POLYTECH©
NICE-SOPHIA

Meilleur au ping-pong grâce à une montre ?

Rapport du projet P16

Encadrant : Frédéric Precioso
Par Daniel Anbarasu, Thibaut Dufour, David Guerrero, Thomas Pascal

Du 07/01/2014 au 07/02/2014

Sommaire

Description du projet

Description donnée par le tuteur :

Ce projet vise à acquérir des données d'accéléromètre de la montre et à les analyser pour ensuite permettre d'améliorer la technique d'un joueur de ping-pong à partir du jeu Chronos Tennis (http://processors.wiki.ti.com/index.php/Chronos_Tennis , http://vimeo.com/13844131). On peut imaginer si le groupe de projet travaille suffisamment bien écrire un jeu de ping-pong avec utilisation de la montre, voire que l'adversaire soit modélisé à partir de profils de joueur enregistrés.

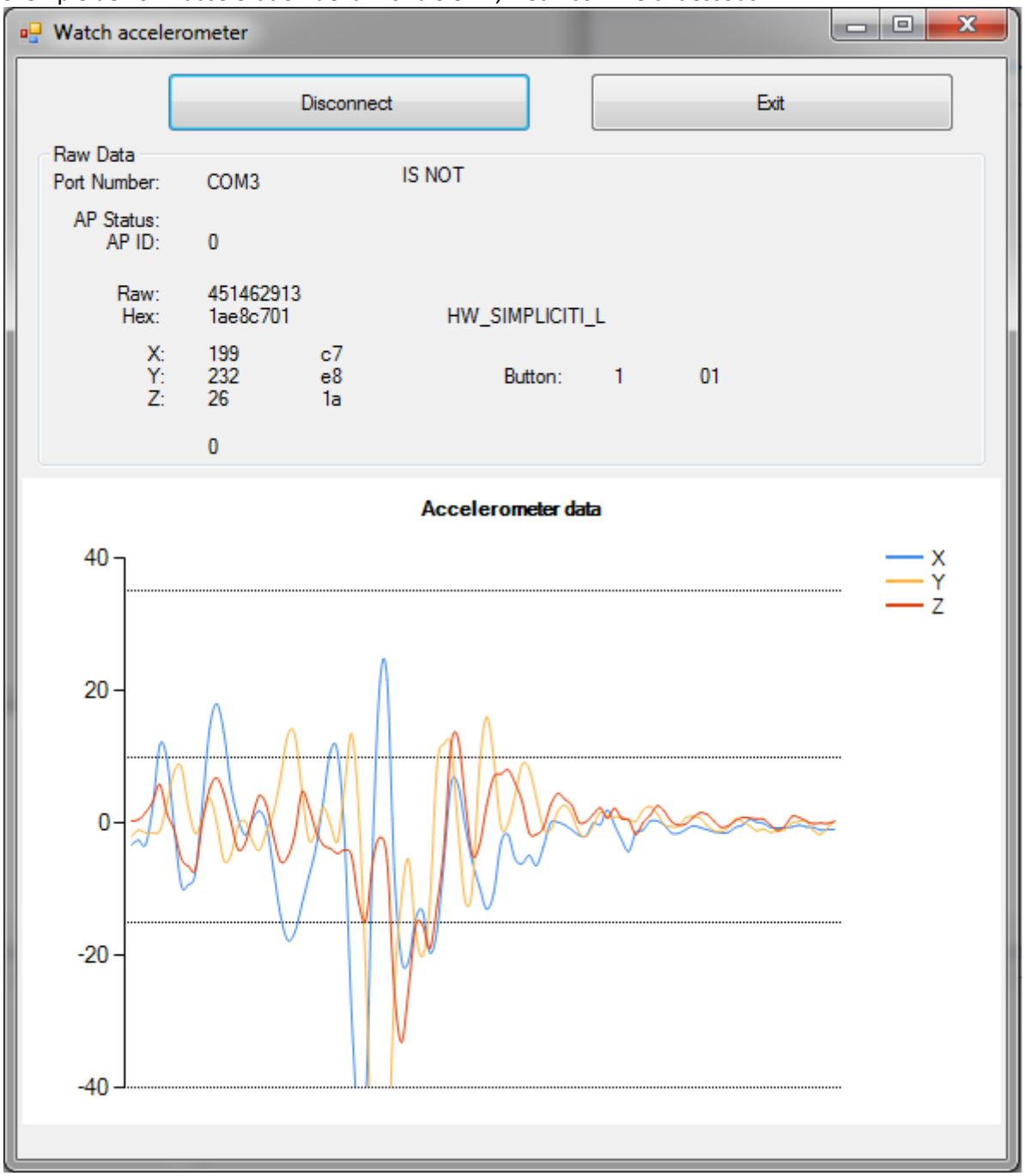
Plus précisément, le but du projet est, grâce à la montre ez430 chronos de Texas Instrument, d'étudier les mouvements effectués par un joueur de ping-pong afin de lui permettre de s'améliorer.
Il nous a été demandé de réaliser le projet en C#, langage qu'aucun d'entre nous n'avait étudié précédemment.
Nous sommes rapidement dirigés sur deux grands axes :

