**Etudiant** : Florian ABRY

**Entreprise** : Wotan Systems
**Maître de stage** : Aymeric MORILLEAU
**Enseignant tuteur** : Guy JUILLARD

# RAPPORT DE STAGE

**Juin 2007**

Animer un mannequin 3 Dimensions reproduisant en temps réel les mouvements réalisés par le système PROBEX à l'aide du logiciel Blender et du langage de programmation Python.


U.T.B – UCB LYON 1, Département Génie Electrique et Informatique Industrielle

## Sujet

Animer un mannequin 3 Dimensions reproduisant en temps réel les mouvements réalisés par le système PROBEX[1] à l'aide du logiciel Blender et du langage de programmation Python.

## Conditions du stage

Stage de fin d'étude du DUT Génie Electrique et Informatique Industrielle (GEII)
Du 03/04/2007 au 29/07/2007 (13 semaines)
Rapport à remettre avant le 14/07/2007. Soutenance le 19/07/2007

## Stagiaire

**NOM** : ABRY
**PRENOM** : Florian
**DIPLOME EN COURS** : DUT GEII

## Maitre de stage dans l'entreprise

**NOM** : MORILLEAU
**PRENOM** : Aymeric
**POSTE** : Ingénieur Recherche et Développement chargé de la partie électronique, électrique et informatique

## Enseignant tuteur

**NOM** : JUILLARD
**PRENOM** : Guy
DATE DE LA VISITE DANS L'ENTREPRISE : 22 mai 2007

## Entreprise

**NOM** : WOTAN SYSTEMS
**STATUT** : SAS (Société par Actions Simplifiées)
**ADRESSE** : CEI 1, 66 Boulevard Niels BOHR, BP 2132
69603 Villeurbanne Cedex
**TELEPHONE** : 04 37 48 84 39
**FAX** : 04 78 94 15 49

## Université

Département GEII – IUT B- Université Claude Bernard Lyon 1
17 rue de France 69627 Villeurbanne Cedex

[1] **PR**océdé d'**O**rthèse de **B**ras **EX**osquelettique.

# REMERCIEMENTS

Je tiens à remercier tout le personnel de Wotan Systems pour m'avoir intégré au sein de leur équipe de travail. Ainsi, j'ai pu découvrir le monde de l'entreprise et, plus précisément, celui d'un bureau d'étude.

Je remercie plus particulièrement Mr Aymeric MORILLEAU qui m'a permis de découvrir son métier et qui m'a proposé un sujet de stage particulièrement intéressant. Il m'a aussi présenté les différents domaines de compétences qu'un étudiant sortant d'un IUT GEII doit savoir maîtriser pour être efficace dans un bureau d'étude tel que celui où il exerce.

Je remercie également Mr Patrick SADOK, le directeur, pour m'avoir permis de réaliser mon stage au sein de son entreprise.

Enfin, je remercie les enseignants de l'IUT GEII pour m'avoir appris les notions utiles à mon stage. Cela m'a permis d'être apte à aborder les thèmes qui composaient cette expérience professionnelle.

# SOMMAIRE

# TABLE DES ILLUSTRATIONS

# RESUME

# ETUDE GÉNERALE

## INTRODUCTION GÉNERALE

L'IUT GEII est une formation pluridisciplinaire dont le diplôme est décerné après un stage technique de 10 semaines dans une entreprise. Le but de ce dernier est double :

Pour l'IUT, il permet de vérifier que les élèves sont capables de s'adapter aux différentes tâches qui pourront leur être demandées au sein d'une entreprise et qu'ils sont capable de restituer les connaissances qu'ils ont apprises pendant les deux années de leur formation.

Pour les élèves, il permet de se confronter au monde de l'entreprise et, ainsi, de confirmer ou d'infirmer leur projet professionnel ainsi que leur poursuite d'étude tout en acquérant une expérience professionnelle. C'est principalement dans l'optique de validation d'orientation que j'ai abordé ce stage. En effet, l'Informatique Industrielle, qui tient une part importante dans le sujet de mon stage, est un domaine qui m'intéresse particulièrement et le Bureau d'Etude un espace de travail qui m'intéresse également.

Ainsi, Wotan Systems est un bureau d'étude et de développement situé dans le Centre des Entreprises Innovantes de Villeurbanne. Il est composé d'une équipe d'électroniciens, programmeurs et mécaniciens et s'est spécialisé sur la technologie des Exosquelettes[2] en se positionnant comme l'entreprise Française à l'avant-garde de cette technologie. La taille modeste de ce Bureau d'Etude permet, outre une plus grande réactivité aux besoins des clients, d'immerger totalement les stagiaires dans son fonctionnement. Une entreprise de cette taille présente aussi pour avantage de confronter à des domaines qui n'auraient pas été abordés autrement tels que la mécanique ou la CAO[3].

Le présent rapport ne se veut pas pour autant une liste exhaustive de mes activités au jour le jour, mais plutôt, après une brève présentation de l'entreprise et de ses réalisations, un document rendant compte de la méthode mise en œuvre pour répondre à la problématique qui m'a été proposée.

[2] Caractéristique anatomique externe qui supporte ou protège un animal (par exemple un insecte). Des recherches scientifiques tentent actuellement de développer des exosquelettes biomécaniques pour des besoins médicaux, industriels ou militaires.

[3] **C**onception **A**ssistée par **O**rdinateur.

# PRÉSENTATION DU SUJET


**Figure 1 : Logo de l'entreprise**

## A) Histoire de l'entreprise :

Le projet de développement d'exosquelette commence en 2002, lorsqu'après la réalisation d'un premier prototype, il est intégré à l'incubateur scientifique Rhône-Alpes Ouest CREALYS. Il entre, peu de temps après, en phase de post-création. La même année, un premier partenariat est signé avec le laboratoire de recherche LBMH[4] de l'Université Claude Bernard Lyon 1. De son côté, Patrick SADOK, initiateur et porteur du projet, suit le programme d'appui à la création d'entreprises de l'Ecole de Management d'Ecully afin de concrétiser un modèle économique cohérent.

C'est en juillet 2004 que le projet se concrétise en entreprise, peu de temps après avoir reçu le label NOVACITE[5]. Cette dernière s'installe sur le domaine de la DOUA à Villeurbanne, au cœur du campus scientifique lyonnais. Dans la même année, deux nouveaux partenariats sont signés avec des laboratoires de recherche publique : INSA-Lyon[6] et CPE-Lyon[7]. La société Wotan Systems devient bénéficiaire du Fond d'Innovation du Conseil général du Rhône.

Les premières démonstrations publiques ont lieu en 2005 devant la Société de Biomécanique Française. Ces démonstrations publiques dévoilaient un nouveau prototype d'exosquelette, le PROBEX 100. Celui-ci à pour vocation de remédier aux déficiences liées à des handicaps moteurs. C'est également en 2005 que le partenariat avec les établissements Lecante[8] est mis en place. Sa

[4] **L**aboratoire de **B**iomécanique et de **M**odélisation **H**umaine.
[5] Label décerné aux entreprises dites innovantes par la Chambre de Commerce et d'Industrie.
[6] **I**nstitut **N**ational des **S**ciences **A**ppliquées.
[7] Ecole supérieure de **C**himie, **P**hysique et **E**lectronique.
[8] Groupe leader dans l'orthopédie.

finalité est de mettre sur le marché des produits issus de la recherche et du développement au sein de Wotan Systems.

Depuis mai 2005, sur la base des prototypes précédents, a été développé une nouvelle orthèse[9] de bras nommée PROBEX 200. Elle se caractérise par de nombreuses évolutions du produit précédent, notamment en termes d'encombrement et de précision de la commande.

Actuellement l'entreprise travaille sur la validation clinique ainsi que le marquage CE[10] de cette orthèse de bras. Elle finalise aussi ce qui sera le PROBEX 201, la version commercialisable de l'exosquelette.

## B) Composition de l'entreprise

L'équipe de Wotan Systems est actuellement composée de 5 personnes :

- 1 Directeur Général
- 1 Directeur Administratif et Financier
- 1 Directeur Technique
- 2 Ingénieurs

**Patrick SADOK** (36 ans) : Directeur Général

A l'origine ingénieur en informatique, diplômé du Conservatoire National des Arts et Métiers (C.N.A.M.), Patrick SADOK est l'initiateur du projet. Il prend les décisions finales sur les modifications à réaliser sur le système. Il gère les employés, trouve des investisseurs et des subventions, et met en place la commercialisation future des produits.

**Jean-François ACHARD** (40 ans) : Directeur Administratif et Financier

Acteur important dans la validité économique du projet, il s'occupe de la gestion et de la comptabilité de l'entreprise.

[9] En opposition à une prothèse, l'orthèse ne vient pas remplacer le membre mais se place autour de ce dernier.
[10] Label certifiant le système aux normes de sécurités votées par la communauté européenne.

**Sébastien HUNTZBUCHLER** (26 ans) : Directeur Technique

Il s'occupe de la conception mécanique du projet et de la CAO. Il définit et réalise les plans de chaque pièce mécanique présente dans le système. Il gère également la coordination entre la mécanique et l'électronique.

**Nathalie BELLONE-DORIOT** (32 ans) : Ingénieure Recherche et Développement

Docteur en biomécanique, elle vérifie que PROBEX respecte la morphologie du corps humain, que le système soit en adéquation avec les différentes normes. Elle réalise également les simulations de contraintes (CAO) afin de valider la solidité des pièces mécaniques conçues. Pendant mon stage, elle n'était pas présente (en congés maternité).

**Aymeric MORILLEAU** (22 ans) : Ingénieur Recherche et Développement

Il réalise toute la partie programmation du projet, s'occupe également des parties électroniques et électriques embarquées. Il gère également la coordination mécanique, électronique et programmation.

En dehors de l'équipe de l'entreprise, durant le stage étaient aussi présents :

**Marie ALGORE** (26 ans) : Stagiaire SPIGBM[11] -sous la tutelle de Sébastien HUNTZBUCHLER-

Elle est chargée de réaliser la notice d'utilisation ainsi que l'organisation des validations cliniques en vue du marquage CE.

**Jérémie POCACHARD** (21 ans) : Stagiaire CPI[12] -sous la tutelle de Sébastien HUNTZBUCHLER-

Il participe à la conception et à l'amélioration du PROBEX 201.

**Yohan PEAUD** (22 ans) et **Vincent BARREAU** (20 ans) : Stagiaires GEII -sous la tutelle d'Aymeric MORILLEAU-

[11] **S**ciences **P**our l'**I**ngénieur **G**énie **B**io**M**édical.
[12] **C**onception de **P**roduits **I**ndustriels.

Ils développent et font de la maintenance sur la partie électrique du système.

Et moi-même, **Florian ABRY** (19 ans) : Stagiaire GEII -sous la tutelle d'Aymeric MORILLEAU-

Chargé de réaliser une interface 3D devant reproduire sur un mannequin virtuel les mouvements réalisés par le PROBEX.

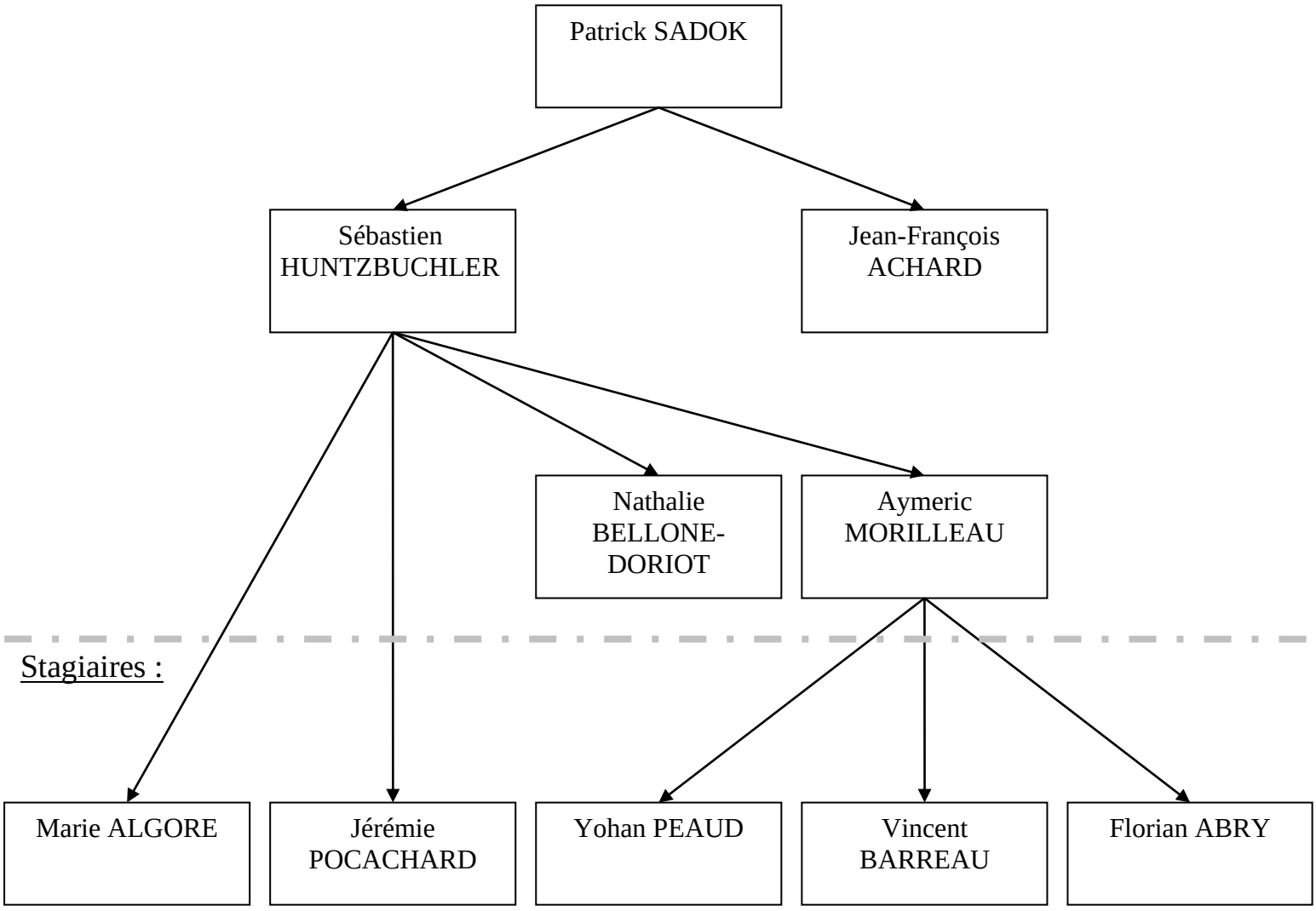



**Figure 2 : Organigramme de l'entreprise à juin 2007.**

## C) Evolution du PROBEX

**PROBEX 001 :**

Ce système est le premier prototype réalisé alors que Wotan Systems n'était pas encore une entreprise, en 2002. C'est une orthèse de membre supérieur qui a pour but d'amplifier la force d'un individu valide à l'aide de vérins « double effet ». En test, il a permis de soulever des blocs de 80kg à une main sans que le porteur n'ait à fournir d'efforts importants. Ce prototype a servi avant tout à la validation initiale des procédés mis en œuvre dans le projet.

