Rapport intermédiaire

Table des matières

[Problématique : 3](#_Toc11414781)

[Cahier des charges : 3](#_Toc11414782)

[Introduction 4](#_Toc11414783)

[Analyse des technologies exploitables 4](#_Toc11414784)

[Les détecteurs de mouvements 4](#_Toc11414785)

[La Kinect : 4](#_Toc11414786)

[Le protocole FTMS 5](#_Toc11414787)

[Shimmer3 IMU 6](#_Toc11414788)

[La Wiimote 7](#_Toc11414789)

[Joy-con 8](#_Toc11414790)

[Phidgets 8](#_Toc11414791)

[Les outils de création de jeux 9](#_Toc11414792)

[Unity 9](#_Toc11414793)

[C++ 9](#_Toc11414794)

[Java 9](#_Toc11414795)

[Autres 9](#_Toc11414796)

[Bluetooth 9](#_Toc11414797)

[Moteur 3D de Marc Hugi et Christian Ammann 10](#_Toc11414798)

[Choix des technologies 10](#_Toc11414799)

[Concepts de prototype 11](#_Toc11414800)

[Contraintes & démarche 11](#_Toc11414801)

[Concept du jeu 13](#_Toc11414802)

[Jeu solo 14](#_Toc11414803)

[Objectifs 15](#_Toc11414804)

[Jeu multi 15](#_Toc11414805)

[Serveurs mondiaux 17](#_Toc11414806)

[La concurrence 19](#_Toc11414807)

[Kinomap 19](#_Toc11414808)

[Run social 20](#_Toc11414809)

[IConnect 20](#_Toc11414810)

[Exercube 20](#_Toc11414811)

[Zwift 21](#_Toc11414812)

[Arcade Fitness 22](#_Toc11414813)

[Fitness gaming 22](#_Toc11414814)

[Post analyse du concept 23](#_Toc11414815)

[Conception & implémentation du prototype 23](#_Toc11414816)

[Boucles de jeux 23](#_Toc11414817)

[Méthode naïve 23](#_Toc11414818)

[Boucle avec délayage 24](#_Toc11414819)

[Boucle avec étape à taille variable 24](#_Toc11414820)

[Boucle à délayage sur la logique 25](#_Toc11414821)

[Scrolling vertical infini 26](#_Toc11414822)

[Objets et collisions 26](#_Toc11414823)

[Méthode naïve 26](#_Toc11414824)

[Broad Phase 27](#_Toc11414825)

[Résolution de collision 28](#_Toc11414826)

[Rochers & Cœurs 28](#_Toc11414827)

[Boucliers 29](#_Toc11414828)

[Vitesse de scrolling 29](#_Toc11414829)

[Mesureurs d’efforts 30](#_Toc11414830)

[EffortCalculator 30](#_Toc11414831)

[Mesureur souris 31](#_Toc11414832)

[Mesureur d’accélération 32](#_Toc11414833)

[Mesureur de fréquence 33](#_Toc11414834)

[Mesureur de vitesse de course 35](#_Toc11414835)

[Déplacements horizontaux 38](#_Toc11414836)

[Clavier 38](#_Toc11414837)

[Wiimote 38](#_Toc11414838)

[Joy-con 39](#_Toc11414839)

[Phidgets 39](#_Toc11414840)

[État actuel 39](#_Toc11414841)

[Prototype 39](#_Toc11414842)

[Mesureur d’effort 40](#_Toc11414843)

[Objectifs restants 40](#_Toc11414844)

[Mesureur pour tapis de course 40](#_Toc11414845)

[Mesureur pour vélo 40](#_Toc11414846)

[Mesureur pour rameur 40](#_Toc11414847)

[Machine Learning 40](#_Toc11414848)

[Transfert de connaissance 41](#_Toc11414849)

[Bibliographie 41](#_Toc11414850)

# Problématique :

Avec la croissance de la demande en sport, l'accroissement des villes, la multiplication des mégalopoles et donc l'augmentation de la pollution.

La start-up 4πR2 souhaite offrir un nouveau type de structure pour encourager le sport en ville sans pour autant respirer dans les pots d'échappement. Elle se différenciera des divers fitness et centre sportif grâce à sa facilité d'accès (positionnel et pécuniaire).

Les objectifs sont de promouvoir le sport, produire de l'énergie renouvelable et améliorer la santé ? **générale** de la population.

Concrètement, la start-up souhaite installer en ville des petits dômes dans lesquels 5 postes seront disponible pour y faire du sport. L'air de ces « bulles d'énergies » seraient purifié ? et toute l'énergie pour les alimenter viendrait des sportifs. Finalement, afin de motiver les gens à faire du sport, un jeu vidéo sera disponible sur chaque poste. La difficulté de ce dernier est de réussir à créer un jeu qui pousse les sportifs en dehors de leurs zones de confort sans pour autant les dégoûter.

C'est dans ce cadre que lors de ce travail de Bachelor, nous mettons en place la création d'un jeu adaptable au sportif.

# Cahier des charges :

Ce projet de Bachelor se déroulera comme suit :

1. Analyser les technologies possibles pour la création d'un jeu synchronisé en temps réel à une séance de sport.
2. Imaginer et/ou choisir un jeu utilisable pour tester et calibrer le système adaptatif au sportif
3. Implémenter un prototype du système et l'adapter à l'expérience utilisateur
4. Ajouter une couche de machine Learning au prototype pour qu'il s'adapte à l'utilisateur

Il s'agira avant tout d'analyser les diverses technologies qu'il est possible d'utiliser pour gamifier une séance de sport. Ainsi que d'analyser les produits existant déjà sur le marché afin de se placer comme complément au offres existantes.