- Un axe d'analyse de mouvement, dont la finalité est la comparaison d'un mouvement effectué montre au poignet à un mouvement de notre base de donnée, par exemple effectué par un joueur professionnel.
- Un axe jeu 3D, plus ludique, qui à terme prendrait en compte la première partie pour attribuer des points en fonctions de la proximité du mouvement effectuée avec le mouvement idéal.

Nous nous sommes rapidement rendu compte que les données envoyées par la montre n'était pas assez précise ni envoyée assez rapidement pour permettre l'analyse d'un mouvement de ping-pong, qui est très bref et sur une amplitude de mouvement faible, et donc sommes parti sur un jeu, et des mouvements de tennis. En effet, ceux-ci sont plus ample moins rapide et donc plus facile à analyser avec peu de points.

Installation

Nécessaire à l'utilisation de la montre :
Tout est détaillé ici : http://processors.wiki.ti.com/index.php/EZ430-Chronos
Il faut tout d'abord télécharger le Control Center for Windows (http://www.ti.com/lit/zip/slac341) puis l'installé en lançant le Chronos-Setup.exe tout juste téléchargé. Ce contrôle center permet par exemple de voir l'accélération de la montre en X, Y et Z comme ci-dessous :



Il faudra ensuite télécharger notre projet et lancer l'exécutable.
Attention, les sources disponibles sur git ne contiennent pas les images/textures, car celles-ci sont trop lourde pour y être mise, donc le projet sur git ne fonctionne pas. Afin que celui-ci compile, récupérerez l'archives contenant le répertoire content et décompressez le dans le dossier XNA2(http://heldroe.org/~heldroe/Content.zip).

Fonctionnalités

Grâce à la montre ez430 chronos, vous aurez la possibilité de jouer au tennis en faisant simplement des mouvements, plus besoin de souris ou de clavier !
Le jeu que nous avons créé permet de s'entrainer au tennis, et de voir si les gestes effectués sont bons ou non. Vous n'avez pas besoin de faire déplacer votre personnage, celui-ci se déplacera automatiquement vers la balle afin d'être capable de la frapper en temps voulu. A vous ensuite de faire le mouvement adéquat pour l'envoyer à gauche (revers), au milieu (coup droit croisé) ou à droite (coup droit arrêté).
Une fois votre mouvement effectué, vous aurez une indication sur la proximité entre votre geste et le geste idéal (qui est un pourcentage de proximité), ainsi qu'une appréciation sur le moment ou vous avez tapé la balle (savoir si vous avez frappé trop tôt, trop tard, ou pile au bon moment).

En plus de cela, nous avons laissé accès aux données brutes de l'accéléromètre. Ce sont les données que nous recevons de la montre et que nous analysons. Laissé tout d'abord car cela nous a aidé à développer, nous avons choisi de les laisser car cela peut aider à comprendre le fonctionnement du programme, et donc aider à sa maintenance et à son évolution.

Recherche

Répartition

La première semaine du projet a surtout été consacrée à la recherche, à la fois au niveau de l'environnement de développement (C#) et des capacités de la montre (accéléromètre).
Nous avons tout d'abord cherché à lier les données de la montre à une application écrite en C# afin de pouvoir analyser les données.

