidence > 0.8f)
```

Enfin, si le "speech engine" confirme ce mot, il faut lui dire quelle tâche effectuer:

```
case "Back":
                        if (Activation == true)
                        { instance.logBox.AppendText("Need Activation" + "\n"); }
                        else
                        {
                            if (WMPController.get().isPlaying() $$
WMPController.get().isPaused()  )
                            { instance.logBox.AppendText("Stop the player to
activate the commands" + "\n"); }
                            else
                            {
                                WMPController.get().previous();
                                instance.logBox.AppendText("BACK" + "\n");
                            }
                        }
                        break;
```

La fonction "isPlaying()" vérifiera si WMP n'est pas en mode "muet", si le volume n'est pas à zéro et si une chanson est en train de jouer sur le lecteur, la fonction et "isPaused()" contrôlera si la pause est enclenchée ou non.

```
 public bool isPlaying()

        {

            return !MyWMP.settings.mute && playing && MyWMP.settings.volume!=0;

        }
```

La reconnaissance vocale reste donc active en continu. Néanmoins l'action est lancée que lorsque la fonction renvoie un "false" confirmant ces critères.

4.3 *Reconnaissance de sifflotements*

4.3.1 Concept

Choisir une musique dans une liste de lecture est chose aisée lorsque deux conditions sont réunies :

- il faut voir ladite liste,
- et avoir arriver à se rappeler des noms des musiques.

Utiliser un système de synthèse vocale pour indiquer les musiques à disposition n'est pas une solution envisageable car cela prendrait beaucoup de temps pour les énumérer si la liste de lecture est longue, et les titres étant potentiellement dans différentes langues, la prononciation ne serait pas correcte. De plus, utiliser un système de reconnaissance vocale pour choisir la musique à jouer n'est pas l'idéal, pour des raisons de langue et d'accent.

Nous avons donc choisi une autre alternative : la création de favoris en sifflant. Lorsqu'une musique plaisant particulièrement à l'utilisateur est jouée, ce dernier peut l'associer à un sifflotement. Et lorsque l'utilisateur souhaite réécouter une de ses musiques favorites, il peut siffler l'air choisi.

4.3.2 Guide

L'utilisation de la reconnaissance de sifflotements est composée de deux parties : l'association d'une musique à une mélodie sifflée et la recherche de la musique associée à une mélodie sifflée.

Pour indiquer au programme que l'on souhaite enregistrer un favori pour la musique actuellement lue, il faut mettre la lecture en pause, dire "listen" puis siffler durant cinq secondes. Il est conseillé de siffler un passage de la musique ayant beaucoup de notes différentes.

Pour ce qui est de la recherche d'une musique correspondant à une mélodie sifflée, la procédure est très similaire : il faut mettre la musique en pause, dire "pitch" puis siffler cinq secondes. La musique ayant un favori le plus proche sera jouée.

Il est tout à fait possible de relier une musique à plusieurs mélodies sifflées.