**Figure 3 : Le Probex 001.**

**PROBEX 100 :**

Ce prototype date de mai 2005. Il diverge de son prédécesseur par plusieurs aspects. Tout d'abord, son objectif n'est pas la même : il ne s'agit plus d'amplifier la force d'un individu, mais de faire de la compensation musculaire pour un individu touché par un handicap moteur. Il diffère aussi par la position du patient qui, cette fois, est assis et par la technologie employée. Des paires de vérins « simple effet » viennent remplacés les « double effet ». Par ailleurs, la captation des velléités de mouvement est désormais proportionnelle. Ce prototype est révélateur de la volonté de l'entreprise de se tourner vers des systèmes de Compensation du Handicap.

**Figure 4 : Le Probex 100**

**PROBEX 200 :**

Datant de Décembre 2005, ce prototype se veut l'évolution du Probex 100. Les mouvements s'y font désormais par motorisation électrique. Il est aussi plus ergonomique, plus précis dans sa manipulation et moins encombrant. Il est principalement destiné aux validations cliniques.

**Figure 5 : le Probex 200**

**PROBEX 201 :**

Cette version actuellement en fin de développement est la version qui devrait être commercialisée. Il devra prendre en compte la rééducation des deux bras.

**Figure 6 : le Probex 201**

## D) La concurrence

Bien qu'étant à la pointe de la recherche Française sur la technologie exosquelette, Wotan Systems n'est pas le seul bureau d'étude à s'y intéresser. Ainsi, les japonais ont ouvert la voie avec leur exosquelette intégral d'amplification de force HAL-5 (voir Figure 7). Les américains de l'université de Berkeley les ont suivi de peu avec leur exosquelette de compensation de charge portée (voir Figure 8).

**Figure 7 : L'exosquelette HAL-5**

**Figure 8 : L'exosquelette de BERKLEY**

Plus récemment, d'autres bureaux d'études se sont penchés sur les orthèses exosquelettiques de membres supérieur dans un but rééducatif. On peut notamment citer le travail réalisé par l'équipe allemande de ARTROMOT ou les Suisses d'ARMIN.

**Figure 9 : L'exosquelette de ARTROMOT**

**Figure 10 : L'exosquelette ARMIN**

# ETUDE TECHNIQUE

## INTRODUCTION TECHNIQUE

Le PROBEX 201 va entrer en phase de production et ce robot se veut le plus polyvalent possible, visant aussi bien des applications dans des centres hospitaliers que chez des kinésithérapeutes libéraux où dans la recherche médicale. Cependant, le système présente quelques lacunes, notamment au niveau de la rééducation neurologique. Il a donc été décidé, en s'appuyant sur les travaux de plusieurs chercheurs, d'intégrer un module graphique optionnel au PROBEX dans le but de le placer à l'avant-garde de la neuro-rééducation. La solution retenue est d'ajouter une interface en 3D représentant un humanoïde s'animant pour reproduire en temps réel les mouvements réalisés par le patient dans le PROBEX, et c'est à moi qu'il est revenu de mettre cette solution en œuvre.

Dans un premier temps, nous verrons plus en détail les raisons qui ont amenées cette problématique à émerger, avant de nous intéresser à la démarche mise en œuvre pour maîtriser les outils nécessaires à sa réalisation. Pour finir, nous nous intéresserons à la mise en œuvre de la solution en elle-même.

# I) JUSTIFICATION DU SUJET

## A) Situation

La génération actuelle du PROBEX (PROBEX 200 et suivantes) s'inscrit dans une démarche de rééducation de personnes souffrant de déficiences musculaires, réalisant ainsi un travail analogue à celui d'un kinésithérapeute.

### *1) Les mouvements de base*

Pour remplir cet objectif, le système est en mesure de faire travailler le bras du patient sur quatre mouvements de base (ainsi que leurs composés) :

- **L'Abduction de l'épaule :**

**Figure 11 : L'Abduction**

- **La Flexion de l'épaule :**

**Figure 12 : La Flexion de l'épaule**

- **La Rotation axiale du bras :**

**Figure 13 : La Rotation axiale du bras**

- **La Flexion du coude :**

**Figure 14 : La Flexion du coude**

*2) Les modes de fonctionnements*

Il faut aussi savoir que le PROBEX a trois modes de fonctionnement :

- **Le mode saisie de données sur PC** : Le médecin programme la valeur de tous les angles par lesquels le système devra passer ainsi que la vitesse de réalisation des différents mouvements. Les mouvements ainsi programmés pourront être exécutés de manière séquentielle ou couplée.

- **Le mode temps réel** : Le médecin commande en temps réel les mouvements de l'orthèse à l'aide de boutons (tels que les figures 11 à 14) et définit des points de passage. L'exosquelette reproduira ensuite ces mouvements en allant de points de passage en points de passage. Ce mode de fonctionnement, bien que moins précis que le précédent, est beaucoup plus intuitif.

- **Le mode réplication** : Le médecin exécute directement le mouvement sur l'orthèse. Le mouvement sera enregistré et répliqué à la demande.

*3) Le fonctionnement du système*

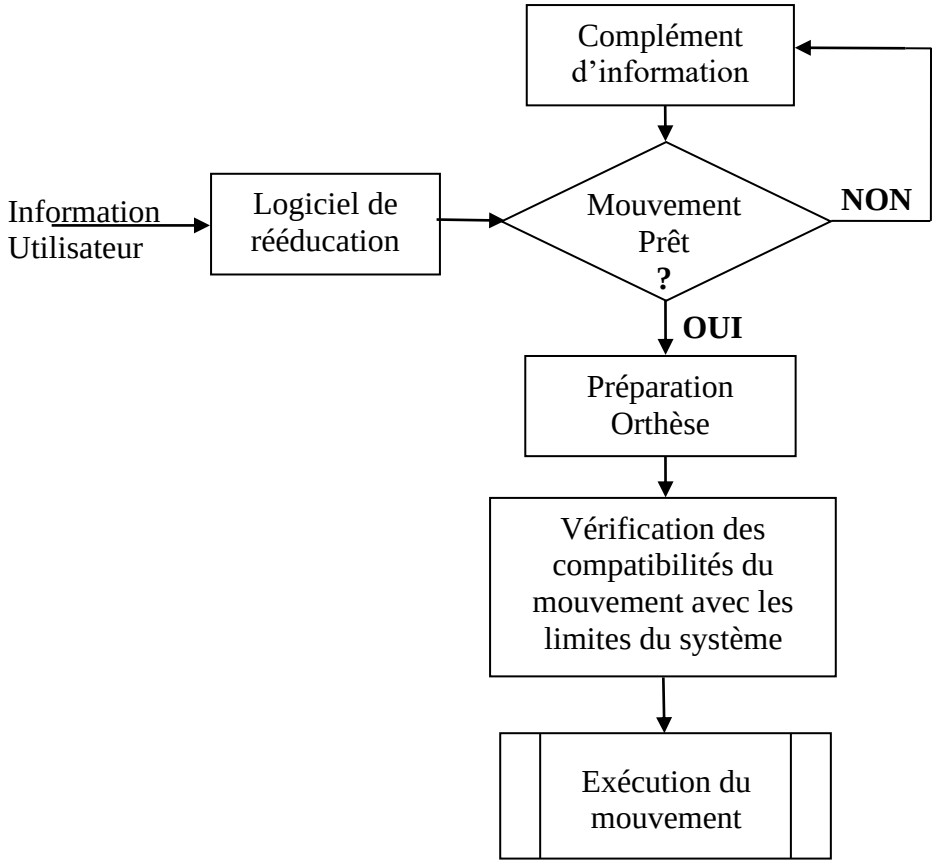



**Figure 15 : Synoptique de fonctionnement du système**

#### *4) Les limites du PROBEX 200*

Le système, tel qu'il est à l'heure actuel, a atteint un certain nombre de limites. Tout d'abord, au niveau de l'amplitude des mouvements, il est bien en dessous de ce que peut avoir à faire un humain. On peut citer en exemple la flexion de l'épaule dont la limite est à 90° (bras à l'horizontale, perpendiculaire au plan du torse) sur le système alors que de nombreux exercices de rééducation l'amène à 180° de rotation autour de l'axe de l'épaule.

L'autre limite du PROBEX est qu'il ne permet de rééduquer que les personnes ayant un problème moteur[13] uniquement causé par une déficience musculaire. Or, la cause d'une paralysie des membres supérieurs peut aussi allier cette déficience musculaire avec un problème neuronal. Les connexions entre le cerveau et les muscles de ce membre ne se faisant plus, une rééducation purement musculaire n'apportera qu'un résultat partiel.

## B) Problème posé

#### *1) But à atteindre*

Comme expliqué ci-dessus, le PROBEX est inadapté pour la rééducation de personnes dont les connexions neuronales sont endommagées. Cependant, des chercheurs ont publié des études[14] montrant que, par l'ajout d'une interface graphique adaptée, la robotique et la biomécanique pouvaient avoir un apport très bénéfique à la neuro-réhabilitation de patients souffrants de troubles neurologiques.

L'étude du Docteur en physique H.I.Krebs parue dans le *Journal of Rehabilitation Research and Devlopment* se base sur un système d'orthèse pilotant le bras du patient. Le principe consiste à ce qu'un thérapeute commande à distance les mouvements du bras robotisé. Ces mouvements sont reproduits sur le bras du patient qui visualise un *feedback* en temps réel du mouvement réalisé sur un écran. Les résultats de cette étude furent concluants et le docteur H.I.Krebs annonce, sans pour autant avancer de chiffres, que cette méthode de neuro-réhabilitation réduit le temps nécessaire à un patient pour recouvrir ses capacités motrices.

[13] Au niveau du muscle.

[14] Voir, par exemple, l'article "*Increasing productivity and quality of care : Robot-aided neuro-rehabilitation"* du Docteur en Physique H.I.Krebs parue dans le « ***Journal of Rehabilitation Research and Devlopment*** » Vol.37, No.6, Novembre/Decembre 2000, p.639-652

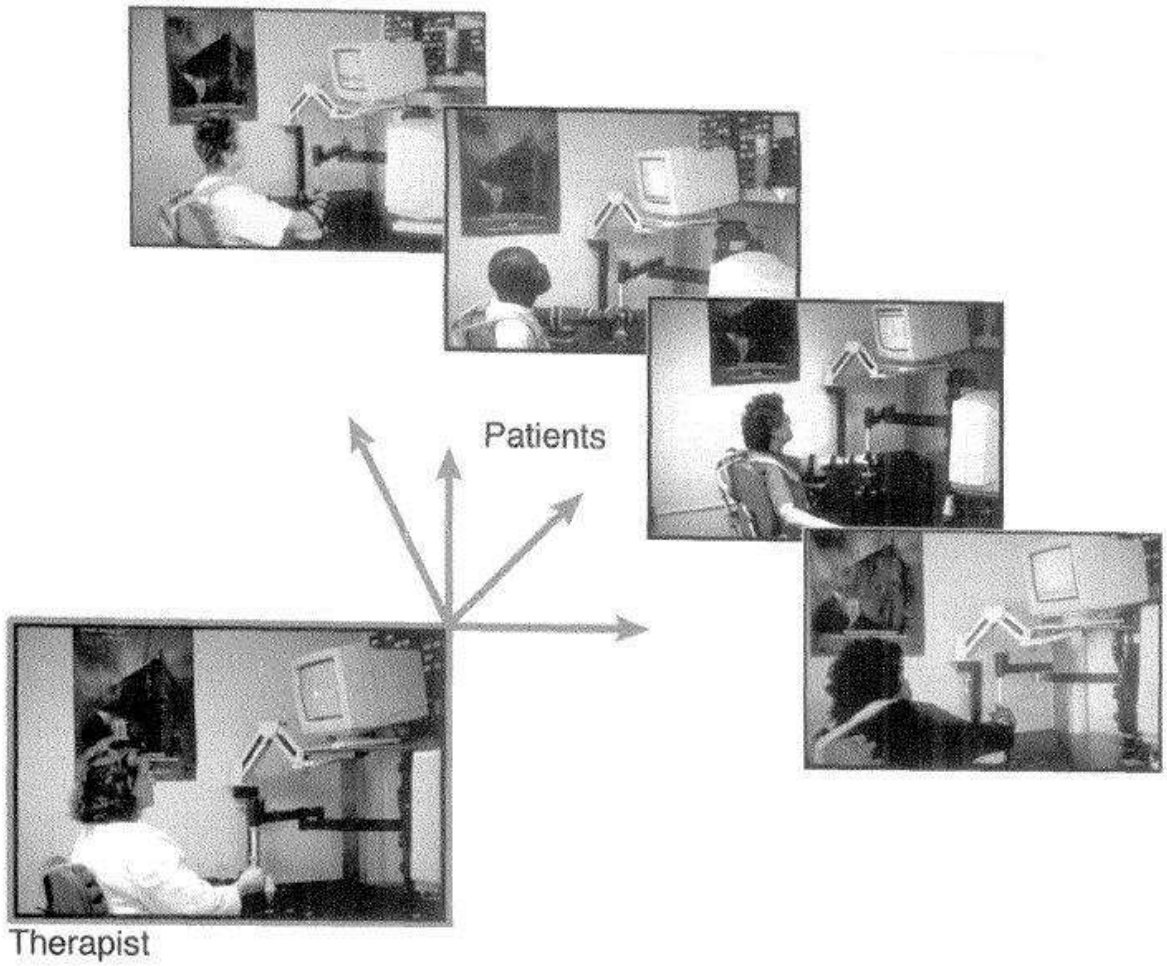



**Figure 16 : Principe de l'étude du docteur Kerbs.**

Partant de cette étude, la volonté de rendre le PROBEX plus polyvalent en laissant la possibilité d'accompagner la rééducation motrice par de la neuro-réhabilitation s'est développée. La solution retenue a été de modéliser un mannequin humain en 3 Dimensions reproduisant en temps réel les mouvements que le système fait réaliser au patient.
Le but de mon stage est donc de mettre en œuvre cette solution.

