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

Interfaces Multimodales

Mini-Projet : Mind Strooper

Auteurs :
Khaim Terrettaz
Lauriane Savary
Yves Peissard

Superviseurs :
O. Abou Khaled
J. Bapst
D. Lalanne
E. Mugellini

11 avril 2011

Abstract

L'effet "Stroop" est un phénomène psychologique qui nous montre la différence des tâches évidentes et tâches cognitif. En 1935, Ridley Stroop a fait l'expérience suivante : Les sujets devaient dire la couleur d'un mot sans lire le mot lui-même. Dans le cadre de ce projet il s'agit de développer une application qui implémente et qui étant l'expérience de Ridley Stroop en utilisant le périphérique Wii Remote ou tout court Wiimote et un microphone.

Ce travail a été effectué par des étudiant en sixième semestre de l'école d'ingénieurs et d'architectes de Fribourg.

Table des matières

1 Introduction

1.1 Contexte

Une interface multimodale veut dire que l'utilisateur d'une application quelconque peut interagir avec plusieurs moyens de communications tel que la parole et la geste. Un des avantages de ce moyen d'interaction est que des personnes handicapé, qui ne peuvent pas profiter des périphériques habituels comme clavier et souris, peuvent quand même profiter de diverses applications informatiques. Une autre motivation pour cette technologie est que les nouveaux moyens de pilotage d'application nous permettent de créer une nouvelle génération de software qui ouvre de nombreuses nouvelles possibilités. Dans le cadre du cours multimodale de l'école d'ingénieurs et d'architectes de Fribourg nous devons développer une application multimodale sous forme d'un jeu.

1.2 But du jeu

L'effet Stroop désigne l'interférence observée entre une tâche principale et un processus cognitif. Il a été découvert en 1935 lors de l'expérience suivante : des personnes devaient identifier la couleur d'un mot (tâche principale) sans lire le mot lui-même. Ainsi, le temps nécessaire à l'identification de la couleur avec laquelle le mot est écrit est beaucoup plus long lorsque le mot est incongruent (le mot "bleu" écrit en "rouge") que lorsque le mot est congruent (le mot "rouge" écrit en "rouge") ou neutre (le mot "lion" écrit en "rouge").

Notre jeu reproduit cette expérience. L'interface est composée de plusieurs zones de couleurs différentes (une par couleur disponible) et d'un mot (une couleur) écrit dans une couleur identique ou différente (voir figures 1 et 2). Le jeu se joue à l'aide d'une manette Wii et de la voix. Le fonctionnement du jeu est le suivant :

- Lorsque le joueur démarre le jeu, les zones de couleurs ainsi qu'un premier mot s'affichent. Le compte à rebours de 2 minutes démarre.
- Pour obtenir le point, le joueur doit pointer vers la zone de même couleur que la couleur écrite (le texte) et dire (voix) dans quelle couleur est écrit le texte.
- On reste sur le même mot jusqu'à ce que le joueur donne la bonne réponse ou que le temps soit écoulé.

Le but de ce jeu est d'obtenir un maximum de point dans le temps imparti.

Fonctionnalités supplémentaires (optionnelles) : En fonction du temps que nous aurons à disposition, nous pouvons imaginer les améliorations suivantes :

- Déplacement des zones de couleur
- Pénalité en cas d'erreur
- Inversion des couleurs (dire le mot écrit et pointer la couleur du mot)
- Implémentation d'une liste des meilleurs scores