4.3.3 Implémentation

Les sifflotements sont transformés en vecteurs de caractéristiques. Lors de la recherche d'un favori, les vecteurs sont comparés et celui s'approchant le plus est choisi. La création du vecteur de caractéristiques doit résister aux contraintes suivantes :

- deux sifflotements correspondant ne seront pas produits à la même vitesse
- ni dans la même gamme de fréquences
- et tout au long d'un sifflement, il y a une dérive dans les fréquences, une même note ne sera pas deux fois identique

Pour ce faire, nous procédons de la manière suivante :

- l'entrée audio est lue durant un certain temps (par défaut 5 secondes)
- une liste de floats A est crée
- le signal audio est découpé en parties de 0.23 seconde
- la transformée de Fourier de chaque partie multipliée par une fonction de Hamming est calculée
- La moyenne de l'intensité de la transformée est calculée, et si la valeur maximale est 35 fois plus grande que la moyenne, la fréquence correspondante est ajoutée à la liste A
- Ensuite, la différence de fréquences entre chaque élément consécutif de cette liste est mise à une échelle logarithmique et arrondie puis mise dans une liste B, les doublons étant retirés

Cette liste B est utilisé comme vecteur de caractéristiques. En pseudo-code, cela donne :

```
Sound input = Microphone.record(5s);
Sound[] chunk = input.cut(23ms);
List A,B;
foreach (Sound s : chunk) {
        F = fft(s);
        if (max(F)<35*avg(F)) continue;
        freq = get_freq(max(F));
        if (freq < 500 || freq > 3000) continue;
```

```
            A.add(freq);
    }
    prev = A.first();
    foreach (x : A) {
            y = (int)round(19 * log(x / prev) / log(2));
            if (y != B.last()) B.add(y);
            prev = x;
    }
```

La durée de cinq secondes pour les sifflements a été choisie parce qu'elle n'est ni trop longue (l'utilisateur ne passe pas beaucoup de temps à siffler), ni trop courte (l'utilisateur peut siffler de nombreuses notes). Le choix d'utiliser une division de deux fréquences consécutives dans une échelle logarithmique de base 2 vient du fait que l'oreille humaine détecte les sons aussi dans une échelle logarithmique. Les valeurs numériques, comme 23ms, n'ont pas été choisies arbitrairement. Nous avons enregistré une série de sifflotements et avons comparé le taux de reconnaissance pour des milliers de paramètres numériques choisis aléatoirement. Cette combinaison est la moyenne des quelques meilleurs. Ils dépendent toutefois probablement de la personne qui siffle et du matériel utilisé. Si ce prototype venait à être réellement utilisé, un système de calibration devrait être implémenté.

Pour la calibration que nous avons faite, six "favoris" et quatre sifflotements pour chacun de ces six ont été enregistrés. Cela fait un total de 36 sifflotements à tester. Certains jeux de paramètres avaient un taux de reconnaissance de 96%.

Utiliser un système de reconnaissance des sifflotements pose toutefois un problème d'ordre pratique et un autre d'ordre culturel. Lors des tests, puis en demandant autours de nous, nous avons appris que de nombreuses personnes ne savent pas siffler une mélodie simple. Ce système nécessiterait donc un apprentissage relativement long et ennuyeux pour certains. Un autre problème vient du fait que dans certaines cultures, il

est mal vu de siffler. En Ukraine et en Russie, par exemple, siffler à l'intérieur d'un bâtiment est censé porter malheur.

4.4 *Reconnaissance de mouvements*

4.4.1 Concept

La dernière partie de notre programme concerne la reconnaissance de mouvements. Celle-ci fut considérée comme nécessaire pour mener à bien nos objectifs : en effet, notre projet devait proposer une solution non-intrusive pour passer des ordres, il nous a donc semblé logique de ne pas activer la reconnaissance vocale à tout instant, mais plutôt d'employer un geste pour prévenir le système que des commandes vocales sont sur le point d'être saisies. Par la même occasion, nous avons souhaité offrir la possibilité aux utilisateurs de manœuvrer entièrement le programme grâce aux mouvements, puisqu'il est envisageable que celui-ci tourne dans un milieu sonore bruyant (auquel cas les commandes vocales peuvent devenir problématiques).

En termes technologiques, quoi de mieux que Kinect, à l'heure actuelle, pour la reconnaissance de mouvements ? Notre choix s'est tout naturellement porté sur l'appareil de Microsoft suite aux avis dithyrambiques que la plupart des développeurs émettaient à son égard.

4.4.2 Guide

La reconnaissance des mouvements de notre programme fonctionne comme suit :

Pour attirer l'attention du programme, l'utilisateur doit placer sa main droite au-dessus de sa tête. À cet instant, la playlist se met en pause et le programme attend des ordres (vocaux ou gestuels). Si l'utilisateur relâche sa main droite, la playlist reprend. Nous avons cependant émis l'hypothèse que ce geste trop anodin puisse être accompli involontairement par les utilisateurs : l'évaluation du programme avait notamment pour but d'infirmer ou de confirmer cette crainte.

