2017-2018

Manuel technique

TPI – Juin 2018

Finger's Cloner

Loïc Dubas
I.FA-P3B – CFPT-I

Gestion des versions

Date	Modifications	N° de version
04.06.2018	Création du document	0.1
05.06.2018	Écriture de l'introduction et des objectifs	0.2
06.06.2018	Analyse concurrentielle, organisations et planning initial	0.3
07.06.2018	Cahier des charges et maquettes	0.4
08.06.2018	Détermination des différentes parties, analyse de l'environnement et maquettes graphiques	0.5
11.06.2018	Plan de tests	0.6
12.06.2018	Rapport de tests	0.7
13.06.2018	Capture d'écrans, conception du code, diagramme de classe	0.8
14.06.2018	Conclusion et annexe	0.9
15.06.2018	Finalisation du document	1.0

Table des matières

1. Analyse préliminaire

1.1 Introduction

1.1.1 Présentation de l'application

Finger's cloner est un programme de reconnaissance de position de main destiné à des fins d'apprentissage. Il permet à l'utilisateur de créer des positions de doigts ou de tenter de copier une position donnée.

1.1.2 Fonctionnement

Grâce au capteur du « *Leap Motion* », cette application affiche la position de votre main en temps réel. Elle affiche aussi une position de main qui correspond au modèle que vous devrez tenter de copier au mieux.

Le *Leap Motion* est un tracker de main. Grâce à une lumière infrarouge, il détecte la paume, les doigts et chacune des phalanges. Intégré à son SDK, il peut détecter si les doigts sont pliés ou non, ou encore s'il y a un « pincement », entre l'index et le pouce par exemple. Avec les données reçues, et avec son intégration de réalité virtuelle, il localise la position de la main dans l'espace situé au-dessus du capteur

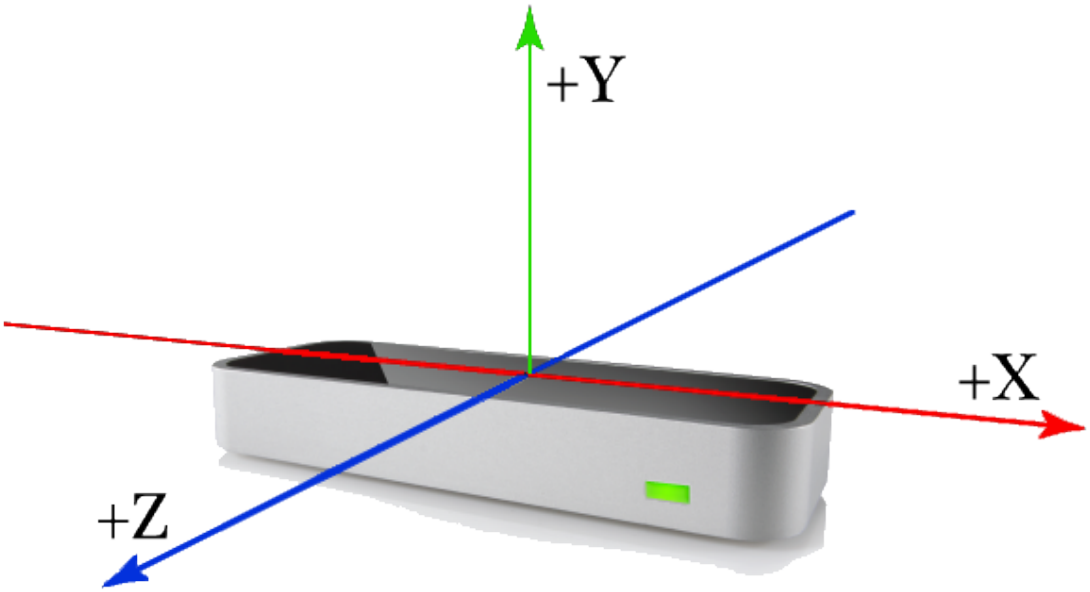

Figure 1. Le Leap Motion utilise trois axes pour localiser la position de la main.

1.1.3 Pourquoi ce projet ?

J'ai choisi de faire ce programme, car je voulais trouver un projet avec une réelle utilité. L'apprentissage du langage des signes, par exemple, peut être bénéfique à tous, et ce projet offre une première version d'une application pouvant s'adapter facilement à cet apprentissage. Dans le domaine de la rééducation aussi, on peut imaginer une évolution du logiciel où le médecin pourrait préenregistrer des positions de main adaptée à son patient.

1.2 Organisation

1. Élève

- Loïc Dubas

2. Maître d'apprentissage

- M. Garchery

3. Experts

- M. Conrad
- M. Bernard

Dès le 4 juin 2018, ma séance de TPI débutera officiellement. Dès lors, j'aurai 10 jours pour tenter de mettre au point mon projet. Avec 8 heures par jour, cela me laisse 80 heures au total pour mener à bien ce travail.

Pour garder une sauvegarde de mon projet de façon sécurisée et simple, j'utilise GitHub. Cela permet, en quelques clics, de faire des copies de mon projet en ligne, ainsi je ne crains pas de panne logicielle ou matérielle. GitHub permet aussi de revenir à d'anciennes versions si besoin est.

Pour démarrer sur une base concrète, les maquettes sont la première chose que j'ai faite. Cela m'a aidé à visualiser ce dont j'avais besoin pour mener à bien mon programme.

1.3 Objectifs

L'objectif principal de mon logiciel est de mettre au point un programme de reconnaissance de position des 5 doigts de la main et de la paume. Il doit permettre à l'utilisateur de sauvegarder une position personnalisée avec un nom, une description et une image. La fenêtre principale affiche sa main en temps réel et le modèle à recopier, avec son nom, sa description et son image, bien entendu. Un code couleur (noir, rouge, orange et vert) indique à l'utilisateur si la position de chaque doigt s'approche de celle du modèle.

D'un point de vue personnel, ce logiciel me permet de confirmer mes compétences en programmation C#, mais aussi ma compétence à apprendre par moi-même. Ainsi, je vais devoir prendre en main le Leap Motion et ses fonctions dans sa version 3 (nommée *Orion*), mais aussi à calculer des positions relatives à la fenêtre en se basant sur les positions normalisées par le Leap Motion.