Il faudra ensuite trouver ou imaginer un jeu qui résout la problématique, qui soit innovant mais suffisamment simple afin de pouvoir rapidement obtenir un prototype qui servira à tester et calibrer la partie machine Learning du projet. C'est à dire la capacité à rendre le jeu capable d'être au service du sport et de l'amusement quel que soit le niveau du sportif.

Lorsque l'idée de jeu sera acceptée par le mandataire et le responsable, le début de l'implémentation commencera. Puisque le projet est grand et qu'il commence à peine, il ne sera pas terminé à la fin de ce travail de Bachelor, il faudra donc mettre un accent important sur la documentation et l'extensibilité de ce premier prototype qui pourrait être reprit par la suite. Le but serait d'avoir à la fin de ce travail de Bachelor la version un joueur fonctionnel et dont la difficulté s'adapterait aux performances de l'utilisateur entre chaque séance.

# Introduction

# Analyse des technologies exploitables

Dans le but de mesurer l’effort et les mouvements du sportif, à travers des signaux, puis de les transformer en impulsions qui seront perçu par un moteur de jeu comme des manipulations de l’état du jeu, le tout afin de rendre un séance sportive récréative. Il a fallu chercher parmi les technologies déjà existantes. Voici la liste que j’ai pu analyser en 3 catégories. Les objets permettant de capturer un mouvement ou une position. Les technologies permettant d’écrire du code pour manipuler ces résultats et créer un prototype de jeu. Et enfin les technologies spécifiques qui viennent d’idées innovantes ou simplement desquels nous dépendront immanquablement.

## Les détecteurs de mouvements

### La Kinect :



Figure 1 Kinect

La Kinect est un appareil servant à manipuler une interface graphique sans utiliser de manette. Pour y parvenir, elle utilise depuis sa version 2 la reconnaissance vocale, la détection de mouvement et la reconnaissance d’image. De nombreux utilisateurs critique sa latence et son manque de précision. Elle est tout de même utilisée par certains jeux. Notamment la série je jeux Just Dance, un jeu de danse où il faut suivre le mouvement des danseurs à l’écran. Cette caméra est utilisable sur XBOX 360, XBOX ONE, Windows 7 et 8. Malheureusement, Microsoft a annoncé l’arrêt de sa production. On retrouve une partie de ces technologies dans le casque de réalité augmenté Hololens et le casque de réalité virtuelle pour Windows 10, mais on perd la qualité principale de n’avoir aucun équipement pour détecter le mouvement. La Kinect 2 à été annoncée sous le nom d’Azure Kinect, elle vise principalement les entreprises. Ocuvera est un des projets réalisés avec. Ocuvera permet de prédire la chute de patients hospitalisé bien avant la chute afin de prévenir les infirmières à temps. Microsoft a annoncé que cette nouvelle caméra pouvait être utilisé dans le cadre des jeux, mais que ce n’était pas l’objectif de la firme qui vise, pour le moment, le domaine médical et robotique. La Kinect est donc un outil vieux et son successeur ne garantit pas de pouvoir l’utiliser dans le domaine des jeux. Sans oublier que nous parlons d’un appareil à environ 400 dollars ou dont la production qui s’est arrêté.



Figure 2 Azure Kinect

### Le protocole FTMS

FTMS ou Fitness machine service est une spécification d’un service Bluetooth qui se démocratise gentiment. Si un appareil de fitness implémente cette spécification, alors il est vu comme un serveur. Il serait donc possible de s’y connecter en tant que client pour lire les données du sportif en temps réel. Plusieurs applications tel que Kinomap utilisent ce protocole pour faire vivre des expériences sportives qui sortent de l’ordinaire. D’après la documentation de cette spécification il est même possible de faire varier les paramètres de la machine (avec une protection sur les changements brusque pour protéger l’utilisateur). L’avantage de cette technologie est qu’elle fonctionnerait sur beaucoup d’appareils, l’inconvénient est que les appareils d’entrée de gamme ne proposent pas de Bluetooth. Et les exercices sans machines, tel qu’avec le poids du corps ou des altères ne pourraient être détectés. Voici une image qui relie la fonctionnalité des machines de fitness aux protocoles Bluetooth pouvant les gérer puis les reliant aux applications qui supportent ces protocoles. Par exemple, le rythme cardiaque (logo noir avec le cœur) est supporté par deux services Bluetooth (HRS et FTMS)

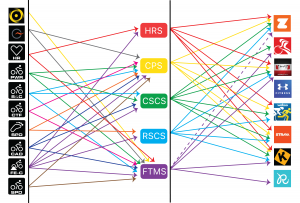


Figure 3 fonctionnalité -> services Bluetooth -> applications

### Shimmer3 IMU



Figure IMU Shimmer3

La Shimmer3 Inertial Movement Unit est un senseur possédant un accéléromètre, un magnétomètre et un gyroscope capable de capturer en temps réel les mouvements de son porteur. L’appareil est léger, configurable et est vendu comme une plateforme largement utilisé comme outil de preuve de concept. L’avantage serait cette fois d’être indépendant de l’appareil sur lequel on fait du sport. De plus, l’institut IICT en possède déjà quelques exemplaires. Le prix étant toutefois élevé (500 euros). Interactive Tango Milonga est un exemple d’utilisation de la Shimmer3, l’application réussit à créer de la musique grâce au mouvement capturé des danseurs de tango, ce qui créé une relation inversée entre les danseurs et leurs musiques.