Une fois la playlist en pause, l'utilisateur peut naviguer en son sein :

- Il peut atteindre la chanson suivante en tendant son bras gauche vers la droite.
- Il peut retourner en arrière en tendant son bras gauche vers la gauche
- Il peut arrêter la playlist en levant la main gauche. Le même geste est employé pour lancer la playlist si celle-ci est déjà arrêtée.

4.4.3 Implémentation

Pour implémenter la reconnaissance de mouvements, nous nous sommes basés sur le SDK Kinect proposé par Microsoft. D'abord commencé sur la version 1.0 du SDK, notre programme a migré peu de temps avant la fin du projet vers la version 1.5, qui offrait notamment le “near mode” (la reconnaissance à moins d'un mètre de Kinect). À ce sujet, nous avons eu la mauvaise surprise de constater que le “near mode” n'était à priori pas compatible avec notre Kinect première génération.

Le code lui-même est d'une grande simplicité. Partiellement basé sur les exemples Kinect proposé par Microsoft, il se décompose en trois parties :

- Initialisation de Kinect (senseur, squelette, squelette suivi par Kinect)
- Vérification de la position de la main droite
- Vérification de la position de la main gauche

La vérification de la position des mains s'effectue par de simples comparaisons de coordonnées. Il nous a effectivement semblé suffisant de se limiter aux coordonnées, plutôt que d'employer l'orientation des os.

Notre programme se met donc en pause si la main droite est au-dessus de la tête (axe Y). Il arrête ou lance une playlist si la main gauche est 15cm au-dessus de la tête (axe Y). Pour avancer d'une chanson, la main gauche doit être au-dessous de l'épaule droite (axe Y) mais au-delà de celle-ci (axe X). Pour revenir en arrière, il faut que le coude gauche et le poignet gauche soit à la même hauteur (axe Y).

5 Évaluation

Afin de s'assurer que les hypothèses sur lesquelles nous avons conçu notre programme étaient valides, une évaluation de notre prototype a été effectuée auprès d'une poignée de testeurs recrutés spécialement pour l'occasion.

Nous souhaitions ainsi découvrir des bugs, mais également des failles d'*usability*.[3]

[3] l'usability est une caractéristique d'un objet, qui comprend le degré de satisfaction de son utilisateur, ainsi que son efficacité et son efficience pour atteindre son objectif.

5.1 *Informations sur les tests*

5.1.1 Types d'utilisateurs et conditions de test

Six personnes ont participé à l'évaluation. Partant du principe que le public-cible de notre programme est particulièrement large, l'idéal aurait été de mettre à contribution des personnes au profil varié. Cependant, pour une première batterie de tests, il nous a semblé plus pertinent de nous adresser principalement à des individus âgés de 25 à 35 ans. Les domaines de connaissance des testeurs étaient néanmoins très différents, ce qui nous assurait de ne pas cloisonner notre évaluation à un groupe d'informaticiens "power users" potentiellement plus à l'aise que la moyenne avec les nouvelles technologies. Nous n'avons pas pu trouver de personnes malvoyantes pour tester l'interface (il aurait fallu que le test se déroule dans un environnement familier pour réduire leur temps d'adaptation), mais les utilisateurs ne pouvaient pas se fier à l'écran d'ordinateur pour comprendre le fonctionnement du test. Ils étaient par conséquent forcés de se fier à leur ouïe, comme l'aurait fait une personne malvoyante.

Cette première évaluation s'est déroulée dans une salle neutre et un ordinateur unique, afin de minimiser les risques d'incompatibilité matérielle ou d'autres mauvaises surprises.