### *2) Cahier des charges*

#### a) Contraintes générales.

- Réaliser une interface graphique 3D en temps réel modélisant les mouvements du patient sur un mannequin numérique humanoïde.
- Le patient devra pouvoir s'identifier au mannequin 3D.

b) Contraintes logicielles.

- Le mannequin 3D devra être modélisé sous *Blender*[15], un logiciel de modélisation 3D qui, en plus d'être libre, est d'une performance extrêmement proche des logiciels professionnels.
- Le programme devra être développé en *Python*[16].
- Le programme en *Python* devra pouvoir communiquer avec le programme en *C++* pilotant le PROBEX dans le but de récupérer les informations d'angles.
- Le programme devra pouvoir s'exécuter automatiquement et être exécuté à distance.

c) Contrainte matérielle.

- Le programme devra être optimisé pour fonctionner sur un PC industriel embarqué (puissance de calcul limitée).

### *3) Difficultés du sujet*

Les principales difficultés du sujet sont venues des contraintes logicielles. En effet, je n'avais jamais programmé en *Python* ni fait de 3D (donc, par extension, jamais utilisé *Blender*). De plus, au sein de l'entreprise, personne n'avait les compétences pour m'épauler dans ces domaines. Par ailleurs, est venu s'ajouter à cela le fait que les interfaces 3D sont, d'une manière générale, très gourmandes en puissance de calcul. En développer une adaptable sur des ordinateurs de puissances moyenne était également un challenge, tout comme celui de faire communiquer à distance deux programmes développés dans deux langages différents.

Au final, ce sujet de stage se trouve être particulièrement motivant car ambitieux et intégré dans le processus de développement d'un système encore à l'avant-garde des technologies actuelles.

[15] Disponible sur www.blender.org
[16] Langage de programmation gratuit et portable. Les classes du logiciel *Blender* ne sont accessibles que dans ce langage. Il est disponible sur www.python.org

# II) PREPARATION A LA REALISATION DU SUJET

Une fois ce premier contact avec mon sujet de stage pris et après avoir dégagé les principales difficultés que je risquais de rencontrer, il m'a fallu prévoir la manière dont j'allais traiter le sujet. Je me suis donc fixé des dates butoirs auxquelles différents points du cahier des chargent devaient être traités.

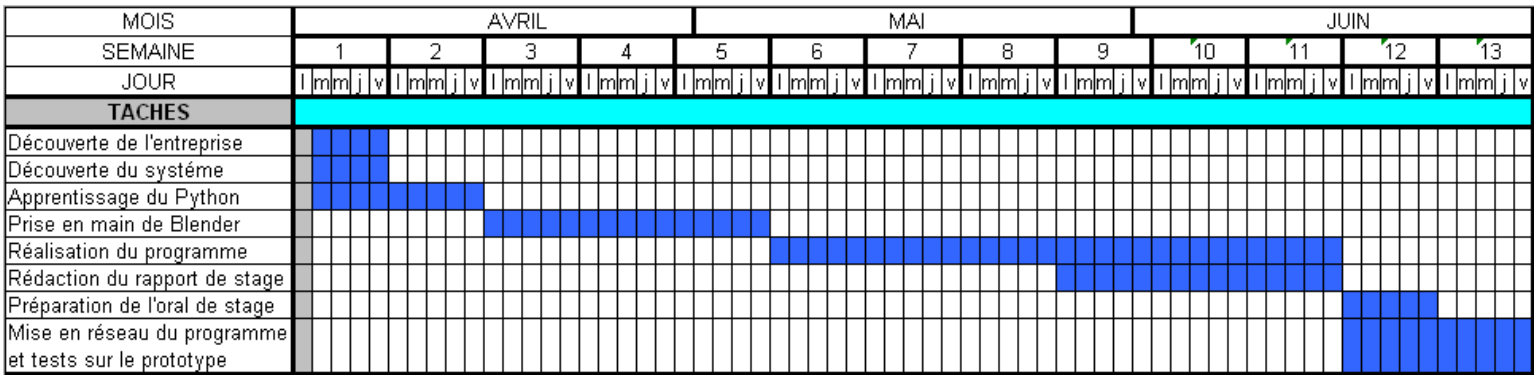



**Figure 17 : Planning prévisionnel du stage.**

De manière à garder des traces du travail réalisé, des feuilles d'heures sont réalisées à la fin de chaque journée (voir **Annexe 1** p.41).

## A) L'Apprentissage du Python

### 1) *Qu'est-ce que le python ?*

Le python, en plus d'être un serpent, est un langage informatique. Ce dernier a été créé en 1990 par l'informaticien Hollandais Guido van Rossum. Il présente plusieurs intérêts. En plus d'être libre, sous une licence proche de la licence BDS[17], il est compatible sur la quasi-intégralité des plateformes informatiques (que ce soit les supercalculateurs, les ordinateurs centraux, ceux utilisant Unix, Linux, Windows, MacOS, Java ou .NET pour les plateformes les plus connues). C'est un langage orienté objet et interprété[18] dont l'interpréteur est disponible sur www.python.org . Il se distingue du C (langage appris pendant la formation à l'IUT) par certains points.

[17] **B**erkeley **S**oftware **D**istribution. C'est une licence libre utilisée pour la distribution de logiciels. Elle permet de réutiliser tout ou partie du logiciel sans restriction, qu'il soit intégré dans un logiciel libre ou propriétaire.
[18] Contrairement aux langages Compilés qui compilent une fois pour toute les programmes en exécutables, les programmes programmés en langages interprétés sont traduits en instructions lisibles à chaque lancements.

### 2) *Les divergences entre le python et le C*

Pour bien se rendre compte des divergences entre le C et le python, je vais faire un parallèle entre un programme simple programmé en python et un programme simple programmé en C. Prenons en exemple un programme calculant le périmètre de cercles dont les rayons sont compris entre 1 et 15 :

| n° | Programme C | Programme Python |
|---|---|---|
| 1 | `#include <math.h>` | `from math import *` |
| 2 | | |
| 3 | `float perimetre;` | |
| 4 | `int rayon;` | |
| 5 | `float f_perimetre(int r);` | |
| 6 | | |
| 7 | `//----------------------------------` | `#----------------------------------` |
| 8 | | |
| 9 | `float f_perimetre (int r)` | `def f_perimetre (r):` |
| 10 | `{` | |
| 11 | `float p;` | |
| 12 | `p = 2*M_PI*r;` | `p = 2*pi*r` |
| 13 | `return p;` | `return p` |
| 14 | `}` | |
| 15 | | |
| 16 | `main()` | |
| 17 | `{` | |
| 18 | `rayon = 1;` | `rayon = 1` |
| 19 | `while (rayon<=14)` | `while rayon <= 14:` |
| 20 | `{` | |
| 21 | `perimetre = f_perimetre(rayon);` | `perimetre = f_perimetre(rayon)` |
| 22 | `printf("%d => %f.\n", rayon, perimetre);` | `print rayon, "=>", perimetre` |
| 23 | `rayon++;` | `rayon+=1` |
| 24 | `}` | |
| 25 | `getch();` | `input()` |
| 26 | `}` | `}` |

**Figure 18 : Divergences entre le C et le python.**

Ce tableau met en exergue plusieurs choses, avec en premier, une différence au niveau de la lourdeur du code. Parmi les raisons qui font que le code en python à l'air plus léger, on peut citer le fait que les variables n'ont pas besoin d'être déclarées. La déclaration se fait automatiquement en fonction des données entrant dans la variable. Ainsi, une variable pourra successivement contenir un entier, un caractère, un flottant et une chaîne de caractère que le programme ne plantera pas.
Une autre raison vient du fait que les fins de ligne sont définies par un retour à la ligne plutôt que par un point virgule et que l'intérieur des boucles est défini par une indentation[19] plutôt que par des crochets. Rien n'empêche d'indenter en C, mais cela ne définira jamais l'intérieur d'une boucle. Les autres divergences sont plus au niveau des instructions (de leur nom ou de leur fonctionnement). Cependant, d'une manière générale, la logique dans laquelle s'inscrivent ces deux langages est la même.

[19] Bloc décalé par des tabulations.

### 3) *L'apprentissage du python*

Initialement prévue sur 2 semaines, la découverte du langage Python aura duré 3 jours au final. Deux facteurs ont fait que cet apprentissage a duré moins longtemps que prévu :
Les convergences avec le C au niveau de l'architecture des programmes fait que seule la nouvelle manière d'écrire les instructions est à assimiler. Le reste est extrêmement semblable au C.
L'autre raison est le tutoriel sur lequel je me suis appuyé pour cet apprentissage. Il s'agit d'un cours écrit par Gérard SWINNEN[20] construit et formulé de manière simple, rendant l'approche du langage intuitive même pour un néophyte. La seule grosse difficulté rencontrée pendant cet apprentissage a été d'appréhender l'aspect « orienté objet » du python.

### 4) *Le python, un langage orienté objet*

Un langage orienté objet est un langage dans lequel il est possible de définir, en plus des variables, des objets qui auront un comportement particulier. Un objet informatique est composé d'une classe comportant des méthodes (sous fonctions) et des attributs (variables). Par exemple, un objet de classe voiture peut être défini par des attributs comme sa marque ou sa couleur et par des méthodes comme tourner ou freiner. Pour mieux comprendre cette nouvelle manière d'aborder la programmation, j'ai réalisé un exercice présent dans le tutorial précédemment cité. Le but de cet exercice était de réaliser un programme comportant un objet représentant un jeu de cartes. Ces cartes devaient être battues et tirées une à une. Le programme commenté se trouve en **Annexe 2** p.42. D'un point de vue programmation en Python, les classes, comme les fonctions, se déclarent en amont du programme principal, juste après l'importation des bibliothèques, par l'instruction *class NomDeLaClasse :*. A l'intérieur de cette classe, on ajoute les éléments la composant (dans notre cas une liste de deux variable, l'une contenant un nombre compris entre 0 et 12 (pour 13 valeurs que peut avoir une carte) et l'autre entre 0 et 3 (pour les 4 couleurs). Toujours à l'intérieur de cette classe, on définit les méthodes et les attributs par l'instruction *def* de la même manière que l'on définirait une fonction. La seule différence est qu'elle doit contenir comme paramètre *self* qui indique qu'elle est appelée à partir d'un objet. Syntaxiquement, l'appel d'un module pas un objet s'écrit *Objet.Module(éventuellement_paramétres)*. Il est bien entendu possible que des modules soient eux même composés de module.

[20] Professeur de sciences et d'informatique à l'université de Liége. Son tutorial est disponible à l'adresse http://python.developpez.com/cours/TutoSwinnen/

Suite à cette première approche de ce langage, il m'a fallu découvrir le logiciel Blender. Il est en effet la raison pour laquelle j'ai du apprendre le Python.

## B) La découverte de Blender

### *1) Qu'est-ce que Blender ?*

Blender est un logiciel libre de modélisation et d'animation 3D. Il est sous licence GPL[21] et sa première version libre sort en 2003. Il a été utilisé dans plusieurs films dont Spider Man 2 pour la création des animations. Son interface (voir **Annexe 3** p.43) développée en OpenGL a la réputation d'un apprentissage long et difficile. En effet, celle-ci est intégralement modifiable et combine un nombre important de menus avec des raccourcis claviers en grand nombre. Une autre particularité de ce logiciel est son poids minime (seulement 30Mo), sa portabilité (il est compatible, entre autres, avec Windows, BeOS, Linux, MacOS, FreeBSD), son moteur de jeu intégré (GameBlender) et la possibilité de lui intégrer des scripts Python. C'est notamment pour cette dernière particularité qu'il a été retenu. En effet, l'intégration de scripts se fait directement dans l'interface du logiciel, ce qui facilite grandement leur création.

**Figure 19 : Exemple de modélisation sous Blender**

### *2) La découverte de la modélisation sous Blender*

Bien que la tâche qui m'a été demandée relève de la programmation plus que de la modélisation, la connaissance du fonctionnement de cette dernière peut s'avérer utile. En effet, la programmation de scripts en Python pour Blender revient à automatiser des actions faites « à la main ». Je me suis donc appuyé sur des tutoriaux, notamment celui de J.M.Soler[22] et sur l'aide de communautés

[21] **G**eneral **P**ublic **L**icence. C'est une licence qui fixe les conditions générales de distribution des logiciels libres.
[22] http://jmsoler.free.fr

francophones telles que le Blender Clan[23]. Lorsque j'ai pensé avoir assimilé les bases, je me suis fixé comme objectif de modéliser le logo de l'entreprise en 3D. Le résultat est visible en **Annexe 4** p.44. Une fois les bases de la modélisation digérées, j'ai dû m'attaquer au développement de scripts Python sous Blender.

## C) Le développement Python sous Blender

### *1) Premier contact avec la bibliothèque Blender pour Python*

Si les bases du langage python ont déjà été abordées, son application au sein de Blender n'est pas non plus sans difficultés. En effet, l'utilisation de Python dans Blender n'est pas encore généralisé et très peu de tutoriaux existent sur ce sujet. Les rares tutoriaux existants s'arrêtent à la modélisation de base, la création de caméra, de boutons ou de textures. J'ai donc choisi de les réaliser et de les améliorer dans le but de me familiariser avec l'éditeur de Texte de Blender. Suite à cela, je ne me sentais pas encore en mesure d'attaquer le programme d'animation d'humanoïde. J'ai donc modélisé, par programmation, un semblant d'humanoïde à angles droits. Mon but était d'avoir une fenêtre dans laquelle on pourrait entrer, en degrés, la valeur des angles de flexion du coude, d'abduction, de flexion de l'épaule de  rotation axiale et qu'après validation le mannequin s'anime pour arriver dans la position voulue.

**Figure 20 : Le « mannequin à angles droits » à animer.**

### *2) Premier programme d'animation : le Mannequin à angles droits*

Ce programme (situé en **Annexe 5** p.45) est un peu l'aboutissement de l'apprentissage mené jusqu'à sa réalisation puisqu'il reprend essentiellement des fonctionnements vus dans des tutoriaux.