### La Wiimote



Figure Wiimote

La Wiimote est la manette proposée par Nintendo pour sa console Wii en 2006. Elle fonctionne via un accéléromètre ainsi qu’une barre fluorescente avec dix DEL infrarouge. On peut se servir de la manette sans la barre. De plus, la console ayant eu beaucoup de succès, de nombreuses librairies ont vu le jour pour utiliser la manette comme détecteur de mouvement. Avec notamment WiiGee, motej et pypi. Un inconvénient est qu’il faut la tenir en main, ce qui pour un tapis de course n’est pas gênant, pour un vélo serait possible, mais pour un rameur serait un problème. Malheureusement cette manette datant, elle n’est plus en production, mais vu la quantité ayant été écoulé, il est encore très simple de la trouver. La manette Wii motionPlus ressemblant très fidèlement à la Wiimote rajoute un gyroscope à la manette ce qui ajoute de la précision au capteur. Le gros avantage de cette manette est son prix (on peut trouver des manettes sur internet à 14 euros).

### Joy-con



Figure Joy-con

Les Joy-con sont les successeurs actuels de la Wiimote. Ils sont utilisés par Nintendo pour leur console la Switch et possède un accéléromètre, un gyroscope et un dispositif haptique (qui permet de faire croire à des sensations tactiles d’un monde virtuel). Plus petit, plus léger que la Wiimote, il serait même concevable de leurs créer une attache comme pour la Shimmer3. Tout en étant beaucoup moins cher que le Shimmer3 et spécifiquement conçu pour la ludification d’applications. Le seul bémol étant la jeunesse de cette technologie. Ce qui en fait une possibilité peu documentée et dont les librairies sont encore très sommaires. Avec la Wiimote, nous avions des librairies au niveau d’abstraction du mouvement, tandis qu’avec les Joy-con, nous en sommes encore à réussir à communiquer avec, grâce à l’ingénierie inversée. Par exemple, le repos <https://github.com/elgoupil/joyconLib> prend en charge les boutons et les joysticks mais pas l’accéléromètre. Cependant, il serait possible de l’améliorer avec les informations sur le protocole des Joy-con regroupé dans le repos <https://github.com/dekuNukem/Nintendo_Switch_Reverse_Engineering/blob/master/imu_sensor_notes.md>. On peut acheter en magasin une paire de Joy-con pour 80.-.

## Phidgets



Les Phidgets sont une collection de senseurs pour absolument tout (température, motion, etc.). Le but de l’entreprise qui les produit est d’offrir des senseurs et contrôleurs à petit prix qui requiert un minimum de connaissance en électronique. La documentation est fournie, les OS sont tous supportés et les langages compatible sont nombreux (C, C#, Python, Java, JS, LabView, VB .NET). Ils sont compatibles avec Unity. Il serait donc intéressant de coupler ces deux technologies pour sortir un jeu basé sur le mouvement. On retiendra notamment le phidget accéléromètre qui permet de mesurer jusqu’à 8g et est fait pour les applications mesurant le mouvement pour seulement 20$. Tandis que le senseur de force permettrait d’avoir un bouton pour signifier le début et la fin du mouvement pour seulement 8$. L’inconvénient est qu’ils sont câblés. Et donc limiterait le mouvement du sportif et nuirait au confort.

## Les outils de création de jeux

### Unity



Il existe beaucoup de moteur de création de jeux, J’ai retenu Unity pour son modèle financier. Gratuit si le revenu annuel ne dépasse pas 100000.- par année, 125.- par mois pour les professionnels dépassant ce revenu. Tandis qu’Unreal Engine propose 5% sur les gains des créations. Unity supporte la 2D, la 3D, les jeux réseaux et permet de déployer sur beaucoup de plateforme. Mais son plus grand atout est sa popularité. En conséquence, il existe des tutoriels pour approximativement tout. Le langage supporté par Unity est C# qui est relativement simple à apprendre. Comme avis personnel, Unity est un incontournable pour la création de jeux et est probablement le choix que j’utiliserais si le projet de ce TB dépasse le stade de la R&D. Néanmoins, cet outil étant très puissant et flexible, il demande un temps d’apprentissage non négligeable. Ce qui est le cas pour probablement tous les moteurs de jeux qui dépasse un certain stade de flexibilité.

### C++

Lorsqu’on créé des jeux vidéo, un point important est la réactivité de l’interface. Tous les calculs doivent être fait aussi efficacement que possible. C++ étant un langage compilé, offre la performance requise pour ce type d’application. En plus, d’être compilé, Il n’existe pas de garbage collector, ce qui signifie que c’est au codeur de choisir quand un objet est détruit ou construit, mais surtout, en plus de ralentir potentiellement le système à cause de la routine du garbage collector, il est aussi possible d’avoir des glitches et des comportements non contrôlés à cause de ce même garbage collector. Le dernier avantage est la maturité de cette technologie, et donc sa grande communauté et documentation, qui permet de résoudre absolument n’importe quels soucis. Le seul désavantage est la difficulté du langage et donc la rapidité à écrire du code en est affecté.

### Java

Java garde l’avantage d’être un langage compilé, cependant il possède un garbage collector et donc tous les points négatifs que l’on vient de citer avec C++. Java sera forcément plus lent que C++ puisque orienté objet. Mais sa facilité d’écriture et mon expérience personnel avec le langage sur plusieurs projets font de lui ma préférence actuelle surtout pour un travail de recherche.