Bibliothèque de communication

Nous avons utilisé la bibliothèque de communication « eZ430 Chronos .Net » (http://ez430chronosnet.sourceforge.net/) pour lier une première application à la montre.
En reprenant un code d'exemple fourni, nous nous sommes aperçus que la bibliothèque était fonctionnelle pour un seul driver en particulier (des fois fournis dans des applications utilisant la montre), qui n'est pas celui fourni officiellement par Texas Instruments, le fabricant de la montre.

En recherchant dans le code de la bibliothèque et à l'aide du gestionnaire de périphériques Windows, nous avons été en mesure de corriger le code et de recompiler un fichier de bibliothèque (DLL) pour nos futures applications. La bibliothèque étant un projet open-source, un rapport de bug a été soumis, afin d'aider les prochains utilisateurs.

L'application d'exemple basique (affichage des données de la montre) a été ensuite reprise et améliorée afin de débuter une analyse des données.

Visualisation des données

La première étape fut de chercher à visualiser les données de la montre sous une forme plus facile à comprendre que 3 courbes (3 axes de l'accéléromètre).
Une implémentation naïve fut de tracer dans un espace 3D, une courbe ayant pour points les points 3D des 3 axes de l'accéléromètre. L'application était fonctionnelle mais la courbe n'avait aucun sens, les données de l'accéléromètre correspondant à une mesure de l'accélération et non de la position de la montre.

Accélération ?

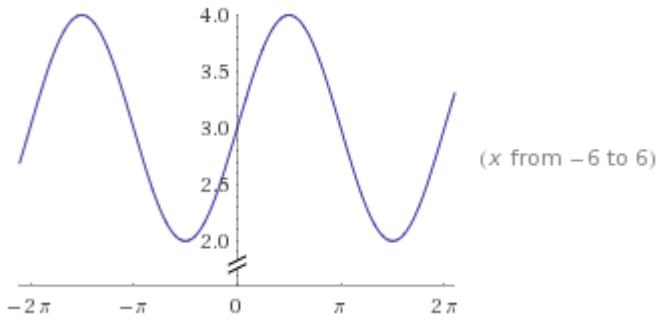
Le but était alors de comprendre réellement à quoi correspondaient les données de la montre.
L'accéléromètre renvoie des données sur 3 axes : X, Y et Z. Ces données sont des entiers, avec une précision de 256 niveaux (-127 à +128).
Pour chaque axe, l'entier correspond à une quantité d'accélération (positive ou négative).
L'accélération correspondant à une dérivée de la vitesse et la vitesse étant une dérivée de la position, une idée fut alors de calculer la position de la montre à partir de des données d'accélération, en intégrant les courbes captées.

Vitesse et position ?

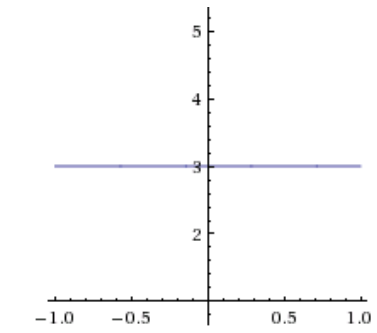
En essayant d'intégrer les données de l'accélération pour obtenir une vitesse et une position pour chaque axe, nous avons obtenu des résultats incorrects.
Il y avait trop d'erreurs, d'une part à cause de la faible précision de l'accéléromètre et du bruit.
C'est alors que nous nous sommes rendu compte que la force de gravité (accélération constante sur Terre) était incluse dans les données renvoyées par la montre. Nous avons dû trouver un moyen de séparer l'accélération réelle (mouvements de la montre) de la gravité (toujours constante).

