Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Modal Mario

« Jouez à Super Mario avec le geste et la voix »

Cours : Interfaces multimodales
Semestre : 6ème
Professeurs : Omar A. Khaled, Jacques Bapst, Denis Lalanne, Elena Mugellini
Etudiants : Garry Heger, Adrien Nicolet, David Zanella
Modalités : Geste & Voix

Table des matières

Introduction

Dans le cadre du cours Interfaces multimodales, un projet est à réaliser. Par groupe de trois étudiants, une application multimodale devait être imaginée, conçue et développée. Une seule consigne : utiliser au minimum deux modalités.

Le sujet

Il existe plusieurs émulateurs de vieux jeux vidéo, reproduisant par exemple des jeux de GameBoy ou Nintendo. Il est possible de configurer ces émulateurs pour utiliser des manettes ou le clavier. L'idée de notre application est de remplacer la manette de Nintendo ou les boutons du GameBoy par des opérations commandées par la voix et les gestes.

Les actions des quatre flèches seront reproduite grâce à des gestes (via la WiiMote). Ainsi, un mouvement vers la gauche actionnera virtuellement la flèche gauche, alors qu'un mouvement vers la droite actionnera la flèche droite.

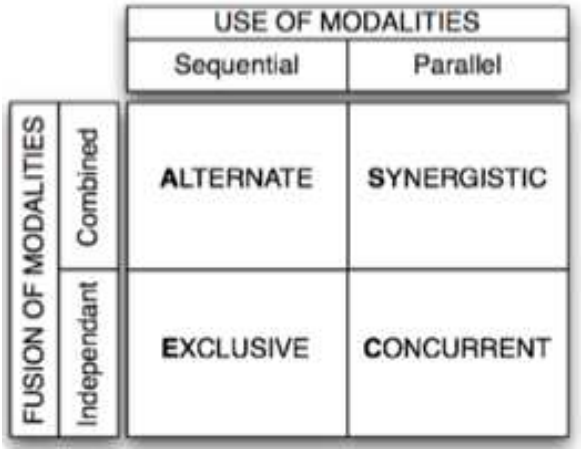
Les boutons « A », « B », « start » et « select » seront actionnés par la voix.
Nous avons choisi d'utiliser un des jeux les plus célèbres, Super Mario Land sur GameBoy.

Modalités d'interactions

Comme défini dans l'introduction, nos deux modalités sont

- La voix
- Le geste

Selon le modèle CASE, plusieurs types de fusions des modalités sont définissables.



Dans ce projet, nous travaillons dans un cas **concurrent**. En effet, les deux modalités utilisées interviennent en parallèle : faire avancer Mario et lui demander de sauter en même temps. Par contre, les modalités sont utilisées de façon indépendante. En effet, bien qu'il faille parfois sauter et avancer en même temps pour franchir des obstacles, il n'existe pas de contrainte de temps directes entre les actions liées à la voix et au geste. Les deux actions peuvent très bien être effectuées entièrement l'une sans l'autre.

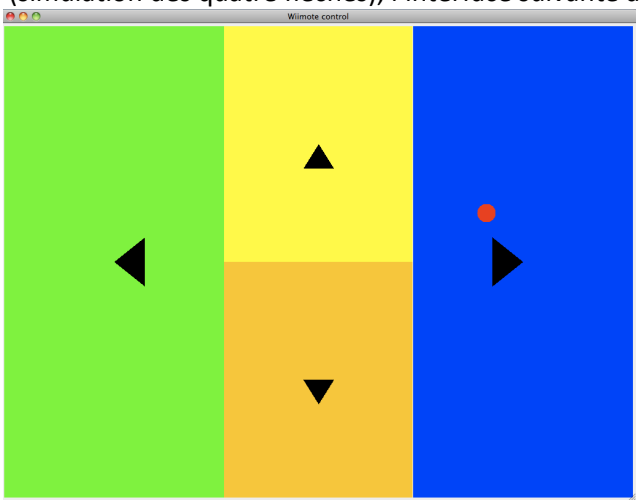
Choix de conception

Dès le début, nous avons choisis de travailler en JAVA, afin de pouvoir exécuter notre application sur une machine Linux, Mac ou Windows ; chacun des membres du groupe travaillant sur une plateforme différente.

Au début du projet, il a également fallu définir comment les interactions allaient être réalisées, avec nos deux modalités. Les chapitres suivants expliquent comment les modalités ont été implémentées.

Représentations des flèches

Pour faire bouger Mario (simulation des quatre flèches), l'interface suivante a été produite :



Le point rouge représente l'endroit où l'utilisateur se trouve, par pointage. Pour le faire bouger, il suffit de bouger la Wiimote naturellement vers la gauche, la droite, le haut ou le bas.