#### a) Les Bibliothèques.

Comme tous les programmes utilisant Blender, il faut commencer par importer la bibliothèque du même nom (ligne *1*). Pour éviter de surcharger le programme, seules quelques classes de cette bibliothèque seront importées. Il faut savoir que cette bibliothèque n'est importable que si le script est exécuté depuis Blender. Les autres bibliothèques importées sont *Math*, notamment pour les

[23] http://blenderclan.tuxfamily.org

opérateurs trigonométriques, et *Tkinter* pour afficher des fenêtres avec boutons et des champs d'entrées de données.

b) Le Programme Principal

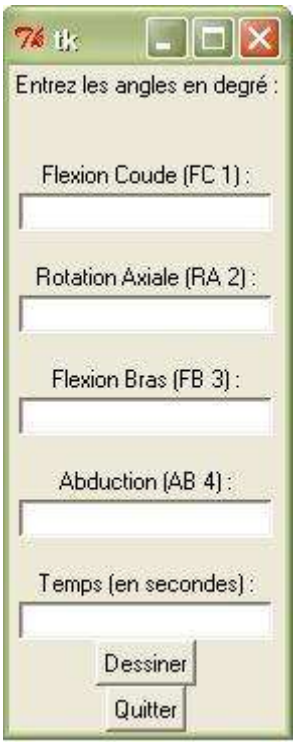



**Figure 21 : La fenêtre.**

Nous allons passer directement de la ligne *6* à la ligne *574*. En effet, pour bien comprendre la fonction présente entre les deux lignes précédemment citées, la connaissance du programme principale est indispensable. Celui-ci débute par l'initialisation de plusieurs variables. Bien que celles-ci n'aient pas besoin d'être déclarées, elles n'ont pas de valeur par défaut. Ces initialisations sont entre les lignes *574* et *582*. Puis vient la programmation de la fenêtre Tkinter. L'idée est de définir une variable comme un objet de type fenêtre (ligne *584*) puis d'ajouter des éléments à cette variable. Ainsi, la ligne *585* crée une zone de texte rattachée à notre fenêtre (nommée *fen1*). D'une manière générale, un objet de type « fenêtre » est composé de plusieurs objets qui lui sont reliés. Dans notre cas ces objets sont de type *Label* (zone de texte), *Entry* (champ d'entrée de donnée) et *Button* (bouton). Leur disposition géographique dans fenêtre se fait grâce au module *pack()*. Lorsqu'on applique ce module à un objet, il se positionnera juste en dessous de l'élément le plus récent ajouté à la fenêtre. Cependant, on peut ajouter un argument à *pack()* pour forcer un objet à aller à un emplacement voulu (voir lignes *586* et *611*). Ces arguments sont *TOP* (pour positionner l'objet le plus haut possible dans la fenêtre), *BOTTOM* (pour un positionnement en bas de la fenêtre), *LEFT* (pour un positionnement à gauche) et *RIGHT* (pour un positionnement à droite). Une fois le positionnement effectué, il faut demander l'affichage de la fenêtre (ligne *612*) par le module *mainloop()*. Cependant, notre fenêtre est composée de boutons et ceux-ci ont un argument supplémentaire à leur création qui permet de définir l'action se passant lorsque l'utilisateur clique dessus. Ainsi l'appui sur 'Quitter' provoque la fermeture de la fenêtre (*command=fen1.destroy* ligne *610)* et l'appui sur 'Dessiner' (ligne *608*) provoque l'exécution de la fonction *dessin*.

c) La fonction *dessin*

La fonction *dessin* est déclarée à la ligne *9* comme une fonction ne nécessitant pas d'argument. Ensuite (lignes *9-10*), on définit les variables utilisées dans le sous-programme comme « globales » ce qui signifient qu'elles ne sont pas remises à 0 à chaque lancement et qu'elles peuvent être utilisées aussi bien dans ce sous-programme que dans le programme principal. La suite consiste à

récupérer les valeurs entrées dans les champs d'entrées de données et à les traiter de manière a avoir un mouvement de bras tenant compte de la position initiale de ce dernier (lignes *11* à 20). La boucle *while()* sert à donner une impression de mouvement, faisant s'approcher le mannequin de sa position finale un peu plus à chaque itération (lignes *24* à *27*). Le programme présent de la ligne *33* à la ligne *331* est celui programmant la position des points de constructions de notre mannequin. Il représente mes débuts en programmation Python dans Blender. Comme c'est très souvent le cas avec les classes Blender, les instructions sont construites en imbriquant des modules les uns dans les autres.

Pour construire ce mannequin, je dessine un certain nombre de points sensés être l'armature de mon mannequin (lignes *33* à *59*). Ces points se construisent via l'instruction *NMesh.Vert(*coordonnées en x, coordonnées en y, coordonnées en z*)*. Le résultat de cette opération est mis dans un variable (ici *v*) et le contenu de celle-ci, ajouté à la liste de construction *me.verts* . Le fonctionnement du programme est basé sur les coordonnées de 2 points (cf Figure 24) qui seront appelés point A et point B. Il m'a donc fallu définir les coordonnées de ces deux points à partir des angles de Flexion Coude (FC dans l'équation), Rotation Axiale(RA), Abduction (AB), Flexion Bras (FB), de la distance entre l'épaule et le point A (DEA), de la distance entre le point A et le point B (DAB) et des coordonnées du point représentant l'épaule (PEx, PEy et PEz).

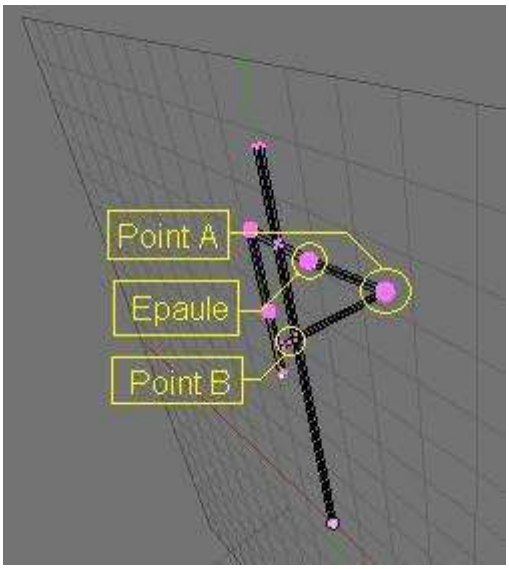



**Figure 22 : Point A et Point B**

Pour le point A, l'équation est :

En x : `PAx = PEx+DEA*sin(AB)*cos(FB)`

En y : `PAy = PEy-DEA*cos(AB)*cos(FB)`

En z : `PAz = PEz+DEA*sin(FB)`

Pour le point B, l'équation est :

En x : `PBx = PAx+PAB*sin(AB+RA)*cos(FB+FC)`

En y : `PBy = PAy-PAB*cos(AB+RA)*cos(FB+FC)`

En z : `PBz = PAz+PAB*sin(FB+FC)`

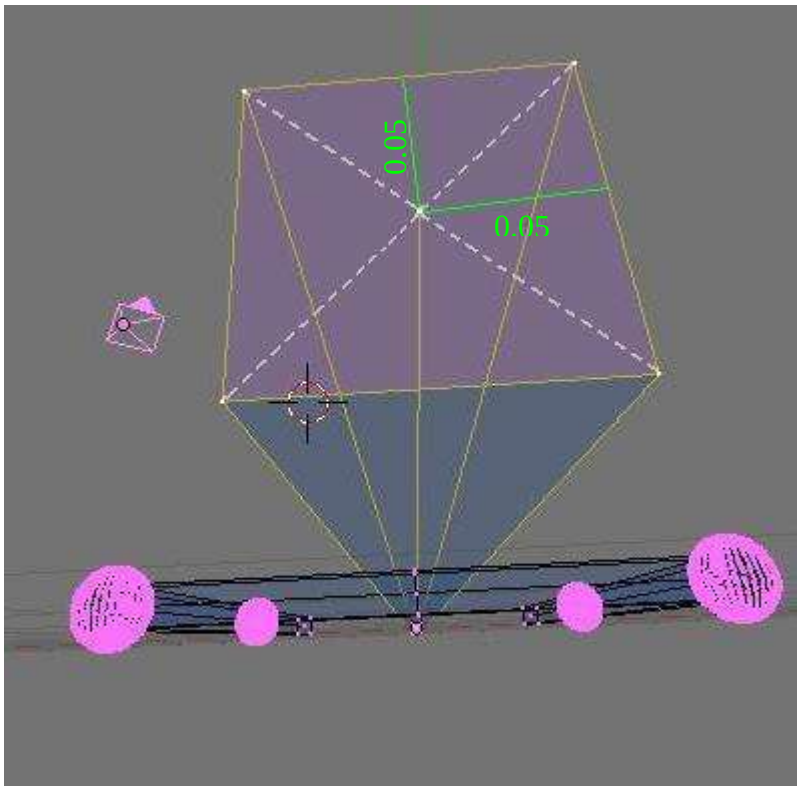



**Figure 23 : La construction du volume.**

Une fois ces points rentrés, il ne me reste plus qu'à donner du volume au tout et à afficher. Je modélise donc 4 points autour de chaque point précédemment construits (lignes *60* à *241*). Ces points sont à une distance de 0.05 (sans unités) sur chaque axe du plan du point central. Ensuite, je définis les points des sphères (lignes *243* à *331*) et relie tous ces points les uns aux autres, 4 par 4 pour les pavés et 3 par trois pour les sphères (lignes *341* à *568*), afin de formés des plans appelés « maillages » en 3D (*Mesh* en anglais). Enfin, ligne *569* et *570*, on relie tous ces maillages ensemble en une seule figure nommée 'Cube' que l'on affiche à l'écran via la commande *Blender.Redraw()*.

Fort de cette expérience concluante, je m'attèle à la réalisation du sujet de mon stage.

# III) REALISATION DU SUJET

## A) Le choix du mannequin avec lequel développer le programme

Le but du sujet qui m'a été proposé est de réaliser une interface reproduisant en temps réel les mouvements réalisés par le PROBEX. Cette interface doit contenir un mannequin humanoïde et une « progress-bar ». Or, le mannequin réalisé précédemment ne répond pas au critère d'identification du patient.

### *1) Ludwig*

**Figure 24 : Ludwig**

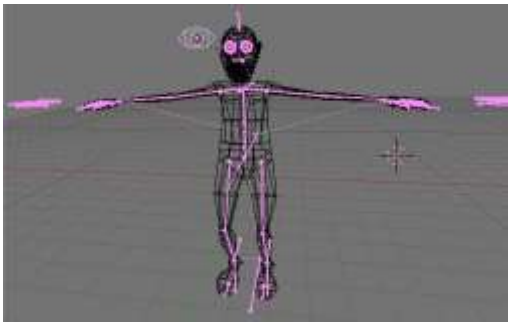


**Figure 25 : L'armature de Ludwig**

Pour commencer ce programme, deux mannequins m'ont étés proposés : Ludwig, un extraterrestre avec une armature[24] déjà construite et une amazone sans armature. J'ai donc essayé, dans un premier temps, d'utiliser Ludwig afin de bénéficier de son armature déjà construite. Je n'ai cependant trouvé aucun tutorial sur internet expliquant comment gérer une armature en python. Seule une personne à pu m'aider sur ce sujet en m'indiquant comment importer une armature en Python et comment en isoler les *Bones* pour travailler dessus. Malheureusement, l'armature de Ludwig est beaucoup trop complexe pour être manipulées via un logiciel. En effet, elle est optimisée pour de l'animation 3D « à la main » et la position de ses *Bones* est contrainte par la position de « *Bones* d'animation » appelés *Ik*[25]. Ces derniers sont conçu pour un rendu intuitifs, mais la position du jeu de *Bones* en fonction de celle de l'*Ik* qui leur est associé est extrêmement dur à mettre en équation. Il m'a donc fallu utiliser l'autre mannequin.

[24] En animation, on appelle armature l'ensemble des objets (appelés Bones) permettant d'animer un mannequin. Ils jouent le même rôle que les os chez un humain.
[25] Un *Ik* est un *Bone* qui n'anime pas un groupe de maillages, mais un groupe de Bone.

*2) L'amazone*

a) Le choix de l'amazone

**Figure 26 : L'Amazone**

Contrairement à Ludwig, l'amazone présente l'avantage d'être plus proche de ce à quoi ressemble un humain, même si son aspect trop « héroïc fantasy » l'empêche d'être utilisable dans un cadre rééducatif. De plus, son maillage beaucoup plus complexe que celui de Ludwig, nécessite beaucoup plus de ressources informatiques pour être animé. Enfin, comme dit plus haut, la particularité de ce mannequin est qu'il n'avait pas d'armature. Une des grosses difficultés du travail fourni avec l'amazone à donc été de « l'armaturer ». Le parti pris lors de cet « armaturage » à été de ne définir des *Bones* que sur les membres susceptibles de bouger. Ainsi, seuls l'avant bras, le bras et la main sont liés à un *Bone*.

b) La création des *Bones*

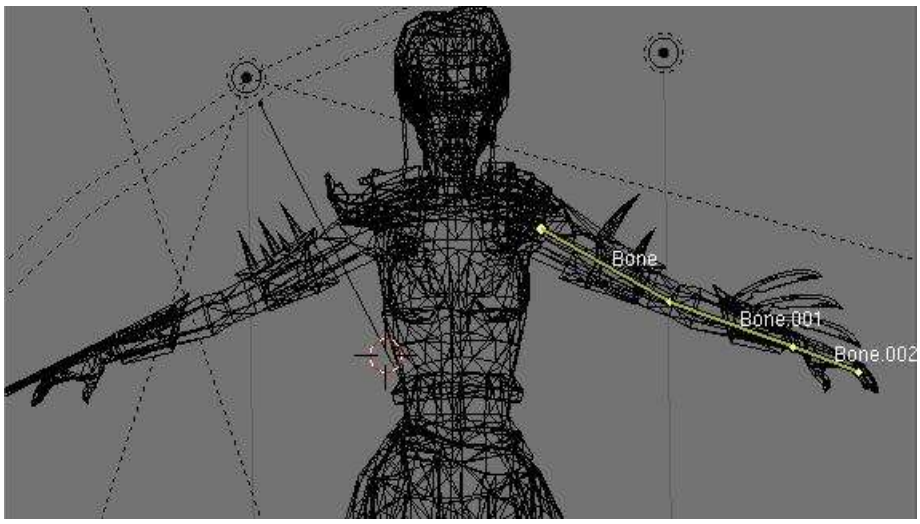



**Figure 27 : L'armature de l'amazone**

La phase de création de *Bones* est une phase longue est fastidieuse. Elle consiste à créer une armature et, dans un premier temps, à lui ajouter des *Bones* parents[26] les uns aux autres. Par exemple, dans celui de la Figure 29, le *Bone* nommé 'Bone.002' est enfant de celui nommé 'Bone.001', lui-même enfant de celui nommé 'Bone'. Cela se justifie par le fait que, chez l'humain, l'os du bras (l'humérus) entraîne ceux de l'avant bras et de la main lorsqu'il bouge. Une fois l'armature créée, il faut désigner le groupe de mailles que dirige chaque *Bone*. Cela se fait souvent maille par maille pour éviter de prendre un groupe de maille inapproprié qui aurait pour effet de déformer totalement le mannequin lors de son animation.