1.4 Planification initiale

	Jour 1	Jour 2	Jour 3	Jour 4	Jour 5	Jour 6	Jour 7	Jour 8	Jour 9	Jour 10
Installation du Leap Motion										
Paramétrage du Leap Motion										
Création de l'interface										
Capture de la main + affichage										
Création de la partie « *Appreneur* »										
Création de la partie « *Apprentissage* »										
Finissions										
Documentation										
Manuel technique										

Ci-dessus, j'ai tenté de planifier mon projet en essayant de m'imaginer les différentes étapes et leurs durées de réalisation. L'installation et le paramétrage du Leap Motion ne me prennent que peu de temps, car j'ai pris le temps de m'informer sur l'appareil en général avant de commencer ce travail.

La création de l'interface ne consiste qu'à poser les éléments sur mes différentes fenêtres.

La capture de la main et son affichage devrait être assez simple. Le Leap Motion renvoie directement les coordonnées de chacun des doigts, de façon normalisée si nécessaire. Son affichage ne devrait consister qu'à une mise à l'échelle par rapport à mon élément d'affichage.

La partie « *appreneur* » gère la sauvegarde des positions de la main. Les coordonnées de chaque doigt et de la paume doivent être enregistrées. Je ne sais pas vraiment comment sérialiser des objets multiples, alors je me réserve quelques jours pour tenter de réaliser par moi-même l'enregistrement.

La partie « *apprentissage* » compare la position actuelle de l'utilisateur au modèle sélectionné. Je ne sais pas encore quelle logique je vais appliquer pour ce faire. Je me garde suffisamment de jour pour cela aussi.

Les finissions consisteraient à « *nettoyer* » tous les bouts de codes qui peuvent être soient supprimés, soient simplifiés, soient unifiés pour s'adapter à différentes situations.

La documentation et le manuel technique doivent évoluer quotidiennement en fonction des avancées du projet.

2. Analyse

2.1 Analyse concurrentielle

Plusieurs personnes se sont déjà penchées sur un programme de reconnaissance de signe de main avec le Leap Motion, mais aucune application commercialisable n'existe. Ces personnes ont pour la plupart codé en Python et elles ne proposent que peu d'interface.
De plus en plus d'établissements s'intéressent à mêler détection de mouvements et traduction en temps réel en langage des signes. Pour l'instant, il n'existe que des ébauches de projet.

- Sign Language Tutor est un programme codé en Python par un certain S. Taylor. Dans le cadre d'un hackathon, lui et son équipe ont réussi à terminer deuxièmes de la compétition. Le code source est disponible à l'adresse suivante : https://github.com/ssaamm/sign-language-tutor, dernière consultation : 14.06.2018.
- Deepali Naglot et Milind Kulkarni sont deux ingénieurs qui ont écrit un article proposant l'utilisation du Leap Motion pour traduire en temps réel le langage des signes. Leur article est disponible en suivant ce lien : https://ieeexplore.ieee.org/document/7830097/, dernière consultation : 14.06.2018.

2.2 Cahier des charges détaillé

2.2.1 Définition de l'audience

Mon projet s'intéresse en premier lieu aux personnes voulant apprendre le langage des signes et celles en rééducation de la main. Grâce à sa simplicité d'utilisation, l'application peut convenir à tous ceux intéressés à apprendre une nouvelle façon de s'exprimer, et ce, peu importe l'âge.

2.2.2 Définition du contenu et de fonctionnalités

- Capture de la main
- Affichage de la main
- Enregistrement :
    - De la position des doigts et de la paume
    - D'un nom
    - D'une description
    - D'une illustration
- Affichage de la position enregistrée
- Modification des informations de la position (nom, description, illustration)
- Suppression d'une position
- Sérialisation de plusieurs positions
- Récupération des positions enregistrées
- Affichage de toutes les positions dans une liste déroulante