Dans ce cas, pour activer la flèche droite, il ne reste plus qu'à presser la touche « B » de la Wiimote. En la relâchant, la flèche est virtuellement relâchée, même si le point rouge se trouve encore en dessus de la zone bleue. Avec cette technique, nous simulons les doigts qui se lève simplement de la flèche, mais qui restent à proximité.

Représentation des boutons

Dans Super Mario Land, le bouton « A » permet de faire sauter Mario. Ainsi, nous avons décidé d'utiliser le mot « Jump » pour actionner le bouton « A ». Le bouton « B » permet par contre de tirer. Le mot « Fire » lui a donc été appliqué. Pour les boutons « start » et « select », le mot correspondant a également été utilisé.

Les puristes reconnaitront ici rapidement un problème. Lorsque l'on presse longtemps sur la touche « A », Mario va sauter plus haut…

Nous avons ainsi dû adapter notre interface pour prendre en compte cette spécificité. Avec le mot « jump », Mario va sauter à une hauteur « moyenne » ; comme si l'on avait pressé sur le bouton durant 800ms. Un autre mot a également été ajouté : « long jump ». Il va aussi actionner la touche « A », mais avec une durée de saut plus grande : 1000ms.

Interaction avec l'émulateur

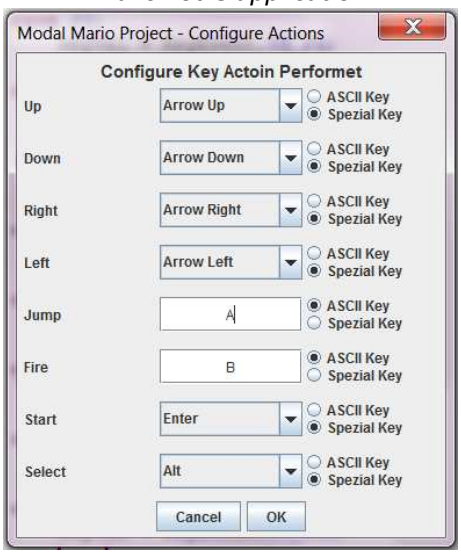
Pour jouer à Mario sur un PC, un émulateur de GameBoy est nécessaire. Plusieurs d'entre eux sont disponibles sur le Web.
Nous avons choisi de travailler avec l'émulateur « gambatte » :
http://sourceforge.net/projects/gambatte/
Il est gratuit et disponible pour les plateformes Windows, Mac, Linux.

L'émulateur est commandé par le clavier. Ceci signifie que lorsque la fenêtre de l'émulateur a le focus, il réagit sur certaines touches du clavier. Ces touches peuvent être configurées dans les options de l'émulateur.

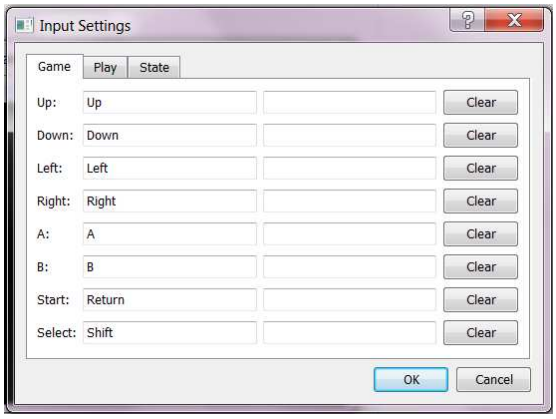
Pour interagir avec L'émulateur, notre programme simule donc l'utilisation du clavier. Il tourne en arrière-plan et réagit sur les inputs de la voix/Wiimote. De cette manière il suffit de mettre le focus sur l'émulateur et le commander.

Les associations de touches/action par défaut est différentes entre les plateformes où l'émulateur est installée. En plus, l'utilisateur peux changer c'est touches. Pour cela, dans notre programme, il existe la même possibilité. C'est à l'utilisateur de contrôlée la cohérence entre les programmes.