[26] Un *Bone* peut être parent et/ou enfant. Un *Bone* enfant sera entraîné par le mouvement de son parent. Cependant, contrairement aux *Bones* contraints à un *Ik*, un *Bone* peut bouger alors que son parent est immobile.

## B) Le programme

Une fois l'armature du mannequin réalisée, il a fallu commencer à coder ce qui sera le programme le programme répondant à la problématique de mon stage. L'intégralité de ce code est présent en **Annexe 6** p.55. La premiére partie de mon travail à donc été de manipuler les *Bones* fraîchement créés via un script Python.

### *1) L'animation des Bones*

#### a) La démarche

Comme expliqué précédemment, un des problèmes auquel j'ai été confronté était le manque de tutoriels sur ce sujet. Visiblement, la manipulation de *Bone* en python n'est pas chose courante. Cela s'explique aisément par le fait que ces *Bones* sont avant tout utile pour simplifier l'animation à la souris et l'animations par *Path*[27]. Toujours est-il qu'a force de recherche et de questionnements, notamment sur des Forums, j'ai été redirigé vers un jeu d'instruction détaillé en ligne[28]. Ce jeu d'instruction comprend un certain nombre d'exemples pour des classes de base ainsi que les détails d'utilisations des modules associés à ces classes. J'ai donc essayé de faire fonctionner les modules me paraissant les plus appropriés pour réaliser le travail qui m'était demandé. Cette phase fut assez longue car, dans la majorité des cas, le jeu d'inscription sus-cité n'indiquait que le type d'argument nécessaire au fonctionnement du module et le ce qu'il était sensé retourner. C'est justement parceque ces données étaient disponibles que j'ai pu voir ce qui allait être une des grosses difficultés de ce travail : les *Bones* ne sont pas animés, comme je l'imaginais, par le modèle mathématique d'Euler[29] ; mais par celui des Quaternions.

#### b) Les Quaternions

| · | 1 | i | j | k |
|---|---|---|---|---|
| 1 | 1 | i | j | k |
| i | i | -1 | k | -j |
| j | j | -k | -1 | i |
| k | k | j | -i | -1 |

**Figure 28 : Table de multiplication des quaternions.**

Un Quaternion est un nombre hypercomplexe[30] à quatre dimensions : une dimension réelle et trois dimensions complexes. Un quaternion s'écrit :

$$H = a \cdot 1 + b \cdot i + c \cdot j + d \cdot k$$

(Où a, b, c et d sont des nombres réels). La multiplication de quaternions entre eux est assez semblable à celle des

[27] *Path* signifie Chemin en anglais. En animation 3D, un *Path* est une trajectoire sur laquelle se déplace un objet ou un *Bone*. Ils sont très utiles pour les animations « répétitives » (tels que la modélisation d'un humain qui marche ou celle d'un grand huit sur des montagnes russes).

[28] http://www.blender.org/documentation/242PythonDoc/

[29] Ce modèle décompose les mouvements en composés de rotations sur 3 axes (x, y et z).

[30] L'ensemble des nombres Hypercomplexes (noté $\mathbb{H}$) est l'ensemble mathématique dans lequel celui ces Ccomplexes.

complexes dans le sens où, élevés au carré, ils peuvent donner un résultat négatif (cf. Figure 28 : Table de multiplication des quaternions). Ce modèle mathématique à plusieurs utilisations, dont l'animation 3D. En effet, il présente un intérêt non négligeable par rapport à Euler : La rotation qu'il modélise n'est pas une somme de rotation. Cela évite ce que l'on appelle le « phénomène de repliement d'angles ». Ce phénomène est propre à la modélisation d'angles par la méthode d'Euler. Il fait correspondre plusieurs vecteurs à un vecteur associé à un angle d'Euler. Cela n'a peut être pas l'aire évident formulé de la sorte, prenons donc un exemple : Soit un vecteur confondu avec l'axe X d'un plan. On demande à ce vecteur d'effectuer la rotation d'Euler définie par une rotation de -20° sur l'axe X, 90° sur l'axe Y  et 20° sur l'axe Z. Les rotations d'Euler étant séquentielles, l'ordre dans lesquelles elles sont effectuées est très important.

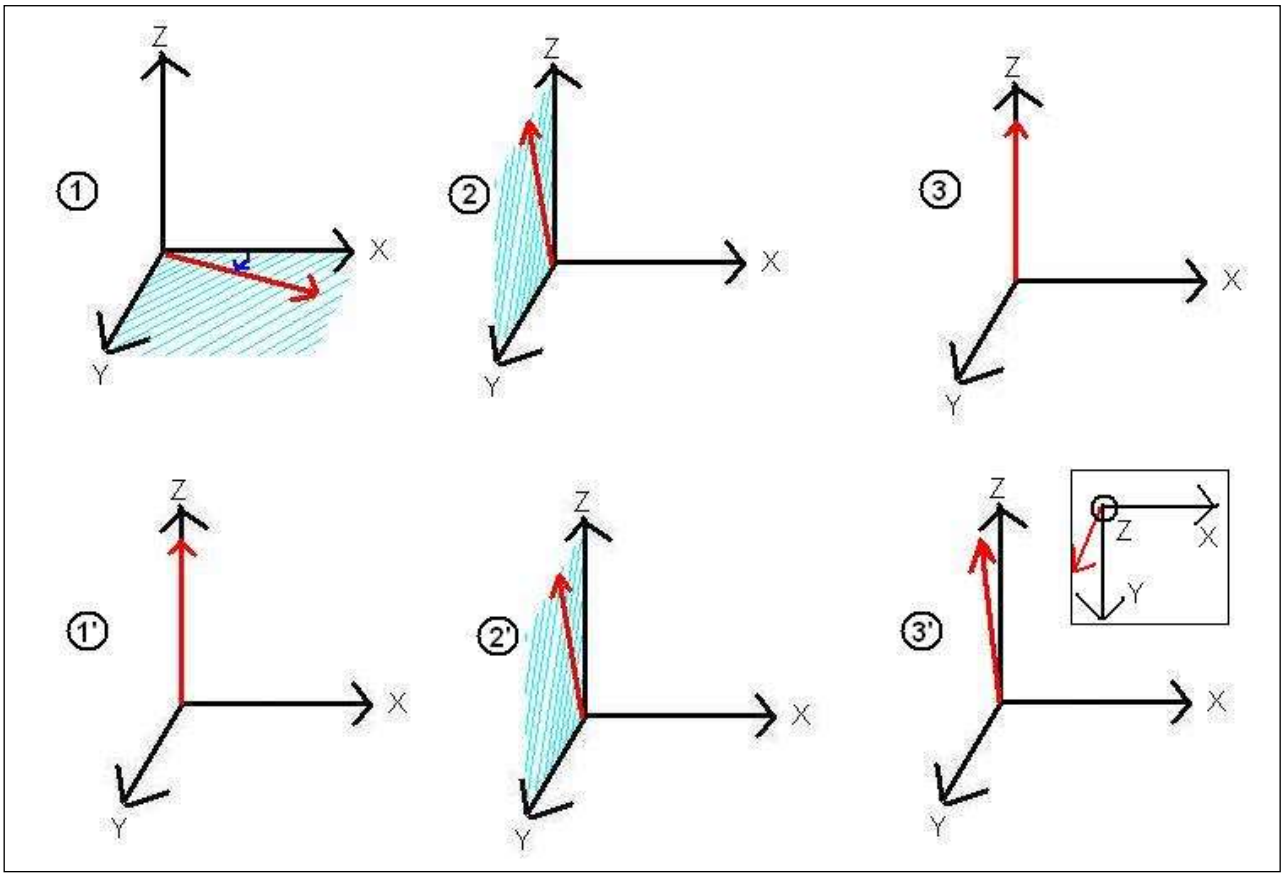



**Figure 29 : Le repliement d'angles**

Dans notre exemple, si on effectue en premier lieu la rotation de 20° sur Z (image 1 de la figure 29), puis la rotation de 90° sur Y (image 2) pour finir par la rotation de -20° sur X (image 3), on se retrouve avec un vecteur confondu avec l'axe Z. L'opération est équivalent à si on n'avait réalisé qu'une rotation de 90° sur Y. On dit donc qu'il y a repliement d'angle. Cependant, si on avait commencé par la rotation de 90° sur Y (image 1') pour continuer par la rotation de -20° sur X (image 2') et finir par celle de 20° sur Z (image 3'), l'angle aurait été différent. En utilisant les quaternions, on s'affranchit de ce problème.

c) L'animation des *Bones* dans le code

Une fois la notion de quaternion assimilée, je me suis mis au code (présent en **Annexe 6** p.55). La première partie du travail à été de récupérer l'armature et les *Bones*. C'est la partie du programme présente de la ligne *40* à la ligne *44*. Dans un premier temps, les données du la scène en cour sont récupérées (lignes *40* et *41*). Puis ce sont celle de l'objet à animer (ligne *42*) pour finir sur celles des *Bones* (lignes *43* et *44*). La liste des *Bones* présent dans l'armature est transférée dans la variable *pbones*. La suite (ligne *45* à *49*) utilise une partie des données précédemment acquises pour initialiser les *Bones* dans leur position de repos (dans Blender, la position dans laquelle les *Bones* ont étés modélisée est appelée position de repos. Tous les mouvements en quaternions sont définis par rapport à cette position et au repère local qui lui est associé (cette position étant considérée comme représentant une rotation de 0° sur X, 0° sur Y et 0° sur Z, même si cela diffère du repère général de Blender)). Si on considère, de la même manière que dans le parafraphe précédent, qu'un quaternion s'écrit Q=a+b.**i**+c.**j**+d.**k**, alors la position d'un *Bone* s'écrit :

`pbone[`*`numéro_du_bone`*`].quat[:]=a,b,c,d`

En l'occurrence, si a=1, b=0, c=0 et d=0, cela met le *Bone* dans sa position de repos. L'instruction suivante (lignes *48* et *49*) assigne la position de chacun des *Bones* à un *Frame*[31] particulier. C'est cette méthode qui est utilisée dans le corps du programme (ligne *88* à *111*). La chose à savoir est que, en quaternion, il est facile de traduire des angles d'Euler simples :

| EULER | QUATERNIONS |
|---|---|
| Rotation de φ sur X | Cos(φ/2) + Sin(φ/2)*i |
| Rotation de φ sur Y | Cos(φ/2) + Sin(φ/2)*k |
| Rotation de φ sur Z | Cos(φ/2) + Sin(φ/2)*j |

Les composées d'angles sont beaucoup plus dur à modéliser, mais une fonction Python (*DifférenceQuats()*) le fait automatiquement. Je crée donc des variables contenant des rotations simples en Quaternion (grâce à la classe *Mathutils* de Blender) que je compose grâce à *DifférenceQuats(*variable1, variable2*)*. La suite est la même manipulation que précédemment. Les seuls ajouts sont à la ligne *109*, où je définis le *Frame* dans lequel les changements de position des *Bones* se sont effectués comme *Frame* à afficher, et à la ligne *111* où je demande à Blender de redessiner le *Frame* actuel. Ainsi, conformément au cahier des charges, cela anime les *Bones.*

[31] Frame signifie Image. Blender étant à la base un logiciel destiné à faire de l'animation, toute position du mannequin doit correspondre à au moins un *Frame* pour être affichée à l'écran.

### *2) La communication avec le programme C++*

L'autre grosse difficulté du programme est de faire communiquer ce programme Python avec le programme pilotant le PROBEX, programmé en C++. Cette communication se fait via une communication client/serveur par sockets.

#### a) Qu'est-ce qu'un socket ?

D'après Wikipédia, « on peut traduire [*socket*] par « connecteur réseau ». Apparu dans les systèmes UNIX, un socket est un élément logiciel qui est aujourd'hui répandu dans la plupart des systèmes d'exploitation. Il s'agit d'une interface logicielle avec les services du système d'exploitation, grâce à laquelle un développeur exploitera facilement et de manière uniforme les services d'un protocole réseau. Il lui sera ainsi par exemple aisé d'établir une session TCP[32], puis de recevoir et d'expédier des données grâce à elle. Cela simplifie sa tâche car cette couche logicielle, de laquelle il requiert des services en appelant des fonctions, masque le nécessaire travail de gestion du réseau, pris en charge par le système. »

Dans nôtre cas, le choix des sockets comme moyen de dialogue s'avère pertinent dans la mesure où il est aisé de mettre en place un serveur et des clients communiquant par ce biais. De plus, cette communication se passe entre deux programmes codés dans des langages différents ayant tous deux une bibliothèque dédiée au dialogue par sockets.

#### b) Le communication par sockets dans le code

Il faut tout d'abord savoir que, dans cette partie « communication », je ne me suis occupé que de la partie Python. L'interface client C++ à été développée par mon maître de stage Mr MORILLEAU. Cependant, pour tester le fonctionnement des sockets (pendant la phase de développement), j'ai créé une interface client sous python. Elle est présente en **Annexe 7** p.57. Son fonctionnement est assez simple. Sur la base de l'interface du programme d'animation du « mannequin à angles droit » (cf. Figure 21 p.26 et lignes *32* à *57* du programme de **l'Annexe 7** p.57), les données récupérées dans les champs sont envoyées part socket (lignes 10 à 17) plutôt que d'être directement traitées. Par ailleurs, la création d'un client (lignes 19 à 31) n'est pas non plus la chose la plus compliquée qu'il soit tant il y a d'exemples sur Internet. Il faut entrer l'adresse IP du serveur (ici 127.0.0.1 car les tests entre les deux programmes Pythons se font en réseau local) ainsi que le port par lequelles sockets vont transiter (ici le 50002, choisi de manière totalement arbitraire) puis tester que ce port de cette adresse IP est bien ouvert (lignes *24* à *28*).

[32] **T**ransmission **C**ontrol **P**rotocol (Protocole de contrôle de transmission)

Du côté du serveur (**Annexe 6** p.55), la démarche est la même (lignes *15* à *28*). Si l'adresse IP de l'hôte diffère, c'est que le client Python n'a servi qu'à faire les tests et que, ceux effectués à partir du programme C++ se sont fait à partir de deux ordinateurs distincts. L'adresse IP de l'hôte ne pet donc plus être 127.0.0.1 mais doit être celle du PC faisant office de serveur. La communication entre les deux programmes (lignes *57* à *72*) se fait avec un protocole de transmission créé par mes soins. Il renvoie 'a' lorsque le serveur à reçu la valeur de l'angle de Flexion du Coude, 'b' lorsqu'il à reçu la valeur de l'angle de la Rotation Axiale, 'c' lorsqu'il à reçu la valeur de l'angle de Flexion du Bras et 'd' lorsqu'il à reçu la valeur de l'angle d'Abduction. D'apparence archaïque, il a l'avantage d'être aisé à mettre en place (aussi bien du côté du serveur que du client) et de faciliter le débogage (il suffit de voir quelle trame n'a pas été envoyée ou n'a pas été reçue pour voir à quel niveau les programmes plantent).