## Autres

### Bluetooth

Tous les senseurs cités précédemment communique en Bluetooth. Il est donc indispensable de connaître les bases de Bluetooth avant de se lancer dans ce projet. Bluetooth utilise les mêmes fréquences que Wi-Fi et donc des potentiels interférences peuvent survenir. Dans une telle situation, le Bluetooth fonctionnant à plus basse puissance que le Wi-Fi, c’est probablement lui qui sera le plus impacté. Plusieurs techniques sont utilisées par les OS pour gérer les interférences. Sous linux par exemple, une fréquence adaptative ou passer les channels déjà occupé par le Wi-Fi. Depuis la version 5 un gros effort à été fait pour diminuer les risques d’interférence, mais il reste important de garder ce potentiel problème en tête. Particulièrement si le prototype impliquera plusieurs sportifs simultanés à longue distances. Une autre limite à connaître sont les 200m de portée maximal (qui diminuera avec chaque source de bruit/obstacle). Le débit varie en fonction de la version utilisé mais tourne autour des 2Mbit/s.

### Moteur 3D de Marc Hugi et Christian Ammann

Durant l’Exposition d’Einstein de 2006 à Berne, 3 vélos avaient été mis à disposition du public. Chaque vélo était face à un écran avec une image 3D fixe. Plus l’utilisateur pédalait vite, plus l’image était déformée afin de simuler une approche de la vitesse de la lumière. Le Moteur 3D avait été écrit par Marc Hugo et Christian Ammann. C’est le type de technologie qui pourrait être utile pour donner une impression de vitesse au joueur. Seulement, la technologie est vieille comme on peut le voir sur l’image qui suit. Peut-être y aurait-il moyen de recontacter les développeurs d’origine pour une collaboration futur. Je le note dans ce rapport à titre informatif. Mr. Marc Hugi travaille actuellement à Attractiv Solution AG, tandis de Mr. Amman est à Giants Software GmbH.

## Choix des technologies

Au début du projet, nous avions souhaité utiliser le protocole FTMS afin de pouvoir mesurer l’effort du sportif directement avec les machines qu’il utilise. Cependant nous ne possédions aucun appareil de fitness accessible pour tester cette approche. Notre mandataire à même commencé à chercher de tels appareils et nous sommes toujours en attente. Après ces quelques analyses, nous avons décidé d’utiliser comme senseurs la Shimmer3, qui rentre exactement dans la cadre de ce que nous souhaitons faire, soit un outil pour une preuve de concept d’une application kinesthésique. De plus, l’institut IICT possède déjà cet appareil et donc aucun frais supplémentaires ne sera généré. À postériori, nous sommes plutôt satisfaits de l’indépendance de la shimmer3 aux appareils de fitness pour ainsi pouvoir potentiellement s’entraîner simplement avec le poids du corps. Nous allons donc continuer ce projet avec Shimmer3, mais il serait intéressant qu’un second projet soit dédié pour faire le même travail avec FTMS.

Pour la création du code, nous avons décidé d’utiliser Java. Le code qui sera produit, n’est pas le code final du projet et donc utiliser un langage dans lequel je suis à l’aise, nous permettra d’avancer plus vite et donc d’explorer plus de pistes. Le tout sans pour autant nuire à la performance du produit final qui à mon avis devrait être écrit par un langage plus optimisé (C++/C) ou un outil dédié tel que Unity. Voir même au contraire, avoir un langage moins optimisé permettra de plus facilement mettre en évidence les goulots d’étranglements.

Plus tard dans le projet, il s’est avéré que nous avions besoin d’un accéléromètre avec un bouton pour signifier le début et la fin du mouvement. La shimmer3 ne possédant pas ce bouton, il a fallu réfléchir à utiliser un autre accéléromètre. Nous avions choisi la Wiimote pour sa bonne prise en main, sa bonne documentation et la maturité de ces librairies. Cependant, des soucis que j’expliquerais plus tard dans ce rapport, nous ont fait adopter les Joy-con.

# Concepts de prototype

En parallèle à la recherche sur les technologies utilisables, nous avons dû réfléchir à un concept de jeu rendant une séance de sport ludique. Ce chapitre explique quels ont été les contraintes et les objectifs qui nous ont fait parvenir à un choix. Puis explique le concept final (trop ambitieux pour un travail de Bachelor seul) et quels sont les limites que nous espérons atteindre. Sans oublier qu’il a fallu faire valider ce prototype autant par le professeur responsable Mr Perez-Uribe, que par le mandataire Mr Grégory Vincent.

## Contraintes & démarche

La plus grande contrainte à été de restreindre au maximum le nombre de manipulation possible. Car afin de maximiser l’entrainement, le sportif doit tout de même rester concentrer sur son activité physique. Ce qui va complètement à l’encontre d’un jeu qui lui est là pour distraire. La plupart des jeux que l’on voit émerger sous forme de réalité virtuelle sont tous extrêmement immersif. A tel point qu’on en oublie notre environnement. Ici, ce n’est pas notre objectif. Nous souhaitons que le jeu serve le sport et que le sport active le jeu. Diminuer le nombre d’interaction possible, diminue en même temps la distraction.

Malheureusement le gain pour le sportif à ne pas être distrait implique un effet négatif pour le joueur, car il supprime un des 4 éléments fondamentaux du jeu. En effet un jeu possède 4 caractéristiques pour être reconnu en tant que tel : les mécaniques, l’histoire, l’esthétique et la technologie. En diminuant les interactions possibles, nous diminuons les possibilités de mécaniques. Or lorsqu’on regarde la concurrence, aucune n’a fait de compromis entre manipulation restreinte et mécanique élaborée. Soit, nous nous retrouvons avec un jeu très amusant complètement immersif. Soit, les applications ne sont pas des jeux, mais un simple moniteur ou film sans aucune interaction ou très peu. La grande difficulté de conception que nous avons aujourd’hui est donc de réussir à trouver un compromis. Qui possède suffisamment peu de distractions/manipulation pour servir à un sportif et améliorer ces performances. Mais assez, pour avoir un jeu digne de ce nom qui amusera et deviendra addictif.