Filtrage des données

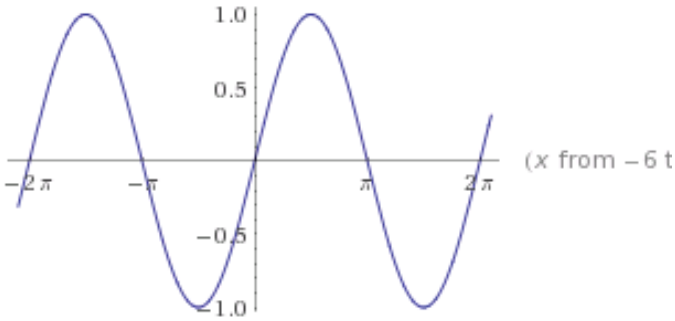
Nous sommes partis du constat que l'implication de l'accélération réelle dans les données de la montre représente des mouvements rapides. L'implication de la gravité, elle, est plus lente et s'explique à cause de la rotation de la montre (basculement de la force de gravité sur les différents axes de l'accéléromètre).
À partir de ce constat, nous avons tenté de filtrer le signal de la montre en reliant les notions de rapidité du mouvement et de fréquence. Ainsi nous pouvons séparer les deux composantes du signal à l'aide d'un filtre dit « passe-bas », qui ne garde que les fréquences les plus basses et donc la composante de la gravité. Un filtre « passe-haut » peut également ne retenir que la composante d'accélération réelle mais présente des inconvénients car peut également amplifier du bruit.
La meilleure option pour récupérer la composante d'accélération fut alors de simplement soustraire la composante gravitationnelle au signal d'origine pour obtenir ce que nous cherchions.



1. **Exemple d'un signal à deux composantes additionnées :** une continue et une variable.



2. **Signal continu (composante gravitationnelle) obtenu à l'aide d'un filtre « passe bas »**



3. **Composante d'accélération obtenue en soustrayant le signal** 2. **au signal** 1.

Résultats finaux de la recherche

Au final, même avec filtrage, il ne nous a pas semblé possible ou même réaliste d'obtenir une position ou même une vitesse à partir des données de l'accéléromètre de la montre. En effet, avec d'une part la faible précision et d'une autre le bruit, une double intégration des données nous a toujours donné des résultats incohérents et inexploitables.
De plus, avec un accéléromètre à 3 axes sans gyroscope (module donnant un angle d'orientation par rapport à l'axe magnétique de la Terre), on ne peut déterminer que 2 axes de rotations sur 3. Ce qui signifie que dans tous les cas, on ne saura jamais dans quelle rotation exacte se trouve la montre.
De tous ces problèmes, il ne nous a pas paru judicieux de continuer dans la direction du mouvement exact de la montre et de l'analyse de la trajectoire.
Cependant, le travail de recherche a fourni des éléments qui nous ont beaucoup servis pour l'analyse de courbes, notamment la partie filtrage de données.

Développement d'un jeu 3D

Environnement de développement

La décision du langage C# et d'un jeu 3D a été décidée assez tôt. Il nous fallait alors trouver une technologie nous permettant d'accomplir ces objectifs.
Un premier pas fut d'essayer la 3D « basique », c'est-à-dire en définissant polygone par polygone, triangle par triangle… Sans même parler de texture, de lumière ou de caméra. Après avoir réussi à tracer quelques courbes de cette manière, nous nous sommes vites aperçus que c'était trop difficile et sans vrai intérêt.
Nous nous sommes alors penchés sur un moteur 3D, spécialisé pour les jeux vidéo, et nous avons trouvé le Framework XNA. Ce dernier, même si limité, nous permet une vitesse de développement assez rapide ainsi qu'une grande communauté. L'idée était de pouvoir profiter de ressources préexistantes pour les améliorer et en faire un jeu complet.
La technologie correspondait bien à ce que nous cherchions et nous avons repris un exemple de projet permettant l'animation d'un personnage.

De l'exemple au jeu

L'exemple repris ne fournissant qu'un personnage muni d'une animation, nous avons dû l'étendre pour en faire un jeu complet : ajout d'éléments graphiques (terrain de tennis, balle, raquette…), physiques (physique de la balle, collisions…), et de jeu (règles du jeu, score…).
Il se trouve que XNA encourage la modification de tels projets et rend même le principe assez simple. Ainsi, en se procurant des modèles 3D et textures libres de droits sur des sites spécialisés, nous avons pu créer un environnement de jeu 3D complet qui correspondait à nos besoins.