A partir de cette base, des premiers essais ont étés réalisés sur le system en mode Réplication (voir p.17 pour les différents modes de fonctionnement). Ces essais se sont montrés concluants puisque les mouvements réalisés par la personne dans le PROBEX étaient fidèlement reproduits à l'écran. Bénéficiant encore d'un peu de temps avant la fin de mon stage, il m'a été demandé d'apporter quelques améliorations à ce programme.

## C) Les améliorations

### *1) Le mannequin*

L'amazone, bien que d'apparence humaine et modélisée avec suffisamment peu de détails pour s'animer de manière fluide sur un ordinateur de puissance moyenne, n'était pas satisfaisante. En effet, son aspect guerrier est assez éloigné de ce à quoi est sensé ressembler un accidenté venant se faire rééduquer. Il m'a donc été demandé de chercher sur internet d'autres mannequins et de les armaturer. Ils devaient ressembler à des humains contemporains et ne pas être trop lourds à armaturer. La principale difficulté de cette tâche vient du premier point évoqué ci-dessus. En effet, Blender est avant tout

**Figure 30 : Mannequin feminin**

**Figure 31 : Mannequin masculin**

un logiciel fait pour déformer la réalité et modéliser des personnages de cartoons et de jeux vidéo. Il est donc beaucoup plus facile de trouver sur internet des castors ninjas jedi armés de lance flammes que des humains contemporains. Ma recherche m'a menée, dans un premier temps, à trouver deux humains réalisés à l'aide du logiciel Make Human (Figures 30 et 31). Je les ai donc armaturés dans le but de les animer. Mais ils étaient trop détaillés et l'ordinateur n'était pas en mesure de les animer décemment. Le mannequin masculin a quand même servi à illustrer les mouvements dans le logiciel de pilotage du PROBEX en apparaissant sur des boutons (repris p.16) s'animant lorsque le curseur passe dessus.

A force d'écumer le net, j'ai enfin trouvé un mannequin satisfaisant, malheureusement sans équivalent féminin. La fluidité de son animation à fini de le valider comme étant le mannequin définitif

**Figure 32 : Le mannequin choisi**

2) *La barre de progression*


**Figure 33 : La barre de progression**

Suite à la désignation du mannequin précédemment cité comme mannequin définitif et à la refonte du programme pour qu'il lui soit adapté, il m'a été demandé d'y intégrer une barre de progression pour que le patient puisse, à tout moment, savoir à quel point son exercice à progressé.

Le code du programme final est présent en **Annexe 8** p.58. Pour la mettre en œuvre, il m'a fallu dessiner le support apparaissant en vert sur la Figure 3 (ce support est en fait un cylindre) et modéliser les lettres. La méthode utilisée pour modéliser la barre (lignes *100* à *160*) est inspirée de celle du programme d'animation du « mannequin à angles droits » : Le programme dessine un carré puis, en fonction de l'avancement (en pourcent), il en dessine un second plus ou moins éloigné du premier. Il crée ensuite des faces en reliant les côtés des carrés dans le but de modéliser un pavé de taille variable.

Cette modélisation terminée, le programme à été jugé totalement exploitable et mon maître de stage, Mr MORILLEAU, à commencé à l'intégrer à l'interface de commande du PROBEX pour le rendre compatible à tous les modes de fonctionnement et permettre au médecin manipulant le système de faire un prévisualisation du mouvement demandé.

La réalisation de ce sujet m'aura pris 7 semaines, comme le montre ce planning réel des tâches réalisées durant le stage (à mettre en perspective avec le planning prévisionnel présent p.22).

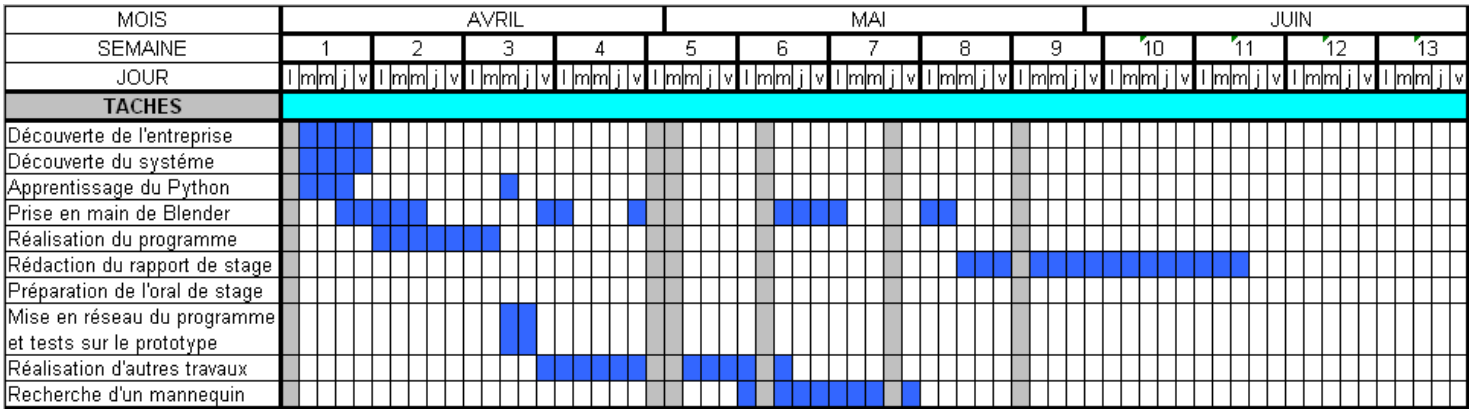



**Figure 34 : Planning réel du stage**

# CONCLUSION TECHNIQUE

Au final, on peut considérer que la réalisation répond à la problématique. Le programme satisfait tous les points du cahier des charges et jouit de quelques fonctionnalités supplémentaires (telle la barre de progression).

Cependant, si cette solution est satisfaisante dans le cadre de la problématique posée, il est possible d'aller plus loin dans le projet de neuro-rééducation par exosquelette. De plus en plus de publications parlent d'intégrer des interfaces proches de celles des jeux vidéo pour focaliser l'attention des patients et les immerger « à la première personne » plutôt que de les laisser passifs dans une interface « à la troisième personne ». C'est dans cette optique que j'ai profité de temps libre pour développer un équivalent du jeu Pong pouvant soit piloter l'exosquelette, soit être piloté par ce dernier (c'est la partie nommée *Réalisation d'autres travaux* dans le planning). Il y a fort à parier que, à moyen terme, c'est dans cette voie que va évoluer la neuro-réhabilitation.

# CONCLUSION GENERALE

Ce stage à été pour moi une expérience enrichissante à plusieurs point de vue :

Tout d'abord, grâce à ce stage, conforter mes choix d'orientation. En effet, j'ai pu me confronter au travail d'informaticien dans un bureau d'étude et tout ce que j'en ai retiré était positif. Cela m'a donné envie d'évoluer dans un environnement de travail tel que celui-ci.

De plus, d'un point de vue technique, j'ai pu mettre en œuvre des compétences acquises lors de ma formation à l'IUT. Celle-ci, de par la pluridisciplinarité qu'elle propose, a aussi facilité mon adaptation face aux différentes situations auxquelles j'ai été confronté durant ce stage. Ce dernier a aussi permis de développer de nouvelle compétences induites par l'apprentissage d'un nouveau langage souvent utilisé dans l'industrie et par la découverte de la modélisation et de l'animation tridimensionnelle.

Ceci confirme que l'IUT GEII fait du technicien supérieur une personne humaine pouvant s'adapter dans n'importe quel milieu de l'entreprise, aussi bien sur le plan technique que humain.

# TABLE DES ANNEXES

# ANNEXE 1 : Feuille d'heure

WOTAN systems
CONCEPTEUR D'ORTHÈSES EXOSQUELETTIQUES

## Feuilles d'heures

**Collaborateur :** Florian ABRY

Semaine N° : 14 du lundi 2 avril 2007 au vendredi 6 avril 2007

| Jour | Rech et dvpt ( Oui/Non) | Projet | Description Taches | | Durée (en heures) |
|---|---|---|---|---|---|
| **Lundi** | | | | **Sous-Total** | ***0*** |
| | OUI | FERIÉ | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| **Mardi** | | | | **Sous-Total** | ***8*** |
| | OUI | Apprentissage Python | Prise en main syntaxe, librairies + réalisation de programmes | | 8 |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| **Mercredi** | | | | **Sous-Total** | ***8*** |
| | OUI | Apprentissage Python | Prise en main bibliothéque graphique Tkinter, fonctions + réalisation de programmes. Premier contact pro. orientée objet | | 8 |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| **Jeudi** | | | | **Sous-Total** | ***8*** |
| | OUI | Apprentissage Python | Prise en main de la programmation orientée objet + réalisation de programmes | | 5 |
| | OUI | Decouverte de Blender | Installation et premiére prise en main. Modélisation d'un cube. | | 3 |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| **Vendredi** | | | | **Sous-Total** | **7** |
| | OUI | Decouverte de Blender | Modélisation de "bonhommes batons" | | 7 |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | OUI | | | | |
| | | | | **Total** | **31** |
| | | | | **Dont R&D** | **31** |

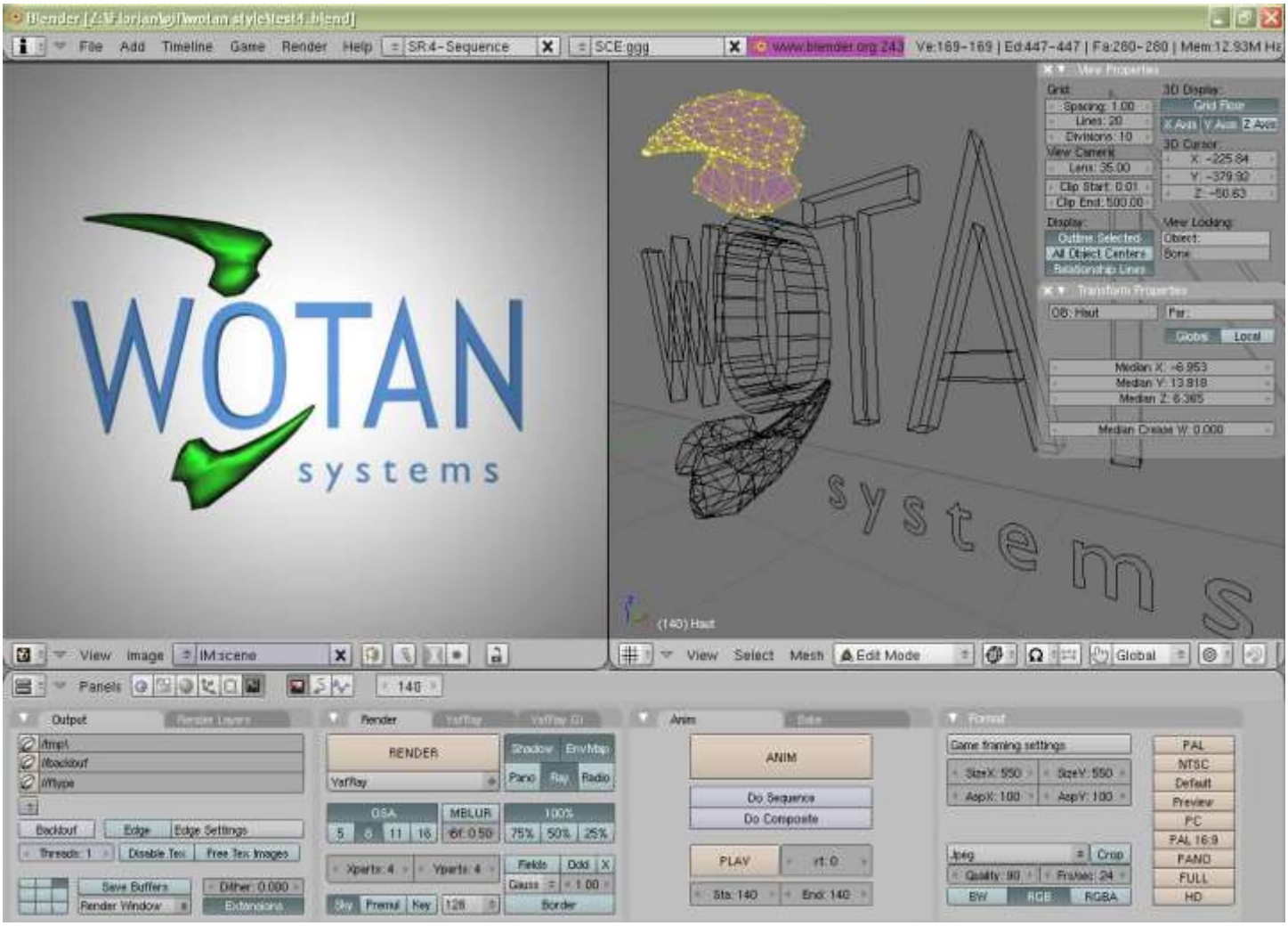
# ANNEXE 2 : Programme de tri de cartes.

```
-*- coding: cp1252 -*- #ajoute les caractères spéciaux comme les accents
from random import * #ajoute la bibliothèque d'instructions générant de l'aléatoire

class JeuDeCartes: #définit une classe nommée JeuDeCartes
    i=0
    cartes=[] #définit la variable cartes comme une liste vide.
    while(i!=4):
        j=2
        while(j!=15):
            cartes.append((j, i)) #ajoute une liste contenant la valeur de i et de
            # j dans la liste cartes
            j+=1
        i+=1

    def nom_carte(self, (val, coul)): #crée une méthode nom_carte. En entrant un
    #numero de valeur et de couleur, il sors le nom de la carte.
        couleur=["coeur", "carreau", "trefle", "pique"]
        valeur=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, "Valet", "Dame", "Roi", "As"]
        a= str(valeur[val])
        b=" de "
        c=str(couleur[coul])
        return a+b+c #fais une seule chaîne de caractère a partir de a, b et c

    def tirer(self): #méthode permettant de tire rune carte du jeu
        c=self.cartes[0]
        del self.cartes[0]
        return c

    def battre(self): #méthode permettant de battre le jeu
        n = len(self.cartes)-1 #récupération du nombre de cartes encre présentes
        a=[]
        while (n>0):
            c=randrange(n)#génere un nombre aléatoire compris entre 0 et le nombre
            #de cartes
            a.append(self.cartes[c])#ajoute la carte dont le numero correspond à la
            #liste a
            del self.cartes[c]
            n=n-1
        a.append(self.cartes[0])
        self.cartes=a

#------------------------------Programme Principal---------------------------------

jeu=JeuDeCartes()#la variable jeu devient un objet de classe carte
jeu.battre()#les cartes sont battues
n=len(jeu.cartes)
while (n>0):
    print jeu.nom_carte(jeu.tirer())#tire les cartes une par une et affiche leur
    #nom à l'écran
    n=n-1
```