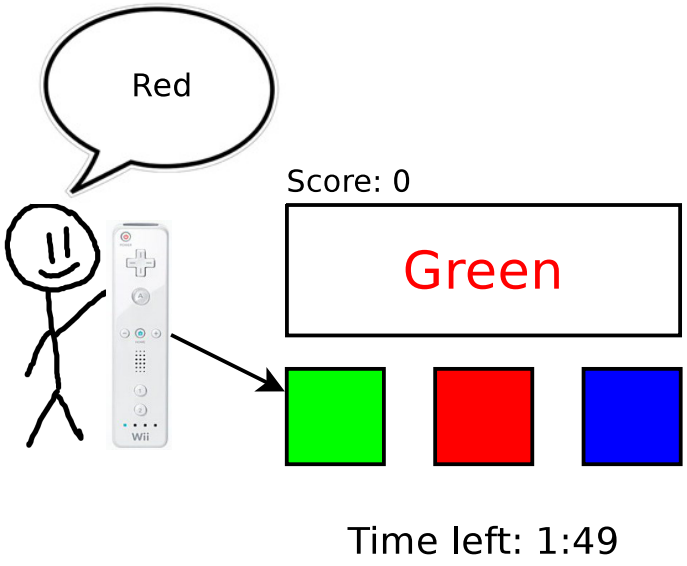

FIGURE 1: Début du jeu

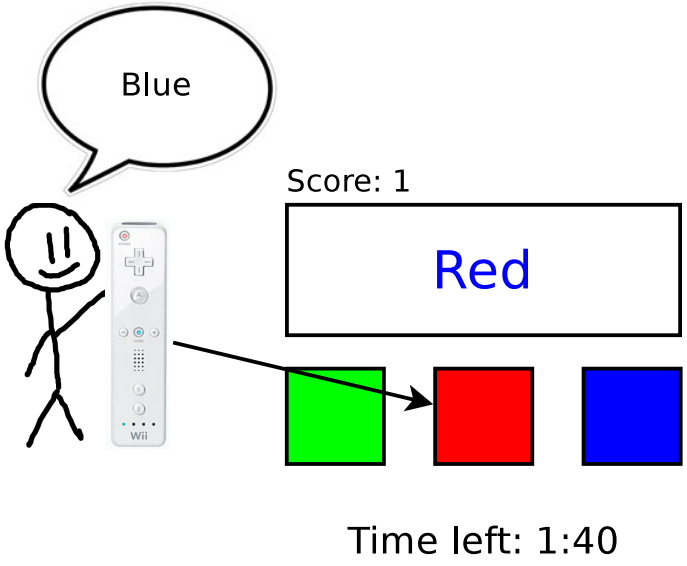

FIGURE 2: Après premier score

1.3 Contraintes

Les contraintes pour ce mini projet sont :

- Utilisation d'au moins 2 modalités
- Le matériel mis à disposition pour les projets est :
  - Kit Phidgets, manette Wii, microphone et camera
  - D'autre matériel peut également être utilisé

2 Analyse

Dans ce chapitre nous allons identifier les différents cas d'utilisations (use cases) lié à notre système multimodale. Après d'avoir spécifié les cas d'utilisations, nous allons découvrir les éléments hard- et software nécessaire pour pouvoir capturer et fusionner les signaux envoyés par l'utilisateur.

2.1 Cas d'utilisation

2.1.1 Acteurs

Deux acteurs principaux de notre application :

- Utilisateur (Être humain)
- Système multimodale

2.1.2 Diagramme de cas d'utilisation

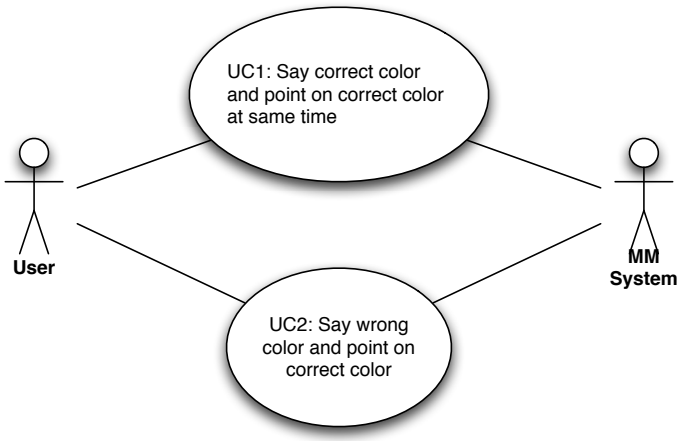

FIGURE 3: Diagramme de cas dútilisation

2.1.3 Description des cas d'utilisation

UC1 : Dire couleur correcte et pointer sur la couleur correcte
Quand l'utilisateur dit la couleur correcte et pointe sur la couleur correcte en même temps il obtient un point de score, et la couleur du mot et le mot lui-même (valeur) serons changés par le système multimodale pour que l'utilisateur puisse dire/pointer une nouvelle couleur. La couleur et la valeur du mot sont chaque fois changés aléatoirement.