Dans notre application :



Dans l'émulateur :



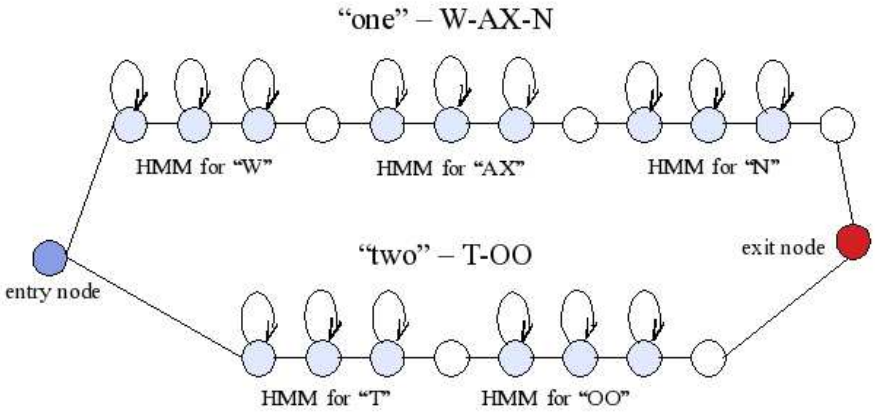
La partie de gauche montre quelle touche est pressée selon le mot prononcé ou le geste détecté. La partie de droite représente quelles sont les touches qui déclenchent les actions dans le jeu. Bien entendu les deux configurations doivent êtes cohérentes, comme dans l'exemple ci-dessus.

Librairies utilisées

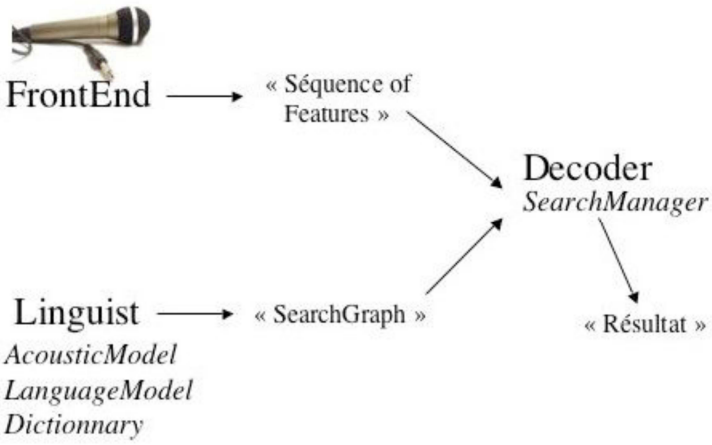
Avec Java, on trouve une quantité de librairies tant pour la gestion de la WiiMote que pour la voix. Nous n'avons pas eu le temps nécessaire pour en tester beaucoup et peut-être trouver la meilleures qui soit pour notre projet.
Après-coup et en discutant avec d'autres groupes, nous avons l'impression que les librairies offertes en .NET sont plus facilement maniables et fonctionnelles. Cela étant nous avons abouti à un résultat. Nous allons maintenant expliquer nos utilisations de librairies.

Sphinx 4

La librairie la plus populaire pour la gestion de la voix en Java est certainement Sphinx. Il s'agit d'un projet open source de reconnaissance vocale écrit en Java. Nous utilisons les librairies javax.speech pour la gestion de la reconnaissance vocale. Le fonctionnement de Sphinx est basé sur un modèle statistique HMM (Hidden Markov Models). En bref un modèle acoustique recueil des phonèmes qui sont comparés avec les signaux d'entrées afin de trouver le mot ou la phrase prononcée.
En gros, il s'agit de se promener dans un graphe afin de ressortir un mot :



Ici, W, AX, N, T et OO sont des phonèmes.
On peut choisir avec quel dictionnaire Sphinx va travailler. On en trouve des très basiques (reconnaissance des nombre de 0 à 10) ou des très complexes (vocabulaire de 60'000 mots). On trouve cela dans plusieurs langues (en tout cas l'anglais et le français).
Voici l'architecture générale de Sphinx 4 :



- Le Front-end découpe la voix en phonèmes pour la passer au décodeur
- Le décodeur est le cœur de Sphinx, il traite les infos du front-end et les compare avec la base de connaissance (linguist) pour fournir le résultat.
- La base de connaissance est « la base de donnée des mots » utilisée par le décodeur. Elle est composée de :
    - Dictionnaire
    - Un modèle acoustique
    - Un modèle de langage

Implémentation dans notre projet

Afin d'implémenter la reconnaissance vocale dans notre projet, nous nous sommes basés sur un exemple simple trouvé sur le site même de Sphinx. Nous l'avons ensuite adapté pour notre application. La structure (grossièrement) est la suivante :

- Librairies jsapi et sphinx4
- Un modèle acoustique : *WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz*
- Un fichier de configuration : commands.config.xml
- Un fichier de grammaire : commands.gram
- Des classes d'implémentation des reconnaissances : Voice.java (et VoiceListener.java pour les évènements)

Pour notre application, le vocabulaire utilisé était très restreint. Seule 5 commandes étaient disponibles :