5.1.2 Protocole

Les testeurs ont reçu chacun une courte introduction écrite à l'évaluation (vous trouverez le protocole des tests sur le CD-Rom mis en annexe), ainsi qu'un résumé des trois phases qui composent ce test :

- Phase 1 "play & stop" : des tests pour enclencher et stopper la playlist

- Phase 2 "next & back" : des tests pour naviguer dans la playlist

- Phase 3 "pitchmark" : des tests concernant la fonctionnalité-phare de notre programme

Deux membres de notre équipe étaient en permanence dans la salle pour observer et prendre des notes sur le nombre, le type et la fréquence des erreurs rencontrées par nos testeurs, l'objectif étant ici de définir le niveau d'efficacité et d'efficience de notre programme.

Concernant le niveau de satisfaction, un questionnaire fut distribué aux testeurs à la fin de la troisième phase. Il était surtout question de leur perception du fonctionnement du programme et de leurs préférences par rapport aux deux modalités présentes.

Vu l'état d'avancement du prototype, nous avons également jugé judicieux d'engager une discussion informelle avec les testeurs durant tout le déroulement de l'évaluation, afin d'obtenir autant de feedback "à chaud", malgré le risque d'influence que cela implique.

5.2 *Résultats*

Les résultats de l'évaluation nous permirent de dégager certains points intéressants concernant notre programme.

5.2.1 Prises de note

Voici les observations que nous avons réalisées durant l'évaluation.

Phase 1, commandes vocales : la commande "stop" répond à la perfection. Une seule erreur fut repérée sur l'ensemble des tentatives (soit les 12 demandées en plus des "stop" supplémentaires utilisés pour arrêter les enclenchements involontaires de la playlist) : elle était due au fait que le testeur n'a pas prononcé sa commande assez fermement. En revanche, la commande "play" est bien trop sensible : si elle répondait présente lorsque les testeurs l'appelaient, elle se permettait également d'intervenir à d'autres occasions (notamment lors des commandes vocales "okay" et "back"). Nous supposons que ce problème est dû à la reconnaissance vocale de Windows : il aurait peut-être été judicieux de choisir un autre terme. En outre, nous avons constaté à cette occasion que la fusion des trois parties du programme n'avait pas été faite correctement : la reconnaissance vocale n'exigeait pas de l'utilisateur qu'il mette la chanson en pause avec un geste de la main droite.

Phase 1, gestes : cette phase fut l'une des plus problématiques de notre test. Dans deux cas, la commande ne fut pas reconnue du premier coup. Les testeurs n'avaient pas pour réflexe de garder deux mains au-dessus de leur tête. Qui plus est, dans un cas, la commande "back" fut joué avant la commande "play" : le testeur avait levé le bras tendu, et lorsque son coude gauche et son poignet gauche furent à la même hauteur, la commande "back" fut reconnue avant la commande "play". Les utilisateurs étaient également surpris de constater que la chanson s'arrêtait automatiquement lorsqu'ils posaient la main droite sur la tête : difficile, en effet, de dire si la chanson était déjà en mode "stop" ou simplement en mode "pause".

Phase 2, vocale : la commande "next" fonctionne bien à partir du moment où le testeur pense à stopper la chanson avant de donner ses ordres. À l'inverse, la commande "back" n'a fonctionné normalement que pour deux utilisateurs : pour les autres, elle n'était soit simplement pas reconnue, soit interprétée comme "play". Nous sommes toujours dubitatifs à ce sujet.

Phase 2, gestes : La commande "next" fonctionne bien, alors que la commande "back" demande parfois à l'utilisateur d'agiter le bras afin que Kinect le considère comme étant réellement tendu. Là encore, la chanson ne reprenait pas automatiquement après qu'un ordre ait été passé, et il était nécessaire de passer l'ordre "play" pour obtenir le résultat souhaité.

Phase 3 : dès la première évaluation, nous avons constaté que la reconnaissance de pitch ne fonctionnait pas. La qualité du micro, le bruit ambiant et l'absence de calibrage pour un utilisateur spécifique ont rendu notre évaluation pratiquement impossible : les fichiers enregistrant les pitches associés à une chanson étaient généralement vides. Qui plus est, les deux commandes vocales associées à la reconnaissance de pitch étaient également problématiques : "Listen" demandait aux testeurs de prononcer le "t", alors que "okay" n'était reconnu que s'il était prononcé "oque" (sans quoi, le programme interprétait "play" à la place).