UC2 : Dire couleur fausse et pointer sur la couleur correcte
Si l'utilisateur dit une couleur fausse mais pointe sur la couleur correcte, un retour est envoyé à l'utilisateur pour lui indiquer qu'il a dit une fausse couleur. La couleur et la valeur du mot affiché ne changent pas après le message d'erreur. L'utilisateur sera obligé de redire la couleur correcte et de pointer sur la couleur correcte.

2.2 Spécification des périphériques

Comme périphérique de pointage nous avons décidé de prendre la **Wiimote**. La Wiimote a besoin de quatre sources de lumière pour pouvoir calculer sa position x, y et z. Nous avons besoin de ces informations pour pouvoir détecter si le pointeur guidé par l'utilisateur se trouve dans la zone de couleur correcte. Les coordonnées sont envoyées au système multimodale en utilisant l'interface bluetooth.

Pour faire la capture de la voix nous avons besoin d'un simple **microphone**. Pour une utilisabilité plus agréable et une distance minime entre microphone et utilisateur, nous avons choisit un **microphone** de type headset qui utilise l'interface bluetooth.

2.3 Diagramme d'interaction

Nous avons identifié les cas d'utilisation et les périphériques correspondant qui permettent de réaliser le jeu. A partir d'un exemple concret nous allons maintenant montrer le cycle de vie d'une interaction utilisateur. Prenons le mot "green" qui est coloré en rouge. L'utilisateur doit donc dire le mot "red" et pointer avec la Wiimote sur la couleur verte. Le son de la voix et la position de la Wiimote sont capturés et envoyés au moteur de fusion qui interprète les signaux. Une fois les signaux entrants fusionnés, le système multimodale doit valider si l'utilisateur a dis/pointé la couleur correcte. Si l'utilisateur a réussit à dire/pointer la bonne couleur, le changement aléatoire de la nouvelle couleur/mot s'effectue.