- Jump
- Long jump
- Fire
- Start
- Select

Pour que Sphinx les reconnaisse, il faut les ajouter dans le fichier commands.gram.

```
#JSGF V1.0;

/**
 * JSGF Grammar for Modal Mario
 */

grammar comm;

public <greet> = (start | select | fire | jump | long jump);
```

Ceci est bien entendu un cas très simple mais avec cette grammaire JSGF, il est possible d'implémenter des reconnaissances bien plus complexes (comme des phrases par exemple).
Ensuite vient le fichier de configuration (commands.config.xml) dans lequel on spécifie par exemple le nom du fichier de grammaire ou le modèle acoustique à utiliser. Il y a encore beaucoup d'autre paramètre plus complexe que nous avons laissé comme dans l'exemple.

Petit extrait :

```
...
<!-- ******************************************************** -->
<!-- The Grammar  configuration                               -->
<!-- ******************************************************** -->

    <component name="jsgfGrammar" type="edu.cmu.sphinx.jsapi.JSGFGrammar">
        <property name="dictionary" value="dictionary"/>
        <property name="grammarLocation" value="resource:/voice/"/>
        <property name="marName" value="commands"/>
    </component>
...
```

Dans le code (petit exemple qui affiche simplement dans la console le mot prononcé) :

```
// gestionnaire de configuration
ConfigurationManager cm = new
ConfigurationManager(Voice.class.getResource("commands.config.xml"));

Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
recognizer.allocate();

// start the microphone or exit if the programm if this is not possible
Microphone microphone = (Microphone) cm.lookup("microphone");

if (!microphone.startRecording()) {
	System.out.println("Cannot start microphone.");
	recognizer.deallocate();
	System.exit(1);
}

while (true) {
	Result result = recognizer.recognize();
	String resultText = result.getBestFinalResultNoFiller();
	System.out.print("<"+resultText+">");
}
```

Notre code de reconnaissance est semblable à celui-ci. Nous y avons ajouté le déclenchement d'un événement spécifique (par exemple JumpEvent) afin de pouvoir travailler avec le principe des « listener ».

Réactivité de la reconnaissance

Ce que nous n'avons pas pensé au moment de choisir la librairie ou même lors du choix de notre projet, c'est la réactivité de la reconnaissance. En effet dans le jeu Mario, lorsque l'on presse A ou B (saut ou tir), l'action est instantanée. Dans notre application, nous avons remarqué que la reconnaissance nécessite un temps non-négligeable. D'après nos estimations, de 200ms à 1000ms doivent être nécessaire jusqu'au déclenchement de l'événement à partir du moment où le mot est prononcé, selon la qualité de l'environnement sonore ambiant.

Cette faiblesse influe sur la jouabilité du jeu et il est nécessaire d'anticiper les actions pour les déclencher un bon moment. Toutefois dans les meilleurs conditions, c'est-à-dire un endroit calme et un microphone près de la bouche, le jeu est entièrement jouable.
Nous avons tout de même tenté de remédier à cette faiblesse en essayant plusieurs pistes :

1. Notre accent anglais n'est peut-être pas très bon et cela gène Sphinx. Nous avons donc implémenté la reconnaissance en français. Résultat : malheureusement pas d'amélioration notable. Même dans certain cas les mots étaient moins bien reconnus (sauter-tirer est plus semblable que jump-fire).
2. Nous nous sommes dit que le fait de chercher 5 mots dans un dico de 60'000 mots est peut-être source de calculs inutils. Nous avons donc essayé avec le dictionnaire des chiffres. Par exemple déclencher le saut en ponçant « one ». Résultat : pas d'améliorations non plus.
3. Nous avons essayé de mettre le thread de la reconnaissance vocale avec une priorité plus haute par rapport aux autres. Résultat : pas d'amélioration.
4. Nous avons cherché comment « tuner » la reconnaissance avec les paramètres dans le fichier XML. Résultat : rien trouvé de concluant.

Finalement nous nous sommes résolus à utiliser notre idée de base avec le dictionnaire anglais.
Autres pistes auxquelles nous avons pensé mais qui, faute de temps, n'ont pas pu être explorées :

- Création d'un dictionnaire personnel avec seulement nos mots.
- Utilisation d'une autre librairie (Speaker Dependant par exemple).

AWT.Robot

Pour générer des évènements natifs, au niveau du système d'exploitation, il existe la classe `java.awt.Robot`. Elle nous donne, avec les méthodes `keyPress()` et `keyRelease()`, la possibilité de simuler un clavier.
Ces méthodes prennent en paramètre le code en int de la touche actionnée. Le code pour toutes les touches est configuré dans la classe KeyEvent (KeyEvent.ENTER pour la touche de retour).