Le projet d'exemple « SkinningSample » fournit un personnage avec une animation de marche sur un fond bleu.

Eléments du jeu

Même en ajoutant des éléments graphiques (terrain, stade, filet…), il reste des principes à prendre en compte pour faire un jeu correct.
Les personnages se déplacent automatiquement (joueur et adversaire), la balle rebondit et on peut frapper la balle au bon moment. Autant d'éléments qui semblent évidents une fois le jeu terminés mais qui ont demandé un réel effort sur le moment.

Physique de la balle

La balle de tennis est capable de rebondir au sol en perdant de la force, de toucher le filet en perdant encore plus de force et d'être frappée par les personnages pour la renvoyer.
La physique est gérée principalement par deux vecteurs, de position et de vitesse. Les rebonds se font grâce à des opérations sur le vecteur de vitesse et à des contrôles sur le vecteur de position.
La balle dispose également d'une vitesse angulaire correspondant à sa vitesse linéaire, celle-ci est simplifiée mais fait tout de même appel à des notions de trigonométrie pour avoir une rotation réaliste.
La balle étant le seul élément physique du jeu, elle gère sa physique elle-même et envoie des événements au jeu pour par exemple signifier qu'elle a touché le filet.

Mouvement des personnages

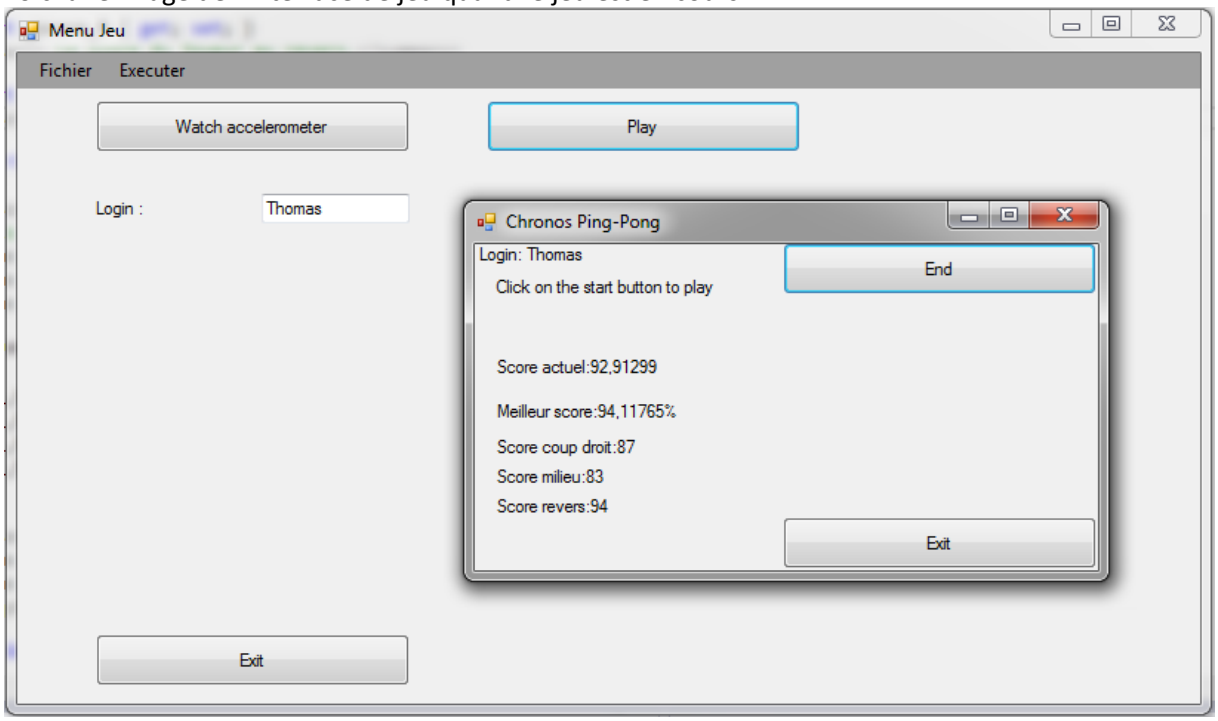
Le premier intérêt de l'exemple « SkinningSample » était l'animation de personnages 3D. Nous avons repris l'animation pour faire marcher (plus ou moins vite) les deux joueurs le long du terrain pour suivre la balle.
Les mouvements sont automatiques et aucun des deux joueurs ne se fera distancer par la balle, seule une erreur de réflexe du joueur principal peut donner suite à une défaite.
En faisant correspondre exactement les coordonnées de la balle (sur un axe) à celle des joueurs (sur le même axe), les mouvements semblaient trop mécaniques.
Pour ajouter un peu de réalisme, les mouvements disposent d'un système les rendant plus fluides. En effet, les personnages auront tendance à augmenter leur vitesse seulement quand la balle s'approche d'eux. Quand la balle est loin, ils économisent leur force en attendant de voir où celle-ci se dirigera.
Ce système permet de détacher les deux personnages en leur donnant des mouvements différents.