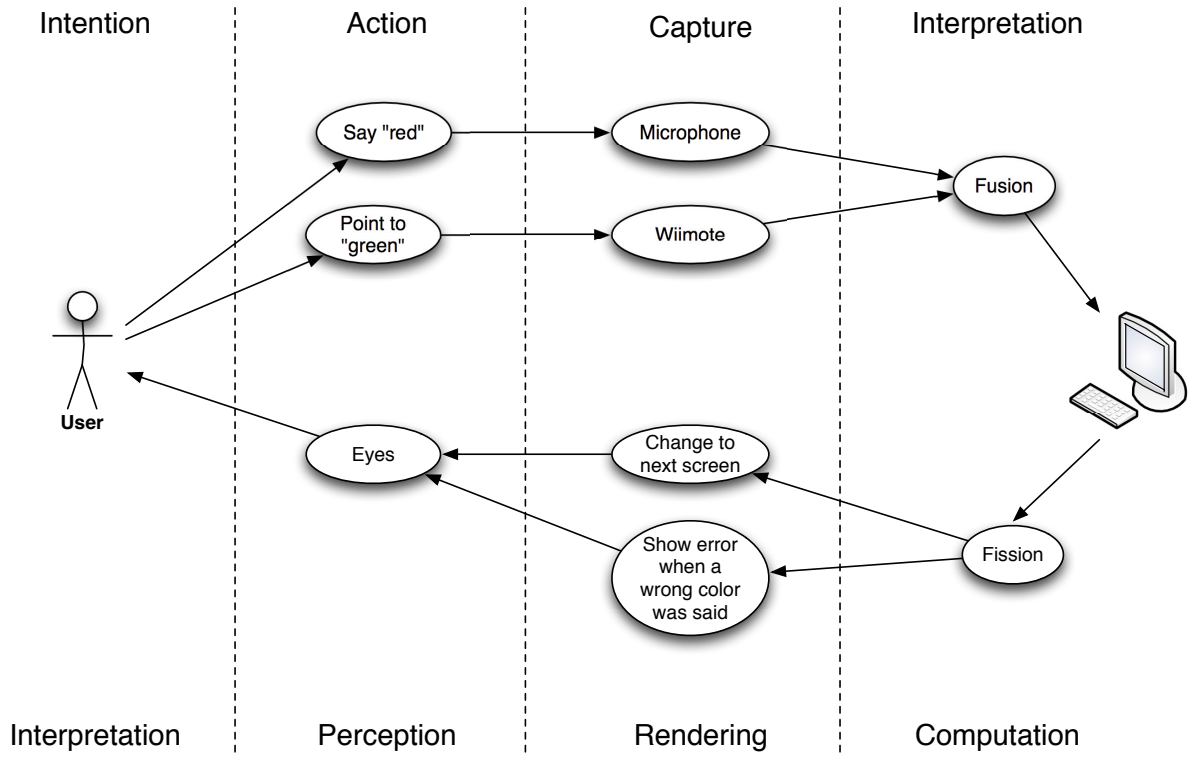

FIGURE 4: Diagramme d'interaction

2.4 Spécification du framework

2.4.1 Langage de programmation

Après avoir fait une recherche des différents framework qui existent actuellement nous avons décidé de développer notre application avec "Microsoft .NET" en utilisant C# comme langage de programmation.

2.4.2 Bibliothèques utilisées

A part les bibliothèques standards du framework .NET nous avons utilisé l'API "wiimote-api" qui se trouve sur `http://code.google.com/p/wiimote-api/` et l'API System.Speech que nous avons simplement dû importer dans le projet .NET.

2.5 CARE & CASE

CARE et CASE permettent d'identifier les types de modalités du côté humain et machine :

CARE (côté humain)

- **C**omplementarity : Deux tâches indépendantes qui s'exécutent en même temps pour atteindre un état spécifique.
- **A**ssignment : Pas de choix d'autres modalités.
- **R**edundancy : Deux tâches distinctes qui ont le même effet. Les tâches peuvent s'exécuter en même temps (AND).
- **E**quivalence : Deux tâches distinctes qui ont le même effet. Les tâches ne peuvent pas s'exécuter en même temps (OR)

Notre application réalise les modes "Complementary" et "Equivalent" de CARE. Quant l'utilisateur dit une couleur, il doit en même temps (complémentaire) pointer avec la manette Wii sur la valeur de la couleur pour obtenir des points de score. A la place de la manette Wii il est possible d'utiliser la souris comme périphérique de pointage. Ceci est considéré comme "Equivalence" dans CARE.

CASE (côté machine)

- **C**oncurrent : Deux tâches indépendantes qui s'exécutent en même temps.
- **A**lternate : Une tâche qui s'effectue en deux ou plusieurs opérations exécutées une après l'autre
- **S**ynergistic : Deux différentes tâches liées exécutées en même temps.
- **E**xclusive : Deux tâches indépendantes exécutées une après l'autre.

Du côté CASE nous respectons l'aspect "Synergistic". La logique de notre programme doit fusionner et gérer ce que l'utilisateur dis et en même temps elle doit vérifier la position de la manette Wii.