# ANNEXE 3 : L’interface de Blender.

*Image tirée du site www.blender.org.*

# ANNEXE 4 : Modélisation du logo de l'entreprise.



# ANNEXE 5 : Premier programme d'animation.

```
import Blender
from Blender import NMesh, Camera, Object, Scene
import math
from math import *
import Tkinter
from Tkinter import *

def dessin():
        global angepaul1, angepaul2, angcoude1, angcoude2, fca, raa, fba, aba, fc,
ra, fb, ab
        fc=eval(entr1.get())
        ra=eval(entr2.get())
        fb=eval(entr3.get())
        ab=eval(entr4.get())
        tp=eval(entr5.get())/0.06226
        fci, rai, fbi, abi = fc, ra, fb, ab
        fc=fc-fca
        ra=ra-raa
        fb=fb-fba
        ab=ab-aba
        k=1
        while(k<=tp+1):

                angepaul1=(k*(ab)*(conv))/tp+aba*conv
                angepaul2=(k*(fb)*(conv))/tp+fba*conv
                angcoude1=(k*(ra)*(conv))/tp+raa*conv
                angcoude2=(k*(fc)*(conv))/tp+fca*conv
                j=0
                i=0
                n=32
                me=NMesh.GetRaw()

                v=NMesh.Vert(0.0, 0.0, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(0, 8.775, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(0.0, 7.125, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(-0.8, 7.125, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(-0.8, 5.425, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(-0.8, 3.925, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(0.8, 7.125, 0.0)
                me.verts.append(v)
                v=NMesh.Vert(me.verts[6][0]+1.7*sin(angepaul1)*cos(angepaul2),
                                        me.verts[6][1]-
1.7*cos(angepaul1)*cos(angepaul2),
                                        me.verts[6][2]+1.7*sin(angepaul2))
                me.verts.append(v)

        v=NMesh.Vert(me.verts[7][0]+1.5*sin(angepaul1+angcoude1)*cos(angepaul2+angco
ude2),
                                        me.verts[7][1]-
1.5*cos(angepaul1+angcoude1)*cos(angepaul2+angcoude2),
                                        me.verts[7][2]+1.5*sin(angepaul2+angcoude2))
        #8
                me.verts.append(v)
```

```
60                  #Definition Volume
61
62                  #Point 0
63                  v=NMesh.Vert(-0.05, 0, 0.05)
64                  me.verts.append(v)
65                  v=NMesh.Vert(0.05, 0, 0.05)
66                  me.verts.append(v)
67                  v=NMesh.Vert(0.05, 0, -0.05)
68                  me.verts.append(v)
69                  v=NMesh.Vert(-0.05, 0, -0.05)
70                  me.verts.append(v)
71
72                  #Point 1
73                  v=NMesh.Vert(-0.05, 8.775, 0.05)
74                  me.verts.append(v)
75                  v=NMesh.Vert(0.05, 8.775, 0.05)
76                  me.verts.append(v)
77                  v=NMesh.Vert(0.05, 8.775, -0.05)
78                  me.verts.append(v)
79                  v=NMesh.Vert(-0.05, 8.775, -0.05)
80                  me.verts.append(v)
81
82                  #Point 2
83                  v=NMesh.Vert(0, 7.125-0.05, 0.05)
84                  me.verts.append(v)
85                  v=NMesh.Vert(0, 7.125+0.05, 0.05)
86                  me.verts.append(v)
87                  v=NMesh.Vert(0, 7.125+0.05, -0.05)
88                  me.verts.append(v)
89                  v=NMesh.Vert(0, 7.125-0.05, -0.05)
90                  me.verts.append(v)
91
92                  #Point 3
93                  v=NMesh.Vert(-0.8+0.05, 7.125-0.05, 0.05)
94                  me.verts.append(v)
95                  v=NMesh.Vert(-0.8-0.05, 7.125+0.05, 0.05)
96                  me.verts.append(v)
97                  v=NMesh.Vert(-0.8-0.05, 7.125+0.05, -0.05)
98                  me.verts.append(v)
99                  v=NMesh.Vert(-0.8+0.05, 7.125-0.05, -0.05)
100                 me.verts.append(v)
101
102                 #Point 5
103                 v=NMesh.Vert(-0.8+0.05, 3.925, 0.05)
104                 me.verts.append(v)
105                 v=NMesh.Vert(-0.8-0.05, 3.925, 0.05)
106                 me.verts.append(v)
107                 v=NMesh.Vert(-0.8-0.05, 3.925, -0.05)
108                 me.verts.append(v)
109                 v=NMesh.Vert(-0.8+0.05, 3.925, -0.05)
110                 me.verts.append(v)
111
112                 #Point 6 horizontal
113                 v=NMesh.Vert(0.8, 7.125-0.05, 0.05)
114                 me.verts.append(v)
115                 v=NMesh.Vert(0.8, 7.125+0.05, 0.05)
116                 me.verts.append(v)
117                 v=NMesh.Vert(0.8, 7.125+0.05, -0.05)
118                 me.verts.append(v)
119                 v=NMesh.Vert(0.8, 7.125-0.05, -0.05)
120                 me.verts.append(v)
121
122                 #Point 6 orthonormal à la ligne
```

```
                v=NMesh.Vert(me.verts[6][0]+0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #33
                                        me.verts[6][1]-0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[6][2]-0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[6][0]-0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #34
                                        me.verts[6][1]+0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[6][2]+0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[6][0]+0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #35
                                        me.verts[6][1]-0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[6][2]+0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[6][0]-0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #36
                                        me.verts[6][1]+0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[6][2]-0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)


                #point 7->6
                v=NMesh.Vert(me.verts[7][0]+0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #37
                                        me.verts[7][1]-0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[7][2]-0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]-0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #38
                                        me.verts[7][1]+0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[7][2]+0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]+0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #39
                                        me.verts[7][1]-0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[7][2]+0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]-0.05*((me.verts[6][1]-
me.verts[7][1])/1.7),          #40
                                        me.verts[7][1]+0.05*((me.verts[6][0]-
me.verts[7][0]/1.7)),
                                        me.verts[7][2]-0.05*(1-(me.verts[6][2]-
me.verts[7][2])/1.7))
                me.verts.append(v)

                #Point  7->8
                v=NMesh.Vert(me.verts[7][0]+0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #41
```

```
                                        me.verts[7][1]-0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[7][2]-0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]-0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #42
                                        me.verts[7][1]+0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[7][2]+0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]+0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #43
                                        me.verts[7][1]-0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[7][2]+0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[7][0]-0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #44
                                        me.verts[7][1]+0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[7][2]-0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)

                #Point  8
                v=NMesh.Vert(me.verts[8][0]+0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #45
                                        me.verts[8][1]-0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[8][2]-0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[8][0]-0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #46
                                        me.verts[8][1]+0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[8][2]+0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[8][0]+0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #47
                                        me.verts[8][1]-0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[8][2]+0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)
                v=NMesh.Vert(me.verts[8][0]-0.05*((me.verts[7][1]-
me.verts[8][1])/1.5),          #48
                                        me.verts[8][1]+0.05*((me.verts[7][0]-
me.verts[8][0]/1.5)),
                                        me.verts[8][2]-0.05*(1-(me.verts[7][2]-
me.verts[8][2])/1.5))
                me.verts.append(v)

                #Sphere 1
                rayon=0.1
                for i in range(0, n):
                        for j in range(0, n):
                                x=sin(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon+0.8
                                y=cos(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon+7.125
                                z=sin(-pi/2+i*pi/(n-1))*rayon
```

```
                                v=NMesh.Vert(x,y,z)
                                me.verts.append(v)

                n0=len(range(0,n))
                for i in range(0, n-1):
                    for j in range(49, 48+n-1):
                        f=NMesh.Face()
                        f.v.append(me.verts[i*n0+j])
                        f.v.append(me.verts[i*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j])
                        me.faces.append(f)

                #Sphere 2
                i=0
                j=0
                n=32
                rayon=0.1
                for i in range(0, n):
                        for j in range(0, n):
                                x=sin(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-
1))*rayon+me.verts[7][0]
                                y=cos(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-
1))*rayon+me.verts[7][1]
                                z=sin(-pi/2+i*pi/(n-1))*rayon+me.verts[7][2]
                                v=NMesh.Vert(x,y,z)
                                me.verts.append(v)

                n0=len(range(0,n))
                for i in range(0, n-1):
                    for j in range(1073, 1073+n-1):
                        f=NMesh.Face()
                        f.v.append(me.verts[i*n0+j])
                        f.v.append(me.verts[i*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j])
                        me.faces.append(f)

                #Sphere gauche 1
                i=0
                j=0
                n=32
                rayon=0.1
                for i in range(0, n):
                        for j in range(0, n):
                                x=sin(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon-0.8
                                y=cos(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon+7.125
                                z=sin(-pi/2+i*pi/(n-1))*rayon
                                v=NMesh.Vert(x,y,z)
                                me.verts.append(v)

                n0=len(range(0,n))
                for i in range(0, n-1):
                    for j in range(2097, 2097+n-1):
                        f=NMesh.Face()
                        f.v.append(me.verts[i*n0+j])
                        f.v.append(me.verts[i*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j])
                        me.faces.append(f)

                #Sphere gauche 2
                i=0
                j=0
```

```
                n=32
                rayon=0.1
                for i in range(0, n):
                        for j in range(0, n):
                                x=sin(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon-0.8
                                y=cos(j*pi*2/(n-1))*cos(-pi/2+i*pi/(n-1))*rayon+5.425
                                z=sin(-pi/2+i*pi/(n-1))*rayon
                                v=NMesh.Vert(x,y,z)
                                me.verts.append(v)

                n0=len(range(0,n))
                for i in range(0, n-1):
                    for j in range(3121, 3121+n-1):
                        f=NMesh.Face()
                        f.v.append(me.verts[i*n0+j])
                        f.v.append(me.verts[i*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j+1])
                        f.v.append(me.verts[(i+1)*n0+j])
                        me.faces.append(f)

                me.addEdge(me.verts[0], me.verts[1])
                me.addEdge(me.verts[2], me.verts[3])
                me.addEdge(me.verts[3], me.verts[4])
                me.addEdge(me.verts[4], me.verts[5])
                me.addEdge(me.verts[2], me.verts[6])
                me.addEdge(me.verts[6], me.verts[7])
                me.addEdge(me.verts[7], me.verts[8])

                #Pavé 1
                face=NMesh.Face()
                face.v.append(me.verts[9])
                face.v.append(me.verts[10])
                face.v.append(me.verts[11])
                face.v.append(me.verts[12])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[9])
                face.v.append(me.verts[10])
                face.v.append(me.verts[14])
                face.v.append(me.verts[13])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[10])
                face.v.append(me.verts[11])
                face.v.append(me.verts[15])
                face.v.append(me.verts[14])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[11])
                face.v.append(me.verts[12])
                face.v.append(me.verts[16])
                face.v.append(me.verts[15])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[12])
                face.v.append(me.verts[9])
                face.v.append(me.verts[13])
                face.v.append(me.verts[16])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[13])
                face.v.append(me.verts[14])
                face.v.append(me.verts[15])
                face.v.append(me.verts[16])
```

```
                me.faces.append(face)

                #Pavé 2
                face=NMesh.Face()
                face.v.append(me.verts[17])
                face.v.append(me.verts[18])
                face.v.append(me.verts[19])
                face.v.append(me.verts[20])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[17])
                face.v.append(me.verts[21])
                face.v.append(me.verts[22])
                face.v.append(me.verts[18])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[18])
                face.v.append(me.verts[22])
                face.v.append(me.verts[23])
                face.v.append(me.verts[19])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[19])
                face.v.append(me.verts[23])
                face.v.append(me.verts[24])
                face.v.append(me.verts[20])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[20])
                face.v.append(me.verts[24])
                face.v.append(me.verts[21])
                face.v.append(me.verts[17])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[21])
                face.v.append(me.verts[22])
                face.v.append(me.verts[23])
                face.v.append(me.verts[24])
                me.faces.append(face)

                #Pavé 3
                face=NMesh.Face()
                face.v.append(me.verts[21])
                face.v.append(me.verts[22])
                face.v.append(me.verts[23])
                face.v.append(me.verts[24])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[21])
                face.v.append(me.verts[22])
                face.v.append(me.verts[26])
                face.v.append(me.verts[25])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[22])
                face.v.append(me.verts[23])
                face.v.append(me.verts[27])
                face.v.append(me.verts[26])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[23])
                face.v.append(me.verts[24])
                face.v.append(me.verts[28])
                face.v.append(me.verts[27])
```

```
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[24])
                face.v.append(me.verts[21])
                face.v.append(me.verts[25])
                face.v.append(me.verts[28])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[25])
                face.v.append(me.verts[26])
                face.v.append(me.verts[27])
                face.v.append(me.verts[28])
                me.faces.append(face)

                #Pavé 4
                face=NMesh.Face()
                face.v.append(me.verts[17])
                face.v.append(me.verts[18])
                face.v.append(me.verts[19])
                face.v.append(me.verts[20])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[17])
                face.v.append(me.verts[29])
                face.v.append(me.verts[30])
                face.v.append(me.verts[18])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[18])
                face.v.append(me.verts[30])
                face.v.append(me.verts[31])
                face.v.append(me.verts[19])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[19])
                face.v.append(me.verts[31])
                face.v.append(me.verts[32])
                face.v.append(me.verts[20])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[20])
                face.v.append(me.verts[32])
                face.v.append(me.verts[29])
                face.v.append(me.verts[17])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[29])
                face.v.append(me.verts[30])
                face.v.append(me.verts[31])
                face.v.append(me.verts[32])
                me.faces.append(face)

                #Pavé 5
                face=NMesh.Face()
                face.v.append(me.verts[33])
                face.v.append(me.verts[36])
                face.v.append(me.verts[34])
                face.v.append(me.verts[35])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[33])
                face.v.append(me.verts[36])
                face.v.append(me.verts[40])
                face.v.append(me.verts[37])
```

```
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[36])
                face.v.append(me.verts[34])
                face.v.append(me.verts[38])
                face.v.append(me.verts[40])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[34])
                face.v.append(me.verts[35])
                face.v.append(me.verts[39])
                face.v.append(me.verts[38])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[35])
                face.v.append(me.verts[33])
                face.v.append(me.verts[37])
                face.v.append(me.verts[39])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[37])
                face.v.append(me.verts[40])
                face.v.append(me.verts[38])
                face.v.append(me.verts[39])
                me.faces.append(face)

                #Pavé 6
                face=NMesh.Face()
                face.v.append(me.verts[41])
                face.v.append(me.verts[44])
                face.v.append(me.verts[42])
                face.v.append(me.verts[43])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[41])
                face.v.append(me.verts[44])
                face.v.append(me.verts[48])
                face.v.append(me.verts[45])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[44])
                face.v.append(me.verts[42])
                face.v.append(me.verts[46])
                face.v.append(me.verts[48])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[42])
                face.v.append(me.verts[43])
                face.v.append(me.verts[47])
                face.v.append(me.verts[46])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[43])
                face.v.append(me.verts[41])
                face.v.append(me.verts[45])
                face.v.append(me.verts[47])
                me.faces.append(face)
                face=NMesh.Face()
                face.v.append(me.verts[45])
                face.v.append(me.verts[48])
                face.v.append(me.verts[46])
                face.v.append(me.verts[47])
                me.faces.append(face)
                NMesh.PutRaw(me, "Cube", 1)
```

```
                Blender.Redraw()
                k=k+1
        fca, raa, fba, aba=fci, rai, fbi, abi

#---------------------------Programme Principal-------------------------------

conv=pi/180

angepaul1=0*conv
angepaul2=0*conv
angcoude1=0*conv
angcoude2=0*conv
fca, raa, fba, aba = 0, 0, 0, 0

fen1 = Tk()
txth = Label(fen1, text="Entrez les angles en degré :")
txth.pack(side=TOP)
txt1 = Label(fen1, text = "\n\nFlexion Coude (FC 1) :")
txt1.pack()
entr1 = Entry(fen1)
entr1.pack()
txt2 = Label(fen1, text = "\nRotation Axiale (RA 2) :")
txt2.pack()
entr2 = Entry(fen1)
entr2.pack()
txt3 = Label(fen1, text = "\nFlexion Bras (FB 3) :")
txt3.pack()
entr3 = Entry(fen1)
entr3.pack()
txt4 = Label(fen1, text = "\nAbduction (AB 4) :")
txt4.pack()
entr4 = Entry(fen1)
entr4.pack()
txt5 = Label(fen1, text = "\nTemps (en secondes) :")
txt5.pack()
entr5 = Entry(fen1)
entr5.pack()
bou2 = Button(fen1, text="Dessiner", command=dessin)
bou2.pack()
bou1 = Button(fen1, text="Quitter", command=fen1.destroy)
bou1.pack(side=BOTTOM)
fen1.mainloop()
```