2.2.3 Maquette préliminaire

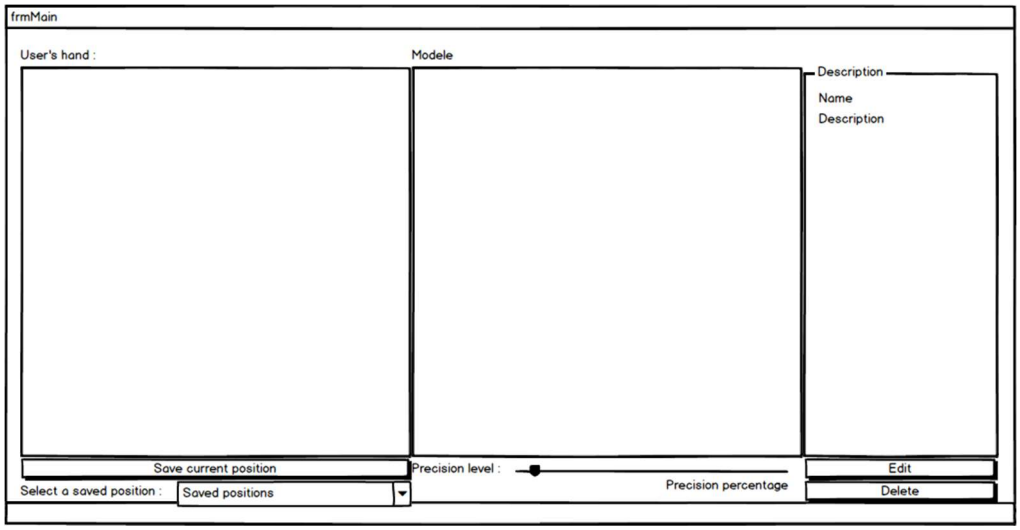

Figure 2. Fenêtre principale

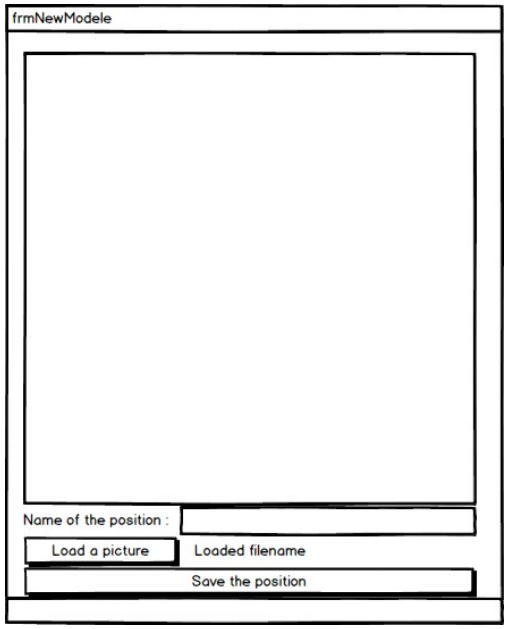

Figure 3. Fenêtre Nouveau Modèle

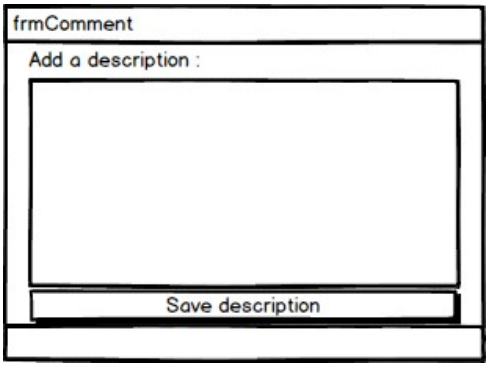

Figure 4. Fenêtre Commentaire

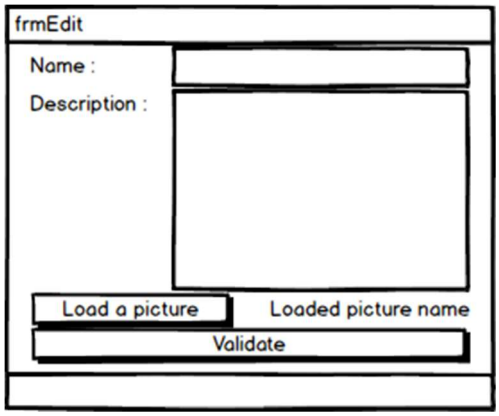

Figure 5. Modifier une position

3. Conception

3.1 Analyse de l'environnement

Pour m'aider dans la conception de mon projet, j'utilise plusieurs softwares :

- **Visual Studio 2017**, en tant qu'environnement de développement C#, qui me permet de faire un programme avec des composants visuels faciles à mettre en place.
- **LeapMotion SDK** est obligatoire pour pouvoir utiliser le capteur de mouvement *Leap Motion*, mais il offre aussi des samples permettant de visualiser facilement les informations reçues par l'appareil.
- **GitHub**, en tant que gestionnaire de version, qui me permet de garder une trace de l'évolution du logiciel et qui me permet de faire des sauvegardes facilement.
- **Word** comme éditeur de texte ce qui me permet d'écrire les manuels nécessaires.
- **Balsamiq** me permet de créer des maquettes de mon programme.

3.2 Détermination des différentes parties

La fenêtre principale (**frmMain.cs**) contiendra deux panel affichant respectivement la main de l'utilisateur et le modèle à copier. Sur le bord droit de la fenêtre, le nom, la description et l'image de la position seront affichés. En dessous de cela, deux boutons permettront de modifier et de supprimer la position en cours. En bas à gauche, une liste déroulante proposera toutes les positions préenregistrées. Un bouton en bas de cette liste permettra d'enregistrer la position actuelle de l'utilisateur. En bas à droite, un slider permettra de choisir le niveau de précision requis pour que la position de l'utilisateur soit considérée comme correcte par rapport au modèle (cf. Figure 6).

Une seconde fenêtre (**frmNewModel.cs**) s'ouvrira lorsque l'utilisateur cliquera sur le bouton « Enregistrer la position ». L'utilisateur pourra choisir un nom et une image pour la position. En cliquant sur « Enregistrer », une autre fenêtre (**frmComment.cs**) s'ouvrira pour que l'utilisateur propose une description à sa position (cf. Figure 7).

Une dernière fenêtre (**frmEdit.cs**) est la fenêtre qui s'ouvrira quand l'utilisateur voudra modifier la position en cours. Il pourra changer le nom, la description et l'image de celle-ci (cf. Figure 8).