5.2.2 Questionnaire et discussions

Les résultats du questionnaire ne sont pas nécessairement objectifs, mais ils représente le sentiment de satisfaction des utilisateurs par rapport à leur expérience avec VirtualDJ. Ils peuvent donc donner de précieuses informations sur les pistes à explorer pour améliorer le plaisir d'utilisation d'un tel programme.

Multimodal Interfaces / Université de Fribourg

Questionnaire d'évaluation du projet "**VirtualDJ**".

Prénom : _____

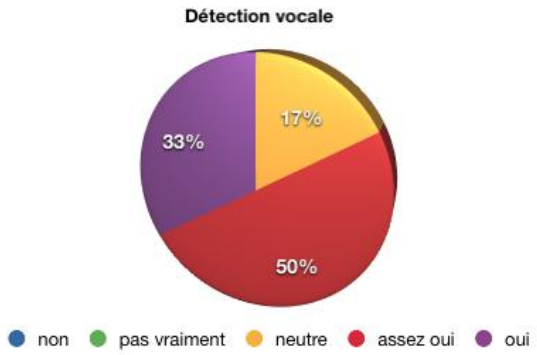
Langue maternelle : _____

Signature : _____

QUESTIONS	NON	Pas vraiment	Neutre	Assez oui	OUI
La détection vocale des chansons à fredonner est-elle au point?					
La détection vocale des ordres est-elle au point?					
La détection gestuelle des ordres est-elle au point?					
Quelles commandes supplémentaires auriez-vous imaginé sur ce projet?					
Quels sont les points sur lesquels une amélioration est possible / nécessaire?					
Le programme vous semble-t-il adapté aux personnes non-voyantes?					
Le conseilleriez-vous à une personne non-voyante?					
L'aspect général du programme vous convient-il?					
Quelle est la modalité que vous avez le plus appréciée?					

5-1 Questionnaire-type proposé aux testeurs

Détection vocale



Globalement, on constate que les utilisateurs sont satisfaits de la reconnaissance vocale. Les problèmes de reconnaissance de la commande "back" et la sensibilité accrue de la commande "play" n'ont manifestement pas entamé le plaisir d'utilisation de nos testeurs.

Détection des gestes

Un constat plus contrasté pour la détection des gestes. Certains testeurs ne rencontrèrent que peu de problèmes avec la reconnaissance de gestes. Nous imputons cette différence aux discussions que nous avons eu avec eu lors de l'évaluation : en effet, après avoir repéré le bug de la pause exposé ci-dessus, nous leur avons systématiquement expliqué le problème. Par conséquent, certains ont probablement interprété ce problème comme un bug dérangeant qui a justifié une note plus basse, alors que d'autres ont choisi d'ignorer le bug puisque nous l'avions déjà repéré.

Commandes supplémentaires

Réponses reçues : 2 x gestion du volume, amélioration du sifflement, appeler une liste de lecture, saut sans devoir stopper la chanson

Cette question avait pour but de nous donner ou de confirmer des idées que nous avions déjà concernant l'évolution possible de notre programme au-delà du cours d'Interfaces Multimodales. Comme on peut le voir, la gestion du volume semble être un des critères les plus importants pour les testeurs.