Afin de trouver ce compromis, nous avons listé et étudié les types de jeux existants. Certains pouvaient être éliminé directement car ils demandent une réflexion active trop intense, trop de lecture :

* Les Fiction interactive
* Les Visual novel
* Les jeux de réflexion
* Les Labyrinthes
* Les Puzzle game
* Les jeux de programmation
* Aventure
* Infiltration
* Jeu de rôle
* Stratégie
* God game
* Jeux de Rythme

Beaucoup de jeux possède des déplacement horizontaux ET verticaux. Impliquant directement une forte demande en interaction, uniquement pour les déplacements. Et donc impossible de trouver un compromis avec ce type de jeux :

* Action
* Combat
* Beat them all
* Jeux de tir
* Rogue-like
* plateformer

Pour certains de ces jeux « inutilisable » on pourrait penser à simplifier un aspect. Par exemple pour un jeu de plate-forme 2D on pourrait rendre les sauts automatiques ce qui supprimerait la gestion de la verticalité. Ou alors pour un jeu de rythme on pourrait diminuer la réflexion en automatisant les touches, il ne faudrait plus que penser à rester en rythme.

D’autres type ont été supprimé de la liste pour des raisons évidente comme :

* Le Survival Horror
* Les Simulation de drague

Après ces filtrages et quelques préférences personnel, il ne restait plus que 4 types de jeux :

1. Les jeux de course
2. Les jeux de rythmes simplifiés
3. Un plateformer 2D simplifié
4. Un infinite runner

Les contraintes suivantes nous ont permis de faire un choix, il nous fallait un jeu rapidement afin de pouvoir commencer la partie recherche. L’institut IICT étant spécialisé dans le machine Learning, le cœur du projet se trouve être dans la collecte, l’analyse des données et en dernier l’adaptivité grâce au machine Learning. Donc un jeu simple, soit une version gratuite avec son code source trouvable sur internet, soit quelque chose qu’on puisse rapidement écrire. Nous avons abandonné les jeux de course et le plateformer pour cette raison. Nous avons ensuite trouvé 2 jeux de rythmes avec leurs code sources :

Le premier c’est openitg, un gros projet écrit en C++, malheureusement aucun commit n’a été fait depuis 2 ans. J’ai tout de même essayé de l’installer mais les versions de directX nécessaire sont trop vieilles pour ma machine.

Le second s’appelle dance dance révolution. Un petit projet écrit en python, qui a nécessité de modifier ma version de python, qui ne possède que 2 chansons et qui ne fonctionne même pas sur ma machine lorsque j’appuie sur jouer.

Après ces deux échecs, nous avons trouvé un infinite runner, relativement simple écrit en javascript. Il s’appelle Lava Run Game et consiste à manipuler un robot de gauche à droite pour esquiver des pierres tombant d’un volcan afin de s’envoler du cratère. Une version jouable de l’originale se trouve sur <https://sidf3ar.github.io/Lava-Run-Game/>, voici à quoi ressemble le jeu :



Figure Lava Run Game

C’est sur cette base que nous avons décidé de construire un concept en gardant en tête les contraintes de conception et les contraintes du mandataire soit un jeu qui puisse plaire à tous :

* Un jeu simple, rapidement écrivable en java
* Un jeu avec peu d’interaction
* Un jeu qui puisse être joué seul et étendue à plusieurs
* Un jeu avec divers modes afin que tous type de joueur/sportif puisse s’identifier
* Un jeu qui puisse contenir un business model lié à la dépense du sportif
* Un jeu qui serve la santé et les performances sportive avant le plaisir

## Concept du jeu

Le concept va être de reprendre les mécaniques du Lava Run Game en y ajoutant un mouvement cyclique pour faire défiler le mur verticalement. L’important c’est que n’importe quel type de mouvement cyclique puisse être détecté comme un effort. Si la personne ne bouge pas, l’effort sera à 0, Si la personne se trouve au rythme qui lui est propre et conseillé pour son propre niveau sportif, alors l’effort est à 1. Si la personne dépasse cet effort « optimum » le mur ralentira proportionnellement au surplus. Les rochers tomberont de manière aléatoire à un rythme déterminé. Pour les éviter, il faudra faire un mouvement ou une position prédéterminée. Nous aurons donc en tout 3 interaction possible par le sportif :

1. Faire avancer le mur avec un mouvement cyclique (courir, pédaler…)
2. Faire un mouvement ou une position pour se déplacer à gauche
3. Faire un mouvement ou une position pour se déplacer à droite

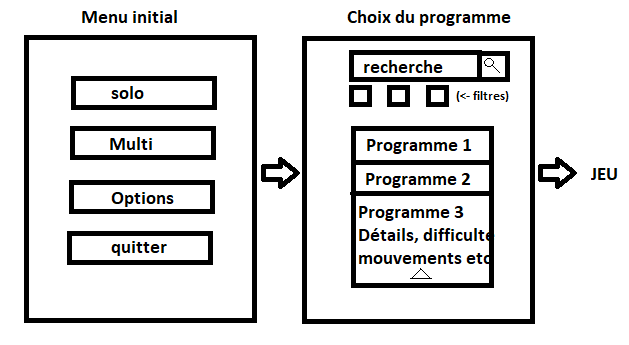
Finalement pour imposer un peu de variété, nous ajouteront des changements d’arrière-plan. Chaque arrière-plan possèdera son propre ensemble de 3 mouvements/positions pour ainsi forcer le sportif à travailler divers muscles pendant une séance. Grâce à ces changements nous pourrons de manière transparente aux utilisateurs, les faire passer par des phase d’échauffement, entraînement et récupération.