# ANNEXE 6 : Le programme d'animation de l'Amazone

```
#-----------------------------IMPORTATIONS-------------------------------

import socket, sys

import math

from math import *

import Blender
from Blender import *
from Blender.Mathutils import *

#---------------------------CREATION DU SERVEUR-----------------------------

HOST = '169.254.60.183'
PORT = 50002
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    mySocket.bind((HOST, PORT))
except socket.error:
    print "La liaison du socket à l'adresse choisie a échoué."
    sys.exit()

print "Serveur prêt, en attente du programme C++"
mySocket.listen(5)
connexion, adresse = mySocket.accept()
print "Client connecté, adresse IP %s, port %s" % (adresse[0], adresse[1])
connexion.send("Vous êtes connecté au serveur Python")

#------------------------INITIALISATION DES VARIABLES-----------------------

conv=(pi/360)
frame = 1
k=0
a=2
fba, aba, fca, raa = 0,0,0,0

#---------------------------RECUPERATION DES BONES---------------------------

scn = Scene.GetCurrent()
context = scn.getRenderingContext()
arm_ob = Object.GetSelected()[0]
pose = arm_ob.getPose()
pbones = pose.bones.values()
pbones[0].quat[:] = 1,0,0,0
pbones[1].quat[:] = 1,0,0,0
pbones[2].quat[:] = 1,0,0,0
for i in xrange(4):
        pbones[i].insertKey(arm_ob, frame, Object.Pose.ROT)

#-----------------------DEBUT DU PROGRAMME DE DIALOGUE----------------------

while(1):

        #---------RECUPERATION DES DONNES ENVOYEES PAR LE PROGRAMME C++---------

        msgClient = connexion.recv(1024)
        print "S>", msgClient
        fc=float(msgClient)
        connexion.send("a")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
```

```
        ra=float(msgClient)
        connexion.send("b")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        fb=float(msgClient)
        connexion.send("c")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        ab=float(msgClient)
        connexion.send("d")
        print "bbb"
        #flexion epaule
        angepaul1 = fb-fba
        #abduction
        angepaul2 = ab-aba
        #flexion coude
        angcoude1 = fc-fca
        #rotation axianle
        angcoude2 = ra-raa
        #pbones[0] => Bone.001
        #pbones[1] => Bone.002
        #pbones[2] => Bone

        #--------------CALCUL DE LA NOUVELLE POSITION DU MANNEQUIN--------------

        frame = a
        aex = Mathutils.Quaternion(cos(angepaul1*conv+fba*conv),sin(-angepaul1*
conv-fba*conv),0,0)
        aey = Mathutils.Quaternion(cos(angepaul2*conv+aba*conv),0,0,sin(-
angepaul2*conv-aba*conv))
        aez = Mathutils.Quaternion(cos(-13*conv),0,sin(-13*conv),0)
        aet = DifferenceQuats(aey, aex)
        ae = DifferenceQuats(aez, aet)
        acx = Mathutils.Quaternion(cos(angcoude1*conv+fca*conv),sin(-angcoude1*
conv-fca*conv),0,0)
        acy = Mathutils.Quaternion(cos(angcoude2*conv+raa*conv),0,sin(angcoude2*
conv+raa*conv),0)
        ac = DifferenceQuats(acy, acx)
        pbones[0].quat[:] = ac
        pbones[1].quat[:] = 1,0,0,0
        pbones[2].quat[:] = ae

        #------------AFFICHAGE DE LA NOUVELLE POSITION DU MANNEQUIN------------

        for i in xrange(4):
                pbones[i].insertKey(arm_ob, frame, Object.Pose.ROT)
        context.currentFrame(a)
        a=a+1
        Blender.Redraw()
        fba = fb
        aba = ab
        fca = fc
        raa = ra
        print "cc"
```

# ANNEXE 7 : Le client Python

```
1  import socket, sys, Tkinter
2  from Tkinter import *
3
4  def envoi():
5          fc=entr1.get()
6          ra=entr2.get()
7          fb=entr3.get()
8          ab=entr4.get()
9          tp=entr5.get()
10         mySocket.send(fc)
11         msgServeur = mySocket.recv(1024)
12         mySocket.send(ra)
13         msgServeur = mySocket.recv(1024)
14         mySocket.send(fb)
15         msgServeur = mySocket.recv(1024)
16         mySocket.send(ab)
17         msgServeur = mySocket.recv(1024)
18
19 HOST = '127.0.0.1'
20 PORT = 50002
21 # 1) création du socket :
22 mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
23 # 2) envoi d'une requête de connexion au serveur :
24 try:
25         mySocket.connect((HOST, PORT))
26 except socket.error:
27         print "La connexion a échoué."
28         sys.exit()
29 print "Connexion établie avec le serveur."
30 # 3) Dialogue avec le serveur :
31 msgServeur = mySocket.recv(1024)
32 fen1 = Tk()
33 txth = Label(fen1, text="Entrez les angles en degré :")
34 txth.pack(side=TOP)
35 txt1 = Label(fen1, text = "\n\nFlexion Coude (FC 1) :")
36 txt1.pack()
37 entr1 = Entry(fen1)
38 entr1.pack()
39 txt2 = Label(fen1, text = "\nRotation Axiale (RA 2) :")
40 txt2.pack()
41 entr2 = Entry(fen1)
42 entr2.pack()
43 txt3 = Label(fen1, text = "\nFlexion Bras (FB 3) :")
44 txt3.pack()
45 entr3 = Entry(fen1)
46 entr3.pack()
47 txt4 = Label(fen1, text = "\nAbduction (AB 4) :")
48 txt4.pack()
49 entr4 = Entry(fen1)
50 entr4.pack()
51 txt5 = Label(fen1, text = "\nTemps (en secondes) :")
52 txt5.pack()
53 entr5 = Entry(fen1)
54 entr5.pack()
55 bou1 = Button(fen1, text="Envoyer", command=envoi)
56 bou1.pack()
57 fen1.mainloop()
58 # 4) Fermeture de la connexion :
59 print "Connexion interrompue."
60 mySocket.close()
```

# ANNEXE 8 : Le programme final

```
import socket, sys

import math
from math import *

import Blender
from Blender import *
from Blender.Mathutils import *

HOST = '127.0.0.1'
PORT = 50002
mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    mySocket.bind((HOST, PORT))
except socket.error:
    print "La liaison du socket à l'adresse choisie a échoué."
    sys.exit()

print "Serveur prêt, en attente du programme C++"
mySocket.listen(5)
connexion, adresse = mySocket.accept()
print "Client connecté, adresse IP %s, port %s" % (adresse[0], adresse[1])
connexion.send("Vous êtes connecté au serveur Python")
conv=(pi/360)
frame = 1
scn = Scene.GetCurrent()
context = scn.getRenderingContext()
arm_ob = Object.GetSelected()[0]
pose = arm_ob.getPose()
act = arm_ob.getAction()
if not act:
        act = Armature.NLA.NewAction()
        act.setActive(arm_ob)
xbones = arm_ob.data.bones.values()
pbones = pose.bones.values()
pbones[10].quat[:] = 1,0,0,0
pbones[27].quat[:] = 1,0,0,0
pbones[28].quat[:] = 1,0,0,0
for i in xrange(4):
        pbones[i].insertKey(arm_ob, frame, Object.Pose.ROT)

c = 52*pi/180
p = 0
k=0
a=2
fba, aba, fca, raa = 0,0,0,0

while(1):
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        fc=float(msgClient)
        connexion.send("a")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        ra=float(msgClient)
        connexion.send("b")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        fb=float(msgClient)
        connexion.send("c")
        msgClient = connexion.recv(1024)
```

```
        print "S>", msgClient
        ab=float(msgClient)
        connexion.send("d")
        msgClient = connexion.recv(1024)
        print "S>", msgClient
        pr=float(msgClient)
        connexion.send("e")
        print "bbb"
        #flexion epaule
        angepaul1 = -fb
        #abduction
        angepaul2 = -ab
        #flexion coude
        angcoude1 = fc-15
        #rotation axianle
        angcoude2 = ra+35
        p = pr*0.131
        #pbones.quat[:] = cos(ang), sin(angY), sin(angZ), sin(angX)
        frame = a
        aex = Mathutils.Quaternion(cos(angepaul1*conv),sin(-angepaul1*conv),0,0)
        aey = Mathutils.Quaternion(cos(angcoude2*conv),0,0,sin(-angepaul2*conv))
        aez = Mathutils.Quaternion(cos(angepaul2*conv),0,sin(angcoude2*conv),0)
        aet = DifferenceQuats(aey, aex)
        ae = DifferenceQuats(aez, aet)
        acx = Mathutils.Quaternion(cos(angcoude1*conv),sin(-angcoude1*conv),0,0)
        acy = Mathutils.Quaternion(cos(-20*conv),0,0,sin(-20*conv))
        ac = DifferenceQuats(acy, acx)
        amz = Mathutils.Quaternion(cos(-90*conv),0,sin(-90*conv),0)
        amy = Mathutils.Quaternion(cos(-20*conv),sin(-20*conv),0,0)
        am = DifferenceQuats(amz, amy)
        pbones[27].quat[:] = ae
        pbones[28].quat[:] = ac
        pbones[10].quat[:] = am
        for i in xrange(4):
                pbones[i].insertKey(arm_ob, frame, Object.Pose.ROT)
        context.currentFrame(a)

        me = NMesh.GetRaw()
        v = NMesh.Vert(-0.770, 6.000, 7.970)
        me.verts.append(v)
        v = NMesh.Vert(-0.580, 6.235, 7.970)
        me.verts.append(v)
        v = NMesh.Vert(-0.580, 6.235, 7.670)
        me.verts.append(v)
        v = NMesh.Vert(-0.770, 6.000, 7.670)
        me.verts.append(v)
        v = NMesh.Vert(-0.770+p*sin(c), 6.000-p*cos(c), 7.970)
        me.verts.append(v)
        v = NMesh.Vert(-0.580+p*sin(c), 6.235-p*cos(c), 7.970)
        me.verts.append(v)
        v = NMesh.Vert(-0.580+p*sin(c), 6.235-p*cos(c), 7.670)
        me.verts.append(v)
        v = NMesh.Vert(-0.770+p*sin(c), 6.000-p*cos(c), 7.670)
        me.verts.append(v)

        face = NMesh.Face()
        face.v.append(me.verts[0])
        face.v.append(me.verts[1])
        face.v.append(me.verts[2])
        face.v.append(me.verts[3])
        me.faces.append(face)

        face = NMesh.Face()
        face.v.append(me.verts[4])
```

```
        face.v.append(me.verts[5])
        face.v.append(me.verts[6])
        face.v.append(me.verts[7])
        me.faces.append(face)

        face = NMesh.Face()
        face.v.append(me.verts[1])
        face.v.append(me.verts[0])
        face.v.append(me.verts[4])
        face.v.append(me.verts[5])

        me.faces.append(face)
        face = NMesh.Face()
        face.v.append(me.verts[2])
        face.v.append(me.verts[1])
        face.v.append(me.verts[5])
        face.v.append(me.verts[6])
        me.faces.append(face)

        face = NMesh.Face()
        face.v.append(me.verts[3])
        face.v.append(me.verts[2])
        face.v.append(me.verts[6])
        face.v.append(me.verts[7])
        me.faces.append(face)

        face = NMesh.Face()
        face.v.append(me.verts[0])
        face.v.append(me.verts[3])
        face.v.append(me.verts[7])
        face.v.append(me.verts[4])
        me.faces.append(face)

        NMesh.PutRaw(me, 'prog', 1)
        a=a+1
        Blender.Redraw()
        fba = fb
        aba = ab
        fca = fc
        raa = ra
        print "cc"
```