3.3 Maquette graphique

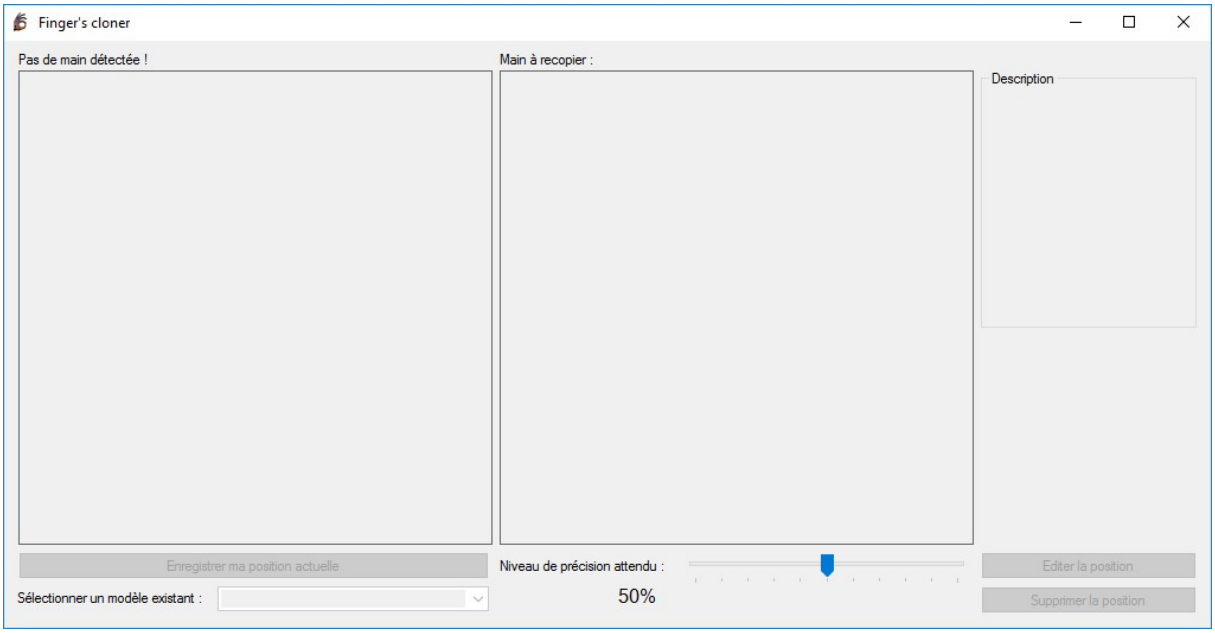

Figure 6. Fenêtre principale

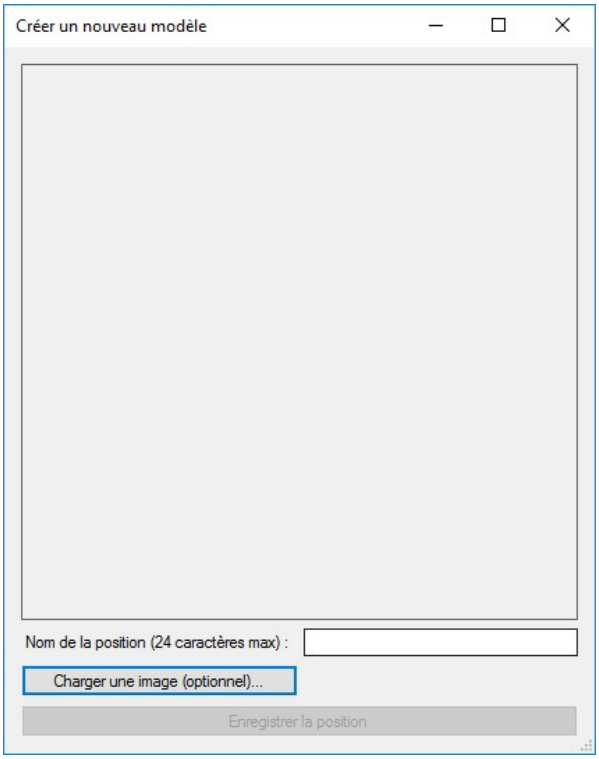

Figure 7. Fenêtre de création de modèle

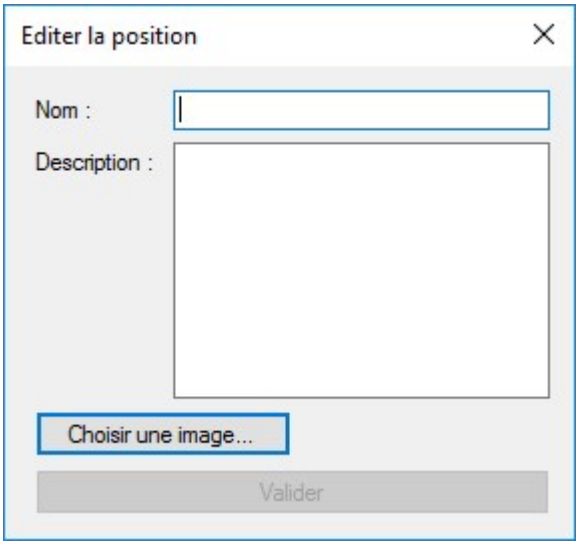

Figure 8. Fenêtre d'édition de position

3.4 Conception du code

Mon code se sépare en plusieurs classes :

La première est celle qui permet de contrôler le Leap Motion. Nommée ***LeapController.cs*** (cf. Figure 18), elle récupère les informations de la main que le capteur détecte. Ces informations sont stockées grâce à la classe *MyHand.cs*.

```
/// <summary>
/// Refresh the fingers info on every frame of the Leap Motion
/// </summary>
/// <param name="sender"></param>
/// <param name="eventArgs"></param>
1 référence | loicdub, il y a 6 jours | 1 auteur, 3 modifications
public void newFrameHandler(object sender, FrameEventArgs eventArgs)
{
    Frame frame = eventArgs.frame;
    InteractionBox iBox = frame.InteractionBox;

    if (frame.Hands.Count > 0)
    {
        Hands = frame.Hands;
        FirstHand = Hands[0];

        PalmPos = FirstHand.PalmPosition;
        PalmNormPos = iBox.NormalizePoint(PalmPos);

        Fingers = FirstHand.Fingers;
        FingersStabPos = new List<Vector>();
        FingersNormPos = new List<Vector>();

        for (int i = 0; i < Fingers.Count; i++)
        {
            FingersStabPos.Add(Fingers[i].StabilizedTipPosition);
            FingersNormPos.Add(iBox.NormalizePoint(FingersStabPos[i]));
        }

        UserHand = new MyHand(PalmNormPos, FingersNormPos);
    }
}
```

Figure 9. Classe appelée à chaque frame qui récupère les informations de la main

MyHand.cs (cf. Figure 20) est la classe qui me permet de stocker les informations d'une main, que ce soit celle de l'utilisateur, celle qui s'affiche lorsque l'utilisateur veut enregistrer sa position ou encore celle qui s'affiche en tant que modèle. Son premier constructeur ne demande que la position des doigts et de la paume. Ce constructeur concerne directement la main de l'utilisateur. Un second constructeur demande les mêmes informations et un nom ainsi qu'une description en plus. Cela concerne la position à enregistrer et le modèle. Un troisième constructeur demande une illustration en plus. Ainsi, si l'utilisateur le souhaite, il peut proposer une photo à afficher avec une position donnée *(cf. Figure 10)*.

```
/// <summary>
/// MyHand constructor
/// </summary>
/// <param name="name">Name of the position</param>
/// <param name="description">Description of the position</param>
/// <param name="palmPosNorm">Normalized position of the palm</param>
/// <param name="fingersPosNorm">Normalized positions of the fingers</param>
/// <param name="image">Image of the position as a string</param>
0 références | loicdub, Il y a 1 jour | 1 auteur, 2 modifications
public MyHand(string name, string description, Vector palmPosNorm, List<Vector> fingersPosNorm, string image)
{
    this.Name = name;
    this.Description = description;
    this.PalmNormPos = palmPosNorm;
    this.FingersNormPos = fingersPosNorm;
    this.Image = image;
}
```

Figure 10. Le constructeur de MyHand

Serialization.cs (cf. Figure 22) ne sert qu'à sérialiser et déserialiser les fichiers XML permettant la sauvegarde des positions. Il reçoit une main en paramètre, une fois celle-ci reçue, il va la sérialiser dans le dossier « *Serial* », où toutes les autres positions sont stockées. Pour la déserialisation, il va prendre tous les fichiers XML du dossier et les déserialiser en objet MyHand *(cf. Figure 11)*.

```
/// <summary>
/// default constructor - initialize directory name
/// </summary>
3 références | 0 modification | 0 auteur, 0 modification
public Serialization()
{
    DirName = "serial";

    Path.GetFileName(DirName);
}

/// <summary>
/// serialize a given MyHand object
/// </summary>
/// <param name="Hand">the hand to serialize</param>
2 références | 0 modification | 0 auteur, 0 modification
public void serialize(MyHand Hand)
{
    PositionName = Hand.Name;
    FilePath = DirName + "/" + PositionName + ".xml";

    XmlSerializer serializer = new XmlSerializer(typeof(MyHand));
    StreamWriter file = new StreamWriter(FilePath);
    serializer.Serialize(file, Hand);
    file.Close();
}
```