Règles du jeu

Les règles du jeu sont implémentées à l'aide d'un système d'événements. Ce dernier permet de recevoir des données analysées (pourcentage de précision, direction de renvoi de la balle…) ainsi que des événements de jeu (filet touché, balle ratée…).
Au final, le joueur peut renvoyer la balle au bon moment en faisant un mouvement avec la montre. Il peut renvoyer la balle dans trois directions différentes et un pourcentage de précision de son mouvement est affiché. Son temps de réaction est également évalué ainsi que son nombre de rebonds à la suite. Le but étant de s'améliorer dans chacun des indicateurs différents.
Le jeu commence à l'appui d'une touche (T) où l'adversaire lui sert la balle, les deux personnages peuvent ensuite s'échanger des rebonds jusqu'au moment où le joueur rate une balle et peut alors choisir de recommencer ou de quitter le jeu.

Les différents menus de jeu

Lorsque le jeu se lance, un menu apparait, d'où il est possible d'ouvrir plusieurs autres menus par de simples cliques.
En cliquant sur « Watch accelerometer », il est possible (une fois la montre connectée) de visualiser les données brutes (l'accéléromètre) de la montre comme expliqué ci-dessus.
Il est aussi possible de se logger avec un nom de login, afin d'avoir un score propre à votre login. Une fois cela effectué, en cliquant sur Play, vous aurez accès à votre page perso avec votre score maximal, et le score actuel que vous ferez en temps réel dans le jeu.
En cliquant sur Start, la partie commence.
Voici une image de l'interface de jeu quand le jeu est en cours.



Comme on peut le voir ci-dessus, une partie est en cours, le jeu calcul les scores du joueur, et les affiches de façon à avoir son score actuel (coup par coup), son meilleur score (tout mouvements confondus), et les meilleurs scores par mouvements.

Partie d'Analyse

But Recherché :

Nous avons ici voulu comparer deux mouvements, afin de voir la proximité entre ces deux mouvements, et si possible arriver à détecter sur quelle phase ceux-ci divergent (détection d'anomalie).
Idéalement, il devrait être possible de visualiser son mouvement, le mouvement idéal, et de voir comment rectifier son geste pour coller au mouvement parfait.

Problèmes rencontrés :

Du début à la fin, de nombreux problèmes sont survenus, rendant très compliqué d'arriver à ce que nous voulions initialement :

- La montre ne renvoie pas les positions, mais seulement l'accélération dans les trois directions de l'espace.
- La montre ne peut envoyer au maximum que un point toute les 20ms, ce qui n'est pas suffisant pour détecter le mouvement puis avoir assez de points pour faire une comparaison phase de mouvement par phase de mouvement.
- Des rotations de la montre entrainaient une énorme variation de l'accélération, rendant les données reçues inexploitables sur certains mouvements.