En plus, des obstacles, des bonus tomberont du ciel. Un bouclier, permettant de traverser les rochers sans effet. Ainsi qu’un boost de vitesse momentané. Un objectif de distance ou de temps déterminera la fin de partie. Un score relatif à la distance parcouru s’affichera en continu. Et afin de donner de la variété en cours de partie, des objectifs s’afficheront, tel que détruire une quantité de rocher avec un bouclier, éviter une chaine de x rochers en zigzagant. Parcourir une distance de x sans être en dessous de 90% de l’effort « optimal » etc. Ainsi nous auront suffisamment de mécaniques servant le sportif sans lui demander une trop grande concentration.

Qu’en est-il des 3 autres bases du jeu ? Pour l’esthétique, on visera la philosophie du mandataire : « une planète en bonne santé, des gens sportifs, de la technologie verte. Par exemple, les bonus de vitesse pourraient être une bouteille d’eau. Pour la technologie, nous avons vu ce qu’il en est dans le chapitre précédent. Tandis que pour l’Histoire, il serait intéressant d’offrir un scénario à la première connexion des joueurs avec une morale sur la santé et les comportements verts. On pourrait à travers ce scénario à chaque nouvelle connexion offrir un petit conseil pour s’améliorer.

## Jeu solo

Lorsqu’on jouera seul, le but principal sera choisi par le joueur. Il choisira un programme sportif, qui sera composé du mouvement circulaire et des deux mouvements pour aller sur les côtés ainsi que leur fréquence qui influenceront le rythme des chutes de pierre. Il faudra évidement que la sélection du programme soit une recherche paramétrable afin que chacun puisse trouver ce qui lui convient. Que ce soit par âge, par objectifs du programme (maintien du corps, performance, endurance etc.).



On notera que le bouton de création des programmes n’y est pas et devra se faire par des spécialistes/administrateurs via un client différent que les utilisateurs « normaux ». Si toutefois on décide d’ajouter la création de programme, il faudra ajouter une annotation « programme amateur ». Une fois en jeu, en solitaire, le concept ne change absolument pas de ce qui a été décrit précédemment dans concept du jeu.

## Objectifs

Les objectifs seront des petits textes définissant un objectif en cours de partie. Les missions accomplies rapporteront des points de score. Lorsque deux équipes sont en compétition, les objectifs seront partagés et les points reviendront à la première équipe réussissant à la compléter.

Puisque les points seront définis par la distance parcouru principalement, un objectif réussit devra correspondre à quelques secondes d’avance (a priori je dirais 15s d’avance) comme si le(s) sportif(s) avait gardé leur rythme moyen pendant ce laps de temps.

Voici quelques idées d’objectifs possible qui resteront à valider :

* Parcourir x distance
* Détruire x rocher avec un bouclier actif
* Ramasser x bonus de vitesse alors qu’un bonus de vitesse est déjà actif
* Produire x énergie pour la bulle d’énergie
* Éviter x rocher en zigzagant
* Ne pas descendre en dessous de x% de votre rythme optimal pendant y seconde

Il va sans dire que chaque objectif ont des difficultés d’implémentation différente et donc demandera une analyse préalable pour déterminer combien de temps nous sommes prêt à mettre pour créer tel ou tel objectif. Mais plus les objectifs seront variés et nombreux, plus les façons de réagir seront multiples et donc on évitera l’ennui par redondance des mécaniques.

## Jeu multi

Les mode multi vont rajouter une grande complexité à l’implémentation du projet. Cependant, de nos jours c’est une fonctionnalité que l’on considère comme acquise et que les gens souhaitent voir.

Pour le multi, il existe deux types de façons de jouer, la manière coopérative et compétitive. Dans la première les gens s’entraident dans un objectif commun. Ici parcourir la plus grande distance possible. Dans la seconde les gens vont se confronter les uns aux autres afin de mesurer leur performance avec un point de vue extérieur. Les deux types sont complémentaire, nous allons donc proposer deux expériences différentes de multi avec une mécanique supplémentaire pour encore une fois varié et éviter l’ennuie par redondance.

Nous allons placer 5 avatars sur la même partie. Les joueurs seront placés à des étages verticaux différent afin qu’ils puissent bouger de gauche à droite librement sans collision. Enfin, nous allons ajouter un effet d’aérodynamisme, plus les joueurs seront éparpillés horizontalement, plus ils seront lents et inversement, plus les joueurs seront alignés plus ils iront vite. Une image mentale pour se représenter l’idée serait un groupe de cycliste dans une compétition sportive. Voici une image représentant ce système :



Figure 9 coopératif positions rapides

Figure 8 coopératif positions lentes

Pour le mode compétitif, nous aurons deux équipes de 5 joueurs. Mais nous ajoutons un membre de l’équipe au milieu de l’équipe adverse. L’objectif de l’intru sera de perturber l’effet aérodynamique et donc bouger de gauche à droite de manière imprévisible pour ralentir l’équipe dans laquelle il est. Tous les effets de collision que l’intrut ramassera seront inversé. Toucher un rocher, plutôt que de ralentir accélérera. Attraper un boost de vitesse ralentira l’équipe. Voici à quoi pourrait ressembler l’interface :