3 Architecture

3.1 Identification des classes

Dans cette section nous allons citer les classes les plus importantes de notre application. Voici le diagramme de classe simplifié :

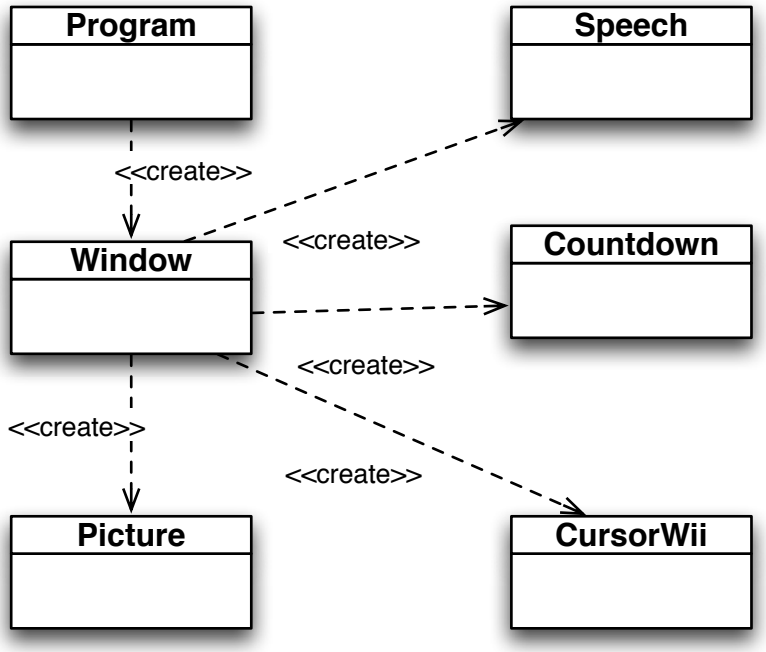

FIGURE 5: Diagramme de classe

Description des classes :

3.1.1 Program

Crée une instance de *Window*.

3.1.2 Window

Démarre *CursorWii*, *Speech*, *Picture* et *Countdown*

3.1.3 CursorWii

Utilise l'API mentionné à la section 2.4.2 pour pouvoir implémenter le comportement du périphérique Wiimote. L'utilisateur de cette classe doit invoquer la méthode *start()* après l'instanciation de la classe. Cette méthode *start()* démarre un nouveau thread. Le

thread "CursorWii" permet d'extraire les informations envoyées par la Wiimote. Dès que l'utilisateur pointe sur une couleur correcte, le thread Wiimote change en état *pointerIn-Rectangle*.

3.1.4 Speech

Utilise l'API *System.Speech* qui est contenue de base avec Windows Vista ou Windows 7. L'utilisateur de cette classe doit invoquer la méthode *start()* après l'instanciation de la classe. Cette méthode *start()* démarre un nouveau thread. Le thread "Speech" traite les signaux d'entrée du microphone. Il filtre les mots "Red", "Green", "Blue" et "Yellow" prononcés par l'utilisateur. Si un mot est détecté, le thread s'occupe du chargement du nouveau mot avec une nouvelle couleur ou dans le cas d'un mot incorrecte il s'occupe de produire un message d'erreur.

3.1.5 Picture

La classe *Picture* sert à dessiner les rectangles sur lesquels l'utilisateur doit pointer.

3.1.6 Countdown

Un simple *Timer*, inclut de base avec .NET C#, qui détecte la fin du jeu. Le timer est exécuté dans un nouveau thread.

4 Design

Dans le chapitre "Design" nous allons décrire le fonctionnement principal de la détection de la voix et de la position de la Wiimote.

4.1 Détection de la voix

L'API *Speech* de Microsoft fonctionne avec un vocabulaire de dictionnaire pré-entrainé. C'est-à-dire que le système reconnaît déjà tout les mots du dictionnaire commun. Pour limiter le choix des mots que l'utilisateur peut dire pour que le système réagisse, il et possible de créer un dictionnaire personnalisé.