Point à améliorer

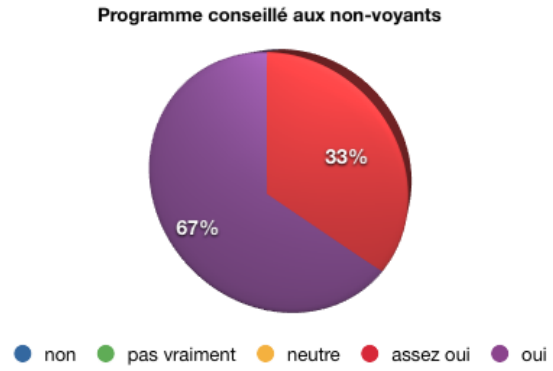
Réponses reçues : mots pour reconnaissance vocale pas pertinents, gestes plus simples, bloquer la reconnaissance vocale lors de conversations, meilleure détection vocale, meilleure reconnaissance gestuelle, pitchmark qui fonctionne

Cette question a donné quelques résultats douloureux : comme on peut le constater, sur l'intégralité de l'évaluation, toutes fonctionnalités de notre programme ont été citées comme problématiques au moins une fois. On peut donc en déduire que VirtualDJ manque globalement de robustesse, comme le laisse également entendre les notes prises durant l'observation de l'évaluation.

Programme adapté aux non-voyants

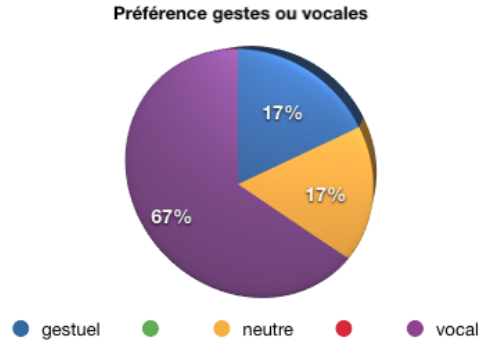
La majorité des testeurs considèrent que le concept de VirtualDJ est adapté pour une personne non-voyante. Nos discussions informelles ont montré que les testeurs étaient même assez enthousiastes à cette idée.

Programme conseillé personnes non-voyantes



Le résultat de cette question est très proche du précédent, étant donné la ressemblance entre les deux intitulés.

Préférence verbal - gestuel



Une grande majorité des testeurs ont été plus séduits par la gestion verbale du programme que par les gestes. Après discussion avec eux, nous pensons que ce résultat est principalement dû au fait que les gestes étaient peu naturels et que le bug de la pause rendait l'expérience particulièrement fastidieuse.

6 Conclusions

Ce travail nous a permis de mettre en application les concepts théoriques des interfaces multimodales vus en cours. Il était d'ailleurs très intéressant de concevoir un projet qui impliquait l'utilisation d'un langage qui ne nous était que très peu familier (C#), et découvrir l'étendu des possibilités (nous les découvrions au fur et à mesure de notre développement) qu'amenaient ces différentes modalités. Aucun d'entre nous n'avait encore aborder le sujet des reconnaissances vocales ou gestuelles dans le cadre de la programmation, et ce fût réellement une bonne surprise.

S'il fallait mettre un bémol à cette enthousiasme, nous pourrions quand même évoquer le fait que nous avons commis l'erreur de faire le test utilisateur peu de temps avant la reddition de ce travail. Cela ne s'est pas ressenti dans le projet final, puisque le programme est plus ou moins celui que nous avions imaginé au départ, mais nous sommes certainement passés à côté du développement de quelques commandes que nous ont conseillés nos testeurs, et qui auraient pu être très intéressantes pour ce travail.

6.1 *Améliorations possibles*

Comme évoqué par nos testeurs, la partie gestuelle pourrait être plus intuitive, avec peut-être des positions plus naturelles. Nous pourrions également imaginé le développement des autres options que propose WMP: un réglage du son effectué par geste, une gestion des playlists ou une sélection d'un artiste par la voix et, pourquoi pas, coupler cette application à un commerce en ligne, ce qui permettrait d'acheter les fichiers audio des chansons qui nous plaisent. Pour résumer, les idées ne manquent pas, contrairement au temps...

7 Annexe

7.1 *Contenu du CD-Rom*

Sur le CD, vous trouverez:

- Le power-point de notre présentation
- Le rapport en format PDF et DOC
- La vidéo de présentation
- Les codes du projet
- Les protocoles de tests