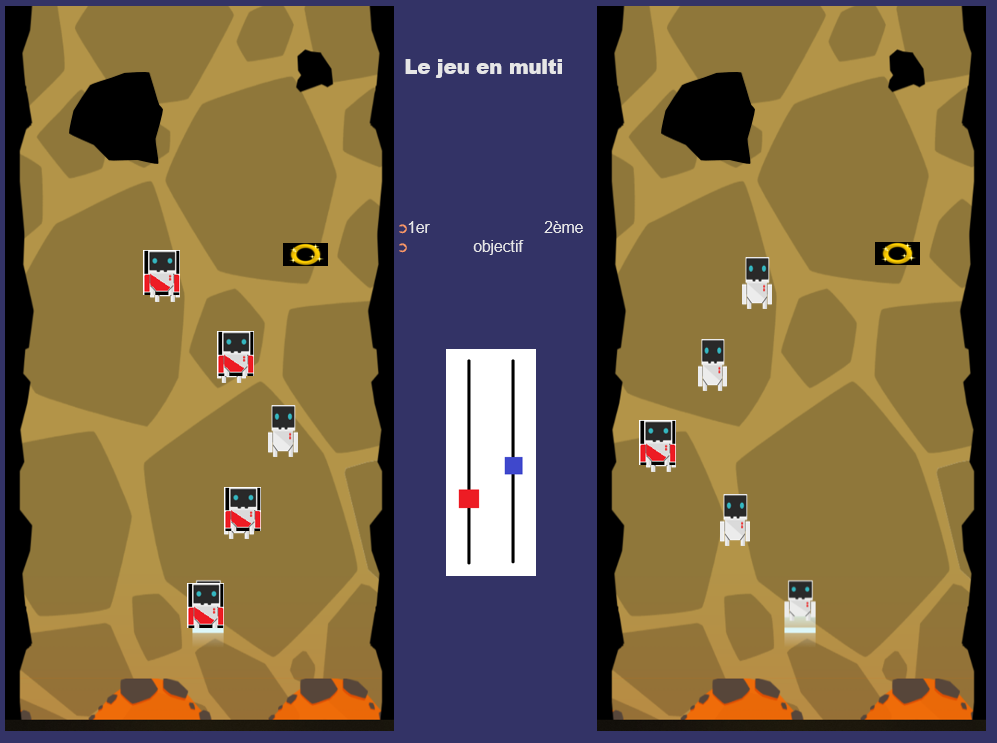


Figure Jeux en compétitif

Lors d’une partie à plusieurs, le moral est un facteur important et la motivation peut dépendre du score actuel. Nous avons donc préféré laisser un affichage 1er – 2ème plutôt que de faire apparaître la position exacte des équipes. Ainsi, les 1ers ne se relâche pas à cause du surplus d’avance et les seconds ne savent pas s’ils sont proche ou loin et peuvent donc toujours garder espoir.

## Serveurs mondiaux

Le but du mandataire est de quadriller les mégalopoles de ces bulles d’énergie. Le prototype puisqu’il pourra offrir une expérience à plusieurs, devra évidement être connecté à travers divers serveurs. Chaque poste de sport serait donc un client. Chaque bulle sportive possèderait un serveur qui serait un hébergeur de partie et pourrait dans le cas limite ou chaque membre d’une bulle héberge une partie compétitive (donc avec le nombre de joueur le plus élevé soit 10) faire participer jusqu’à 50 clients à la fois dans 5 parties. Car je le rappel une bulle sportive est composé de 5 postes sportifs. De plus, chaque serveur devra pouvoir communiquer avec les autres bulles. Le mieux serait d’avoir un dernier type de serveur, qui serait une solution cloud. On pourrait ainsi offrir élasticité et sécurité sans trop d’effort et permettrais d’avoir un système de matchmaking ainsi qu’une base de données commune. C’est aussi sur ces serveurs que le calcul du machine Learning permettra d’affiner l’effort « optimal » à chaque partie pour un client donné. La base de données devra être répliquée pour permettre d’offrir le service sur l’ensemble du globe sans latence et les serveurs globaux devront implémenter des algorithmes de programmation répartie afin d’éviter les incohérences. Voici un résumé de cette hiérarchie en illustré :