Figure 11. Le constructeur de Serialization et la fonction de sérialisation

Paint.cs (cf. Figure 21) gère le dessin. À l'initialisation, il requiert la largeur et la hauteur de l'élément dans lequel il va devoir dessiner. Trois fonctions dessinent respectivement un rond, un trait et la dernière fonction place les deux ensembles pour former un doigt. Les trois mêmes fonctions existent en version colorée qui demande une couleur en paramètre pour dessiner les ronds de la bonne couleur. Une dernière fonction transforme les données de positions des doigts, originellement normalisées, en positions dites « réelles », c'est-à-dire la position exacte des doigts sur l'élément d'affichage de la main dépendamment de l'emplacement de la paume *(cf. Figure 12)*.

```
/// <summary>
/// Calculate the position on the panel with the normalized vector
/// </summary>
/// <returns>A list of vector with the finger's position to the palm</returns>
2 références | loicdubas, il y a 5 jours | 1 auteur, 1 modification
public List<Vector> normToPalmPanelPos()
{
    float scaleFactor = PanelHeight + CIRCLESIZE;
    List<Vector> fingersPanelPos = new List<Vector>();
    Vector originToPalm = new Vector(Hand.PalmNormPos.x, 0, Hand.PalmNormPos.z);
    List<Vector> originToFingers = new List<Vector>();

    for (int i = 0; i < Hand.FingersNormPos.Count; i++)
    {
        originToFingers.Add(new Vector(Hand.FingersNormPos[i].x, 0, Hand.FingersNormPos[i].z));

        fingersPanelPos.Add(new Vector((-originToPalm + originToFingers[i]) * scaleFactor + palmPanelPos));
    }

    return fingersPanelPos;
}
```

Figure 12. La fonction qui transforme les positions normalisées en position réelle (sur le panel)

La fenêtre principale, **frmMain.cs** (cf. Figure 23), reçoit la main du Leap Motion et celle enregistrée. Elle envoie les données à la classe Paint.cs qui permet de dessiner la main sur le panel souhaité. Le même principe se passe avec le modèle. Le panel de l'utilisateur est rafraîchi 10 fois par seconde pour assurer un maximum de fluidité sans trop surcharger l'ordinateur.

Le constructeur de la classe crée le dossier qui stockera les positions sérialisées, s'il n'existe pas. Il initialise le LeapController, la classe de sérialisation et la classe Paint à laquelle il envoie les dimensions du panel. Il met à jour la liste déroulante pour afficher les positions enregistrées et le modèle et ses informations (cf. Figure 13).

```
/// <summary>
/// default constructor
/// </summary>
1 référence | loicdub, Il y a 1 jour | 2 auteurs, 11 modifications
public frmMain()
{
    InitializeComponent();

    // create the serial folder if not exist to store saved positions
    Directory.CreateDirectory("serial");

    DoubleBuffered = true;

    // initialize the leap controller
    leapController = new LeapController();

    // initialize serialization class
    savedPositions = new Serialization();
    // initialize paint class
    paint = new Paint();
    // send panel dimensions to paint class
    paint.GetPanelSize(pnlUserHand.Width, pnlUserHand.Height);

    updateCombobox();
    updateModele();

    // get value of trackbar
    precision = trackBar1.Value;
}
```

Figure 13. Le constructeur de la classe frmMain.cs

```
/// <summary>
/// Calculate distance between each fingers of user's and modele's hand
/// </summary>
/// <returns>A list of distances between each fingers</returns>
1 référence | loicdub, Il y a 1 jour | 2 auteurs, 2 modifications
private List<double> comparePosition()
{
    handsDiff = new List<Vector>();
    fingersDist = new List<double>();
    List<Vector> modelePanelPos = paint.normToPalmPanelModelePos(modeleHand);

    for (int i = 0; i < paint.FingersPanelPos.Count; i++)
    {
        handsDiff.Add(paint.FingersPanelPos[i] - modelePanelPos[i]);
        fingersDist.Add(Math.Sqrt(
            (Math.Pow(handsDiff[i].x, 2)) + (Math.Pow(handsDiff[i].z, 2))
            ));
    }

    return fingersDist;
}
```

Figure 14. Fonction de comparaison de la position de la main et du modèle affiché

La fenêtre de création de nouveau modèle, **frmNewModele.cs** (cf. Figure 24), reçoit en paramètre une main, celle de l'utilisateur, dont il enregistre la position des doigts et de la paume. L'utilisateur peut proposer un nom et une image. Lorsque l'utilisateur clique sur « Enregistrer », la fenêtre **frmComment.cs** (cf. Figure 25) s'ouvre et permet à l'utilisateur d'ajouter une description à sa position. Une fois celle-ci validée, les fenêtres de création de modèle et de commentaire se ferment et la position est sérialisée. La fenêtre principale rafraîchit le menu déroulant et l'affichage du modèle.

```
/// <summary>
/// create new modele form
/// </summary>
/// <param name="fingersNormPos">finger's normalized position</param>
/// <param name="palmNormPos">palm's normalized position</param>
1 référence | loicdub, Il y a 1 jour | 1 auteur, 2 modifications
public frmNewModele(MyHand handToSave)
{
    InitializeComponent();
    DoubleBuffered = true;

    leapController = new LeapController();
    paint = new Paint();
    paint.GetPanelSize(pnlModele.Width, pnlModele.Height);
    serialization = new Serialization();

    this.currentPosition = handToSave;
}
```

Figure 15. Le constructeur de la classe frmNewModele.cs

Sur la fenêtre principale, si une position est chargée en tant que modèle, le bouton « Éditer la position » devient cliquable. La fenêtre **frmEdit.cs** (cf. Figure 26) s'ouvre. Elle permet à l'utilisateur de changer le nom, la description et l'image de la position. Quand l'utilisateur valide les changements, l'ancienne position est supprimée et la nouvelle est sérialisée (cf. Figure 17).

```
/// <summary>
/// default constructor
/// </summary>
/// <param name="modelHand">the position to edit</param>
1 référence | loicdubas, Il y a 1 jour | 2 auteurs, 2 modifications
public frmEdit(MyHand modelHand)
{
    InitializeComponent();

    handToEdit = modelHand;
    nameHandToEdit = modelHand.Name;
    imageHandToEdit = modelHand.Image;

    serialization = new Serialization();

    tbxName.Text = modelHand.Name;
    tbxDescription.Text = modelHand.Description;
}
```

Figure 16. Le constructeur de la classe frmEdit.cs

```
/// <summary>
/// edit the hand
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
1 référence | loicdubas, Il y a 1 jour | 2 auteurs, 2 modifications
private void btnValidate_Click(object sender, EventArgs e)
{
    handToEdit.Name = tbxName.Text;
    handToEdit.Description = tbxDescription.Text;
    if (loadedPicture == null)
    {
        imageAsString = imageHandToEdit;
    }
    else
    {
        handToEdit.Image = imageAsString;
    }

    serialization.deletePosition(nameHandToEdit);
    serialization.serialize(handToEdit);
}
```

Figure 17. L'événement sur le bouton "Enregistrer"

Sur la fenêtre principale, si une position est chargée en tant que modèle, le bouton « Supprimer la position » devient cliquable. En cliquant sur ce bouton, une fenêtre de confirmation s'ouvre pour s'assurer du choix de l'utilisateur. Si celui-ci confirme, la position est supprimée.