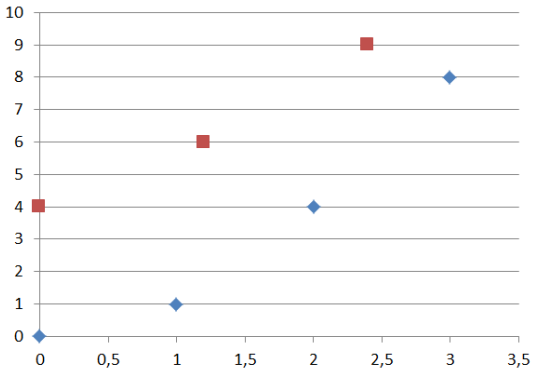
Classes et fonctionnement :

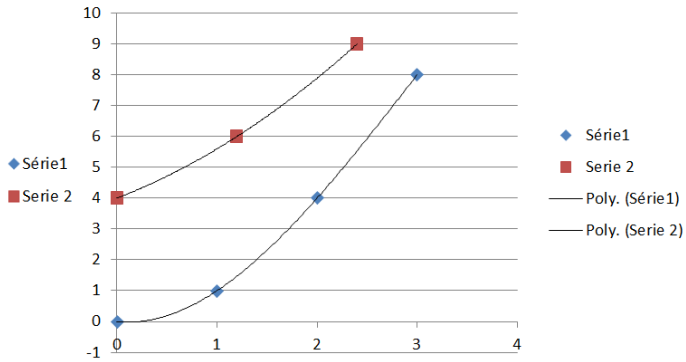
La classe **Analyseur** :
Cette classe reçoit l'accélération en X, Y et Z, et a pour but de détecter lorsqu'un mouvement s'initie afin de créer une trajectoire correspondant à ce mouvement.
Son fonctionnement est au final assez simple. Il y a une `VITESSE_SEUIL` (vitesse à partir de laquelle on considère que c'est un mouvement et non pas un simple déplacement de montre), `NOMBRE_AFF_REUSSITE` qui est le nombre de point d'affilé ayant une vitesse supérieur à la `VITESSE_SEUIL` pour que l'on considère être dans un mouvement. De même, `NOMBRE_AFF_ECHEC` qui est le nombre de point d'affilé ayant une vitesse sous `VITESSE_SEUIL` pour que l'on arrête le mouvement.
Tous les points sont stockés dans une liste de points, et toutes les trajectoires reconnues sont stockées dans une liste.
La variable `_tempsIni` correspond au temps initial auquel l'analyse débute.
Le `_tempsEchantillonage` est la durée séparant la réception de données.
Un booléen, `_isRecording`, qui est à true lorsque nous sommes dans un mouvement et à false sinon.

La classe **Trajectoire** :
Cette classe modélise une trajectoire. Elle contient une liste de points3D(`_points`), un temps d'échantillonnage(`_tempsEchantillonage`), une vitesse seuil(`_vitesseSeuil`), et le temps auquel cette trajectoire a commencé(`_tempsIni`).

Les classes **Interpolateur**, **InterpolateurLineaire** et **InterpolateurLagrange** :
Cette classe permet d'interpoler une trajectoire (qui contient seulement quelques points) en un point donné afin de pouvoir comparer deux courbes qui n'ont pas forcément les mêmes données et la même précision.

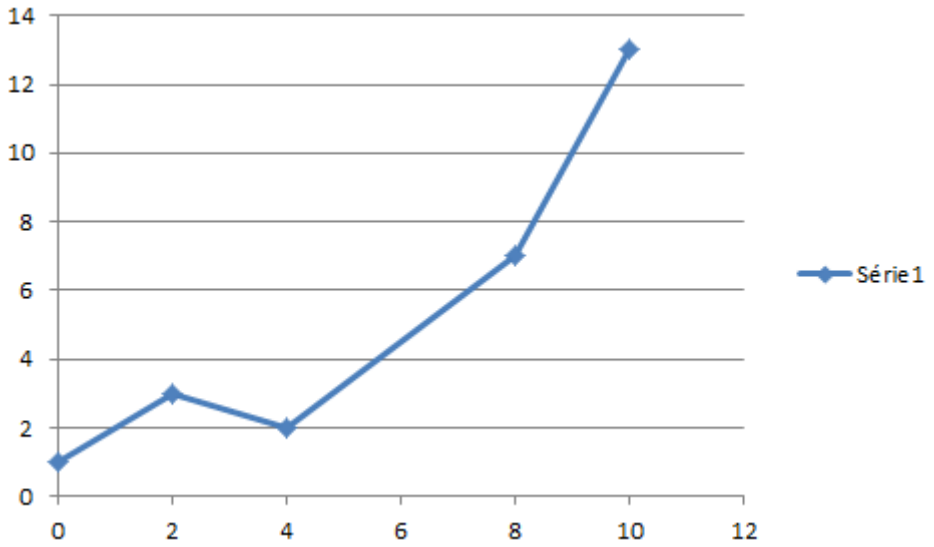
Pourquoi a-t-on besoin d'interpoler nos trajectoires ? Sans interpolateur, il nous est impossible de comparer une grande partie des trajectoires, pour le peu que celles-ci n'aient pas le même nombre de point ou le même temps d'échantillonnage.