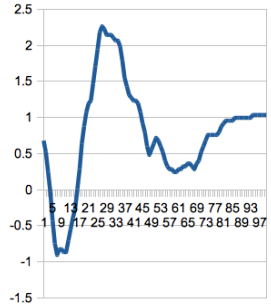
MoteJ

Après plusieurs tentatives avec d'autres librairies, nous avons décidé de choisir MoteJ. En effet, celle-ci permet d'utiliser facilement le pointeur infrarouge de la Wiimote.
Site officiel : http://motej.sourceforge.net/

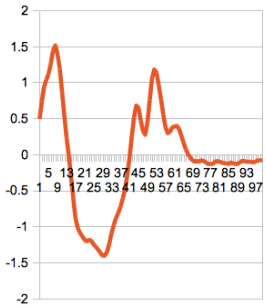
Avant d'utiliser à cette librairie, nous avons du en essayer plusieurs autres. Nous étions tout d'abord partie sur la librairie Wiigee qui permettait d'enregistrer des gestes puis de les reconnaître. Son désavantage : impossible d'enregistrer la phase de calibration des gestes. Il fallait donc recommencer l'enregistrement des quatres gestes à chaque démarrage de l'application. Cette faiblesse est annoncée officiellement sur le site, mais n'est pas présentée de façon très voyante.
Site officiel : http://www.wiigee.org/

Suite aux problèmes de Wiigee, nous sommes ensuite partie sur WiiRemoteJ. Cette librairies permettait d'acquérir les données bruttes des accéléromètres.
Site officiel : http://www.world-of-cha0s.hostrocket.com/WiiRemoteJ/

Nous avons essayé d'exploiter ces données pour détecter un mouvement vers la droite, gauche,… En effet, les courbes avaient toujours un aspect similaire :



Mouvement haut-bas sur l'axe des Z



Mouvement gauche-droite sur l'axe des X

Cependant, dans le temps à disposition, il nous aurait été impossible de produire une détection précise des mouvements.
C'est ainsi que nous avons décidé d'utiliser le pointage infrarouge de la Wiimote, plutôt qu'une reconnaissance de gestes liée aux accéléromètres.

Points à améliorer

Notre projet fonctionne, aveces fonctionnalités de base. Mais à cause du temps à disposition, notre programme a quelques faiblesses. Les points à améliorer et optimiser sont :

- Lenteur dans la détection des mots
- Perte de contrôle si l'utilisateur pointe en dehors des zones de couleur
- L'émulateur doit avoir le focus
- Problème de fiabilité dans la reconnaissance des mots avec Sphinx
- Ergonomie de l'interface utilisateur

Conclusion

En guise de conclusion nous constatons à quel point il n'est pas évident d'ajouter de nouvelles modalités. Dans ce cas très concret par exemple, les choix de Nintendo à l'instant où ils ont sorti le GameBoy ne sont pas anodins. D'un point de vue technologique premièrement, il n'était pas possible à l'époque d'utiliser d'autres modalités que celles fournies par la manette. Il s'agissait de la seule interface fiable pour jouer à un jeu de ce type. Aujourd'hui, cette approche a largement fait ses preuves.

Nous avons tenté de revisiter l'interaction permettant de commander Mario, en utilisant des modalités *nouvelles*. Nous constatons que la jouabilité n'en n'a pas été améliorée. Notre jeu n'est pas aussi intuitif et jouable qu'avec une manette !

Il faut aussi remarquer que ce jeu a été conçu pour une manette et que la conception d'un nouveau jeu en prenant compte dès le départ ces nouvelle modalités en modifierait le développement et l'approche de la conception. Pour ne citer qu'un exemple, le jeu Mario a été porté sur la console Wii et se joue avec la WiiMote d'une façon très agréable (mais la voix n'est une modalité dans ce jeu).

Dans un projet suivant, notre approche pour ces deux modalités serait bien différente grâce à cette 1ère expérience avec la reconnaissance vocale et la gestion du geste.
En effet, nous pourrions utiliser notre programme comme base de développement. En le rendant un peu plus générique, il serait facile d'ajouter des mots à détecter ou des gestes à prendre en compte.

Sources

- Sphinx 4 – project webpage : http://cmusphinx.sourceforge.net/sphinx4/
- Sphinx 4 : http://sandy.blanchet.free.fr/fr/public/Documents/RapportFinal.pdf
- Logiciel de reconnaissance vocale Sphinx-4 : http://diuf.unifr.ch/courses/05-06/mmi/projects/eif-sphinx/Rapport.pdf