3.5 Diagrammes de classe

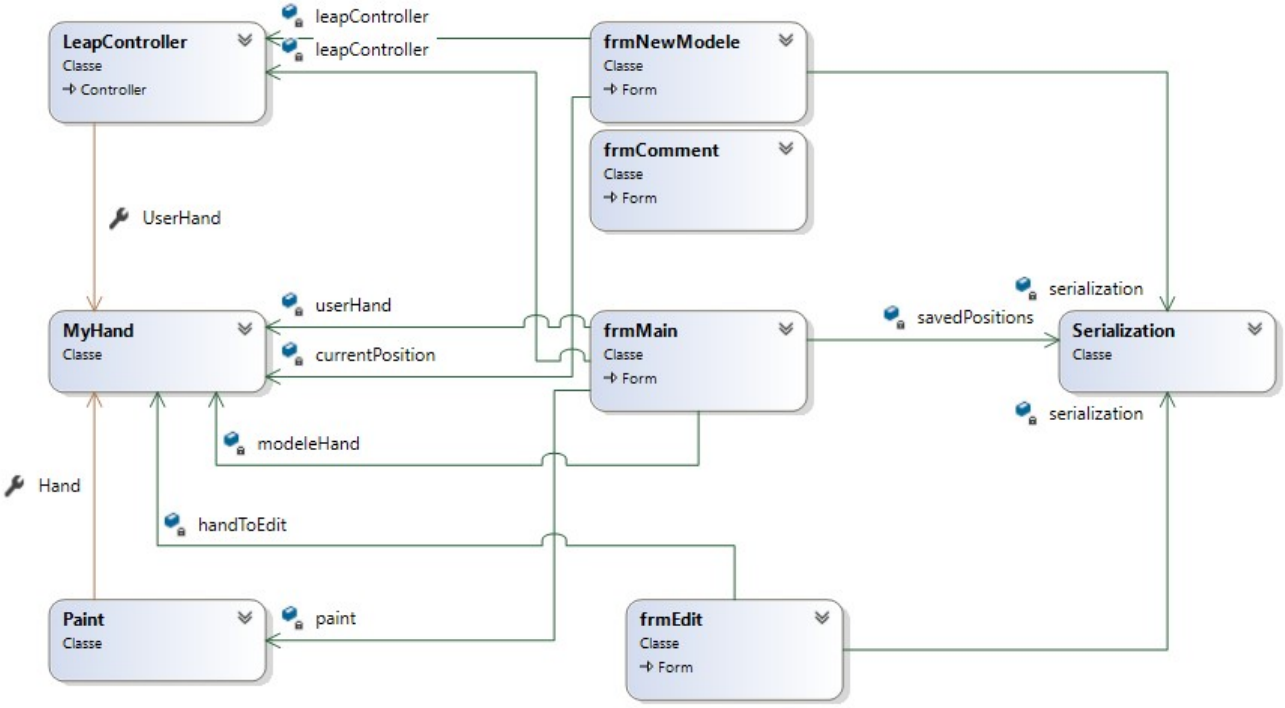

Figure 18. Diagramme de classe

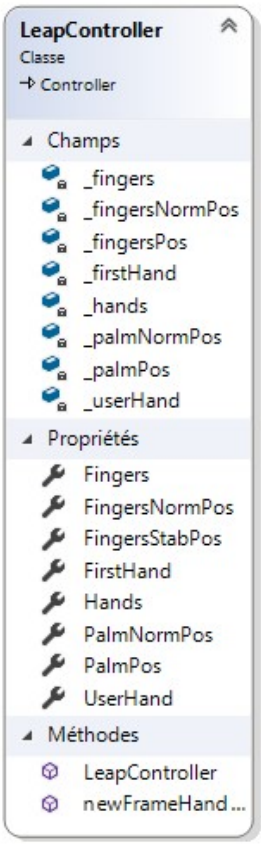

Figure 19. Classe LeapController.cs

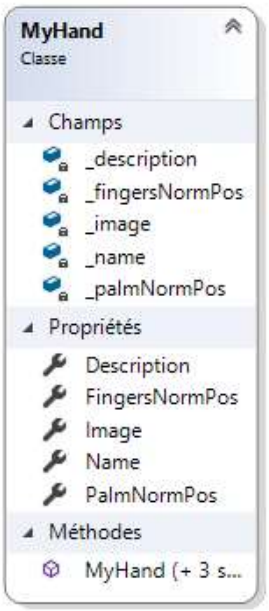

Figure 20. Classe MyHand.cs

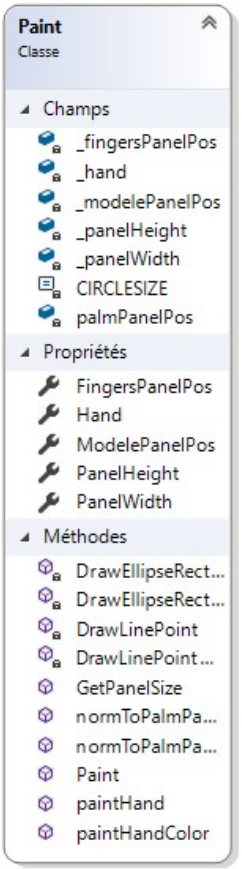

Figure 21. Classe Paint.cs

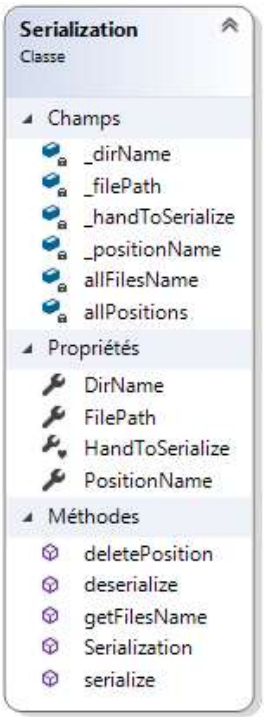

Figure 22. Classe Serialization.cs

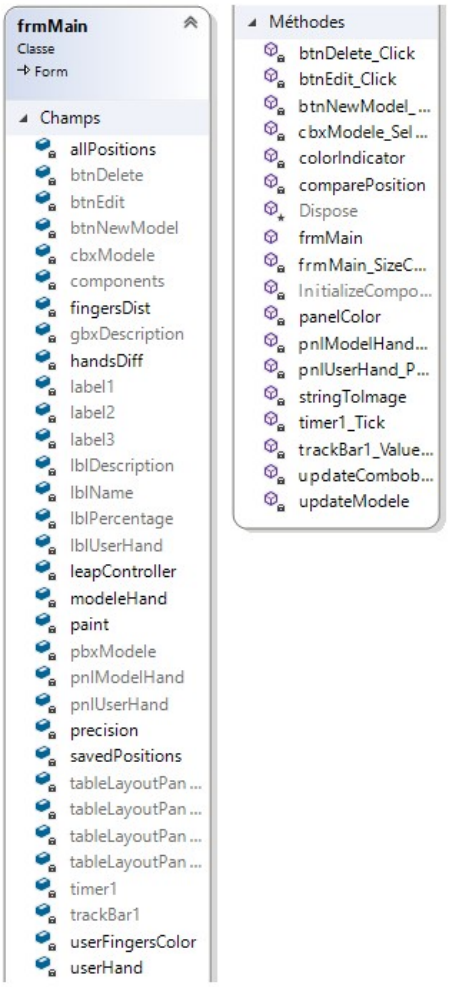

Figure 23. frmMain.cs

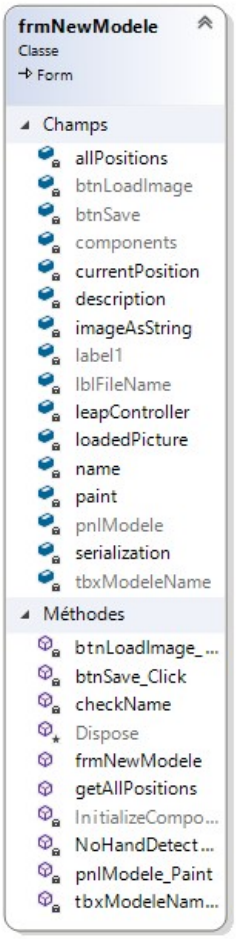

Figure 24. frmNewModele

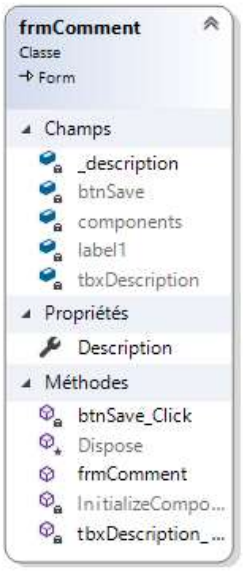

Figure 25. frmComment.cs

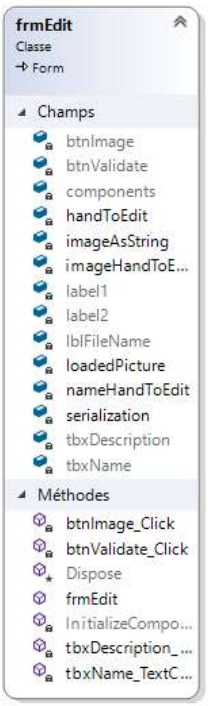

Figure 26. frmEdit.cs

4. Tests

4.1 Plan de tests

Environnement de test : Windows 10 Entreprise 64-bit

N°	Auteur	Scénario	Résultat attendu
1	Loïc Dubas	Passer la main au-dessus du Leap Motion	La main apparait sur le panel gauche.
2	Loïc Dubas	Cliquer sur « Enregistrer ma position actuelle »	Une fenêtre « Créer un nouveau modèle » s'ouvre.
3	Loïc Dubas	Entrer un nom dans la zone de texte	Le bouton « Enregistrer la position » devient cliquable.
4	Loïc Dubas	Cliquer sur « Charger une image (optionnel)… »	Un explorateur de fichier s'ouvre.
5	Loïc Dubas	Sélectionner une image	Le nom du fichier chargé s'affiche à côté du bouton « Charger une image (optionnel)… »
6	Loïc Dubas	Cliquer sur « Enregistrer la position »	Une fenêtre « Ajouter une description » s'ouvre.
7	Loïc Dubas	Entrer du texte dans la zone de texte.	Le bouton « Enregistrer la position » devient cliquable.
8	Loïc Dubas	Entrer un nom de position déjà utilisé par un autre modèle	Le bouton « Enregistrer la position » n'est plus cliquable.