On voit qu'ici, sans interpolation, il serait très compliqué de comparer les deux séries de points (qui, dans notre cas seraient des trajectoires), alors que dans le second, il suffit de prendre un nombre de point donné (une dizaine par exemple) équitablement réparti et de comparer les valeurs point par point.

Deux méthodes d'interpolation ont été implémentées :
Un **interpolateur linéaire** et un **interpolateur polynomial** utilisant la méthode de **Lagrange.**
Voici ce que renvoi l'interpolateur linéaire :



Le fonctionnement est simple : on trace des droites entre chaque point consécutif de l'échantillon. Un des désavantages majeur de cette méthode est de ne pas être capable de donner des valeurs en dehors de l'intervalle de point.

Concernant l'interpolateur de Lagrange, cela revient à interpoler une série de points par un polynôme d'ordre nombre_de_points-1.
Ainsi (1, 1) sera approximé par f(x)=1 ,
(0,0) (2,5) (4,8) sera approximé par f(x)= -0.25 X^2 + 3 X

D'un point de vue purement programmation, ces classes n'ont qu'une méthode à implémenter qui est `getValueInterpolee`, prenant en paramètre le temps auquel nous voulons avoir la valeur et retourne le point correspondant (retourne un point3D, c'est-à-dire un triplé (x,y,z)).

Les classes **Comparateur** et **ComparateurSimple** :
Ces classes ont pour but de comparer deux trajectoires, ou plus précisément deux courbes de trajectoires interpolées (laissant libre en amont de choisir la méthode d'interpolation voulue).
Un comparateur possède donc deux interpolateurs `inter1` et `inter2` , qui sont des interpolations des trajectoires à comparer, ainsi qu'un nombre de points à interpoler `nombrePoints`.
La seule méthode que doit implémenter un comparateur est `compare` qui renvoi un float étant le pourcentage de proximité entre deux interpolations de trajectoire.
Dans le cas du `ComparateurSimple` il est assez basique, mais c'est tout de même un moyen simple et assez rapide de comparer deux trajectoires.
Point par point, il calcule la moyenne des différences relatives à la différence maximale (`MAX_VALUE`) dans les trois dimensions spatiales, et garde toutes ces valeurs dans une liste.
La méthode renvoi la moyenne des valeurs contenues dans cette liste.

La base de Donnée

La classe DataBase:
Cette classe permet de stocker des mouvements préenregistrés. En effet, nous avons enregistré plusieurs types de mouvements, comme le coup droit ou le revers, afin de pouvoir comparer le mouvement du joueur a ces mouvements là. Ainsi, lorsque le joueur effectue un mouvement, nous savons s'il s'agit d'un coup dirigé vers la droite, vers la gauche ou vers le milieu du terrain.

Conclusion

Le but du projet, que nous avons défini environ une semaine après le début du projet était un jeu de tennis 3D utilisant les comparaison de courbes et permettant de voir la proximité avec un mouvement parfait, et nous pouvons dire que celui-là a été atteint.
Dans les évolutions possibles, nous pourrions mettre en place l'algorithme de random forest pour améliorer la comparaison de courbe, d'implémenter les « vrais » règles du tennis (service, comptage de point, out…) ou encore embarquer le jeu dans un site web permettant d'opposer deux joueurs.
Mais une amélioration qui serait assez simple et qui apporterai beaucoup de liberté, serait de faire grossir la base de données et de trajectoire de balle possible, rendant possible par exemple des coupés, des smash ou des slices.