Pour finalement détecter la voix humaine, nous procédons en plusieurs étapes :

1. Initialisation et définition du dictionnaire
2. Enregistrement pendant X secondes
3. Retirage des mots qui se trouvent dans le dictionnaire
4. Validation des mots par rapport aux règles du jeu
5. Rendement
6. Retour an point 2

4.2 Détection des coordonnés de la Wiimote

Pour détecter la position actuelle de la Wiimote il suffit d'utiliser l'API "wii-mote" mentionné à la section 2.4.2. Nous procédons en plusieurs étapes :

1. Enregistrement d'un listener (écouteur)
2. Système attend jusque l'utilisateur bouge la Wiimote
3. Appel de l'écouteur
4. Écouteur valide la position de la Wiimote
5. Si le pointeur de la Wiimote était sur la couleur correcte, nous passons en état *pointerInRectangle*
6. Retour au point 2

Si nous débranchons la Wiimote du système, nous aimerons que l'utilisateur puisse toujours commander l'application avec la souris de l'ordinateur. Pour ça nous avons besoin de la librairie *user32.dll* qui nous permet d'avoir accès aux cordonnées de la souris.

5 Implémentation

Dans ce chapitre nous allons illustrer la partie du code la plus importante pour la détection de la voix et de la position de la Wiimote. Nous ne commentons pas le code ci-dessous. Le principe de fonctionnement est comme décrit à la section 4.1.

5.1 Détection de la voix

5.1.1 Définition du dictionnaire

Listing 1: Speech.cs : Préparation

```
private void prepareSpeech() {
        grammar = createSampleGrammar();
        recognitionEngine = new SpeechRecognitionEngine();
        recognitionEngine.SetInputToDefaultAudioDevice();
        recognitionEngine.LoadGrammar(grammar);
}
```

Listing 2: Définition du dictionnaire

```
private Grammar createSampleGrammar() {
            Choices commandChoices;

            // prepares choices according to operating-system language
            switch (Configuration.getOsLanguage())
            {
                case "English":
                    Console.WriteLine("ENGLISH");
                    commandChoices = new Choices("Red", "Blue", "Green", "Yellow");
                    break;
                case "francais":
                    Console.WriteLine("FRANCAIS");
                    commandChoices = new Choices("Rouge", "Bleu", "Vert", "Jaune");
                    break;
                default:
                    Console.WriteLine("DEFAULT");
                    commandChoices = new Choices("Red", "Blue", "Green", "Yellow");
                    break;
            }
```

```
            // creates grammar
            Grammar g = new Grammar(commandChoices);
            g.Name = "Available programs";

            // returns grammar
            return g;
        }
```

5.1.2 Enregistrement et validation

Listing 3: Enregistrement et reconnaissance

```
private void speechThreadJob() {
    bool valid = false;
    while (countdown.getTimer().Enabled)
    {
        valid = false;

        result = recognitionEngine.Recognize(new TimeSpan(0, 0, 2));

        if (result != null && result.Words.Count != 0)
        {
            foreach (RecognizedWordUnit word in result.Words)
            {
                customLog(word);

                // checks answer
                if (word.Text.Equals(Configuration.convertColorName(window.getColorLabel().
                    ForeColor.Name)) && picture.pointerInRectangle(Cursor.Position))
                {
                    // update score
                    int score = Int32.Parse(window.getScore().Text);
                    score++;
                    window.getScore().Text = score.ToString();

                    // correct answer
                    valid = true;
                    window.getSpeechValid().Image = null;

                    // generates and displays next word
                    window.BeginInvoke(new Window.randomLabelDelegate(window.randomLabel));
                }
            }

            // user doesn't give the correct answer
            if (!valid)
            {
                window.getSpeechValid().Image = new Bitmap("error.png");
            }
        }
    }
}
```

5.2 Détection des coordonnées de la Wiimote

Listing 4: Détection de la position

```
private void cursorThreadJob() {
    wii = new Wiimote();
    wii.WiimoteChanged += wii_WiimoteChanged;

    try {
        // connect Wiimote
        wii.Connect();

        // retreive accel and IR infos
        wii.SetReportType(InputReport.IRAccel, true);
    }
    catch (WiimoteNotFoundException) {
        return;
    }

    state = wii.WiimoteState;

    // display battery level
```

```
        displayBattery();

        // control the position pointed by the Wiimote
        while (window.getRunning()) {
            ir = wii.WiimoteState.IRState;
            Cursor.Position = new System.Drawing.Point(System.Convert.ToInt32(((ir.IRSensors
                [0].RawPosition.X - 1023) * -1) * Configuration.ratioX), System.Convert.ToInt32
                (ir.IRSensors[0].RawPosition.Y * Configuration.ratioY));
        }

        wii.Disconnect();
    }
```