9	Loïc Dubas	Cliquer sur « Enregistrer la position »	Les fenêtres « Créer un nouveau modèle » et « Ajouter une description » se ferment. Le nouveau modèle apparait dans la liste déroulante.
10	Loïc Dubas	Sélectionner une position dans la liste déroulante	La position sélectionnée, son nom, sa description et son image, s'il en a une, s'affichent.
11	Loïc Dubas	Cliquer sur « Éditer la position »	Une fenêtre « Éditer la position » s'ouvre avec le nom et la description préchargés.
12	Loïc Dubas	Changer le nom, puis cliquer sur Valider	La fenêtre se ferme et le nom de la position s'est mis à jour.
13	Loïc Dubas	Réitérer le test N°10 et changer la description	La fenêtre se ferme et la description de la position s'est mise à jour.
14	Loïc Dubas	Réitérer le test N°10 et changer l'image	La fenêtre se ferme et l'image de la position s'est mise à jour.
15	Loïc Dubas	Cliquer sur « Supprimer la position »	Une fenêtre demande une confirmation de suppression.
16	Loïc Dubas	Sur la fenêtre de confirmation de suppression, cliquer sur « Non »	La fenêtre de confirmation de suppression se ferme.
17	Loïc Dubas	Sur la fenêtre de confirmation de suppression, cliquer sur « Oui »	La fenêtre de confirmation de suppression se ferme et la position chargée n'apparait plus.
18	Loïc Dubas	Faire glisser le slider de précision	L'indicateur en pourcentage, juste en dessous du slider, s'adapte à la position du slider.
19	Loïc Dubas	Copier la position du modèle	La main « utilisateur » change de couleur si elle s'approche de la position du modèle.
20	Loïc Dubas	Copier la position du modèle avec le curseur de précision à 0%	Les doigts de la main « utilisateur » deviennent verts même si la position n'est pas exacte.

21	Loïc Dubas	Copier la position du modèle avec le curseur de précision à 100%	Les doigts de la main « utilisateur » restent noirs ou deviennent rouges si la position n'est pas exacte.

4.2 Rapport de tests

N°	Testeur	Date	OK / KO	Résultat obtenu
1	Loïc Dubas	12.06.2018	OK	La main apparait sur le panel gauche.
2	Loïc Dubas	12.06.2018	OK	Une fenêtre « Créer un nouveau modèle » s'ouvre.
3	Loïc Dubas	12.06.2018	OK	Le bouton « Enregistrer la position » devient cliquable.
4	Loïc Dubas	12.06.2018	OK	Un explorateur de fichier s'ouvre.
5	Loïc Dubas	12.06.2018	OK	Le nom du fichier chargé s'affiche à côté du bouton « Charger une image (optionnel)… »
6	Loïc Dubas	12.06.2018	OK	Une fenêtre « Ajouter une description » s'ouvre.
7	Loïc Dubas	12.06.2018	OK	Le bouton « Enregistrer la position » devient cliquable.
8	Loïc Dubas	12.06.2018	OK	Le bouton « Enregistrer la position » n'est plus cliquable.
9	Loïc Dubas	12.06.2018	OK	Les fenêtres « Créer un nouveau modèle » et « Ajouter une description » se ferment. Le nouveau modèle apparait dans la liste déroulante.
10	Loïc Dubas	12.06.2018	OK	La position sélectionnée, son nom, sa description et son image, s'il en a une, s'affichent.
11	Loïc Dubas	12.06.2018	OK	Une fenêtre « Éditer la position » s'ouvre avec le nom et la description préchargés.
12	Loïc Dubas	12.06.2018	OK	La fenêtre se ferme et le nom de la position s'est mis à jour.
13	Loïc Dubas	12.06.2018	OK	La fenêtre se ferme et la description de la position s'est mise à jour.
14	Loïc Dubas	12.06.2018	OK	La fenêtre se ferme et l'image de la position s'est mise à jour.
15	Loïc Dubas	12.06.2018	OK	Une fenêtre demande une confirmation de suppression.
16	Loïc Dubas	12.06.2018	OK	La fenêtre de confirmation de suppression se ferme.
17	Loïc Dubas	12.06.2018	OK	La fenêtre de confirmation de suppression se ferme et la position chargée n'apparait plus.
18	Loïc Dubas	12.06.2018	OK	L'indicateur en pourcentage, juste en dessous du slider, s'adapte à la position du slider.
19	Loïc Dubas	12.06.2018	OK	La main « utilisateur » change de couleur si elle s'approche de la position du modèle.
20	Loïc Dubas	12.06.2018	OK	Les doigts de la main « utilisateur » deviennent verts même si la position n'est pas exacte.
21	Loïc Dubas	12.06.2018	OK	Les doigts de la main « utilisateur » restent noirs ou deviennent rouges si la position n'est pas exacte.

5. Conclusion

Je pense avoir mené à bien ce projet. J'ai pu implémenter toutes les fonctionnalités exigées en me laissant assez de temps pour revoir et finaliser la documentation technique. Les difficultés rencontrées ont été surmontées grâce aux différents sites consultés (voir chapitre 6.1.1) et aux aides reçues (voir chapitre 6.1.2). Les sites consultés n'ont que fait m'indiquer la bonne façon de faire telle ou telle fonction bloquante, je m'en suis inspiré, mais rien n'a été copié tel quel, excepté les fonctions de dessin de rond et de trait.

Le planning initial a été respecté au mieux, mais contrairement à ce que j'avais prévu avant le début ce travail, la partie apprentissage et la partie « *appreneur* » ont été faites en parallèle. Hormis ces deux parties, les finitions m'ont pris deux jours. Le premier jour, je me suis concentré sur les finitions du programme, et le deuxième jour m'a permis de terminer la documentation technique et le manuel utilisateur.

La plus grande difficulté de ce travail a été de transformé les positions normalisées en positions relatives au panel.

6. Annexes

6.1 Sources

6.1.1 Bibliographie

- https://msdn.microsoft.com
- https://stackoverflow.com
- https://developer-archive.leapmotion.com/documentation/csharp/index.html

6.1.2 Aides reçues

- M. GARCHERY
- Mme MOTA
- Mme TERRIER

6.1.3 Remerciements

Je tiens à remercier les experts, M. CONRAD et M. BERNARD, qui ont montré de l'intérêt pour mon travail.

Je remercie aussi M. GARCHERY qui m'a suggéré l'utilisation du Leap Motion en tenant compte de mes envies pour ce travail. Il a su guider le développement de mon application et il m'a apporté de l'aide lorsque j'avais des doutes.

Merci aussi à Mme MOTA et Mme TERRIER qui m'ont permis de me débloquer dans une des premières parties du développement. Grâce à elles, j'ai pu mieux structurer mon code et faciliter les affichages des différentes parties.

6.1.4 Procédure d'installation

1. Pour faire fonctionner correctement le logiciel, il faut premièrement installer le SDK à partir du site Internet du Leap Motion (https://developer.leapmotion.com/get-started/).
2. Une fois l'installation terminée, connectez le Leap Motion à l'ordinateur. Une LED verte sur le côté du Leap Motion devrait témoigner de son bon fonctionnement.
3. Ouvrez le projet Visual Studio, cliquez sur le menu « Projet », puis tout en bas, cliquez sur « Propriétés de projet-pre-tpi ». Naviguez dans l'onglet « Événement de build » et dans le textbox intitulé « Ligne de commande de l'événement pré-build », ajoutez la ligne suivante :

   « *xcopy /yr "**\LeapSDK\lib\x64\LeapC.dll" "$(TargetDir)"* »

   Remplacez les deux astérisques par le chemin d'accès de la librairie LeapC.dll téléchargée avec le SDK du Leap Motion.

4. Si ce n'est pas déjà fait, ajoutez la référence « LeapCSharp.NET4.5 » aussi disponible dans le dossier du SDK.

Une fois cela effectué, le programme devrait fonctionner normalement.

6.1.5 Tables des illustrations