Listing 5: Listener

```
    // listener called if Wiimote state has changed
    private void wii_WiimoteChanged(object sender, WiimoteChangedEventArgs args) {
        WiimoteState ws = args.WiimoteState;
        if (ws.ButtonState.A)
        mouse_event(MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0);
        else if (!ws.ButtonState.A)
        mouse_event(MOUSEEVENTF_LEFTUP, 0, 0, 0, 0);
    }
```

6 Tests utilisateurs

Nous avons implémenté les langues anglais et français. Pour pouvoir tester le comportement et surtout la détection de la voix, nous avons pris des sujets de plusieurs langues maternelles différentes pour tester notre reconnaissance de voix. La conclusion de nos tests était positive. Le comportement principale de l'application fonctionne. Mais nous avons constaté que chez les personnes qui n'étaient pas de la langue maternelle anglaise ou française, la reconnaissance fonctionnait moins bien. Sur la vidéo de démonstration qui est sur le CD en annexe, nous voyons une personne germanophone qui dit les couleurs en anglais. Au début de la vidéo la personne doit dire quatre fois le mot "red" pour que le système le détecte. Avec des personnes de langue maternelle anglaise nous n'avons pas ce problème.

Pour pouvoir améliorer le problème des différents accents et dialectes il faudra améliorer la reconnaissance des mots qui se trouvent dans le dictionnaire. Ceci pourrait se faire à l'aide des réseaux de neurones électronique entrainés. Dans le cadre de ce projet il n'y a pas assez de temps pour pouvoir réaliser ceci.

7 Conclusion

7.1 Trouver la bonne idée

Au début de ce projet nous avons dû chercher des idées pour une réalisation de jeu multimodale. Pour nous il n'était pas évident de savoir qu'est-ce qui est faisable dans les 8 x 4 heures que nous avons à disposition pour ce projet. Il s'agissait alors de trouver une idée de jeu qui est multimodale et qui est réalisable dans les délais. Grasse à la simplicité du jeu choisit, nous avons pu réaliser entièrement ça que nous avions planifié.

7.2 Trouver le bon framework

Vu que nous n'avons pas eu d'expérience de développement multimodale, nous avons dû d'abord rechercher les différents frameworks qui existent actuellement sur le marché. Avec le framework .NET nous avons fait le bon choix. Il nous permettait de rapidement développer la détection de la voix et de la position de la Wiimote. Malgré tout, à notre avis, il n'y a pas beaucoup de sujets sur Internet qui parlent concrètement de la reconnaissance de la voix, et ça nous prenait quand même environ 8 heures jusqu'à ce que nous ayons pu faire une première détection de la voix.

7.3 But final

Le but final est pour nous considéré comme réussit. La reconnaissance de la voix et le pointage de la manette Wii sont détectés correctement, et le retour à l'utilisateur fonctionne.

7.4 Expérience

Pour nous, ce mini-projet était une bonne occasion pour pouvoir étendre nos connaissances en C#. La Wiimote et la reconnaissance de voix était un domaine complètement inconnu pour nous. Ce projet nous montrait qu'il est possible d'utiliser aussi d'autres périphériques que clavier et souris.

7.5 Améliorations possibles

Pour améliorer notre jeu il serait possible d'ajouter une couche de reconnaissance de voix qui étend celle de Microsoft. Les mots "jaune", "vert", "bleu" et "rouge" seront entrainés sur le système par rapport à des utilisateurs spécifiques.

Pour encore rendre le jeu plus difficile nous pouvons imaginer que les objets colorés sur lesquels l'utilisateur pointe bougent en permanence. En plus nous pourrons imaginer de pénaliser l'utilisateur au cas où il dit une couleur incorrecte. Un changement de mode (dire le mot écrit et pointer la couleur du mot) serait aussi une extension possible.