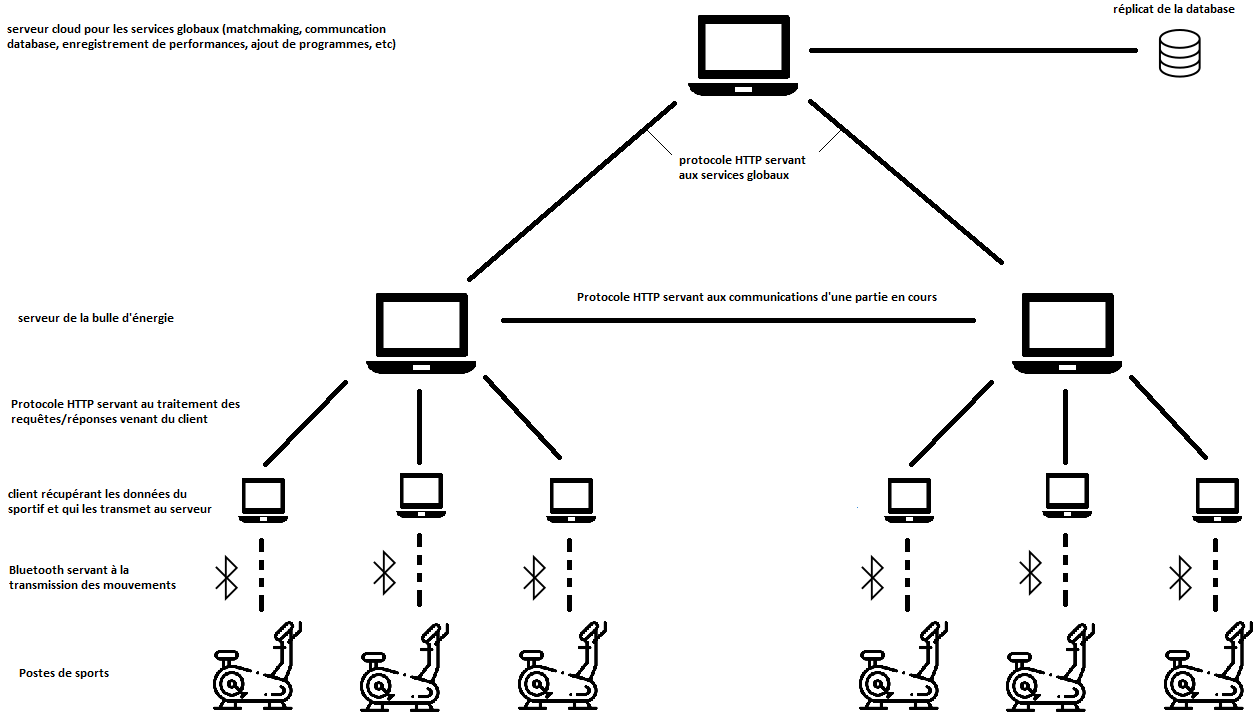


Figure architecture globale

## La concurrence

Un nombre non nul de concurrent ont déjà produit des applications impliquant sport et ludification. Nous allons voir dans cette partie lesquels existent. Ce type de technologie étant en plein essor, il en existe un très grand nombre, on ne pourra donc pas tous les montrer et très certainement nous en rateront.

### Kinomap



Figure Kinomap UI

Kinomap est une application disponible gratuitement sur IOS et Android permettant de suivre un tracé extérieur avec de beaux paysage depuis son tapis de course, vélo ou rameur chez soi. L’application offre un système de monitoring. Mais surtout, il communique avec les appareils de fitness pour modifier leurs résistances ou leurs inclinaisons en fonction du terrain virtuel affiché. La seule partie ludique disponible est de pouvoir se confronter à d’autres personnes. La vidéo étant géolocalisé on peut suivre son tracer sur une carte et pourquoi pas un jour aller sur place.

### Run social



Figure Run social UI en course

Run social est un concurrent direct de Kinomap et propose donc aussi de parcourir des paysages virtuels depuis des appareils d’intérieur. Il supporte cependant moins d’appareils.

### IConnect

A cause du nom très générique de cette application, il existe beaucoup d’application du même nom. Je n’ai donc pas trouvé d’information dessus. Je sais uniquement qu’elle existe selon un membre de T-Fitness et qu’elle est moins apprécié car fonctionne sur moins d’appareils que Kinomap et Run social. Elle est donc probablement dans la même veine d’application servant à courir à travers une vidéo préenregistrée.

### Exercube

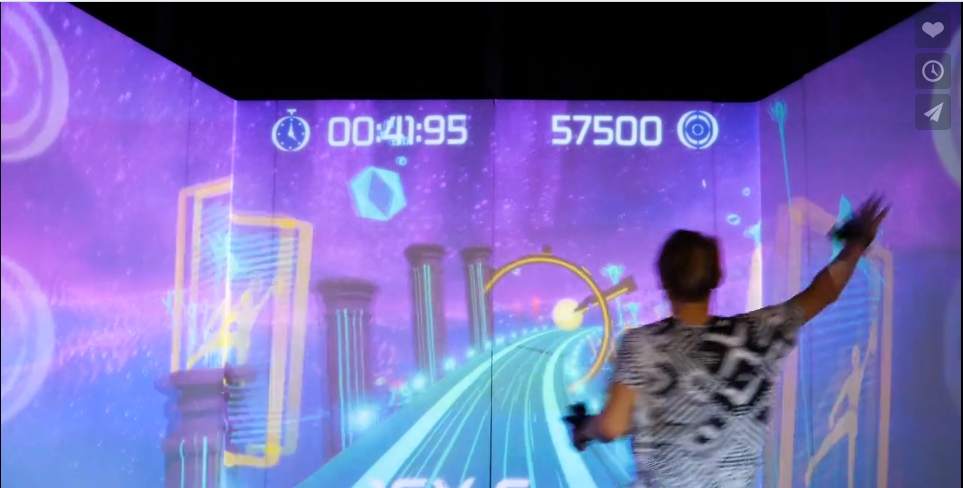


Figure Exercube

Exercube combine trois projecteurs, un capteur cardiaque ainsi qu’un détecteur de mouvement qui leurs est propre afin d’offrir une expérience unique combinant sport et jeu. Le principe est de se déplacer dans un cube et de réagir au décor en esquivant des laser et en ramassant des boules lumineuses affichées sur les parois du cube. Si je devais le comparer à un sport, je dirais que cette expérience ressemble à du squash. C’est typiquement le genre de technologie que nous souhaitons faire pour ce travail, peut-être juste en accordant un peu plus de contrôle à la partie sportive. C’est-à-dire un système qui va mieux contrôler le rythme auquel on bouge pour permettre aux non sportifs de se rendre compte du rythme qu’ils doivent maintenir pendant une longue période et ainsi éviter de les dégoûter.

### Zwift

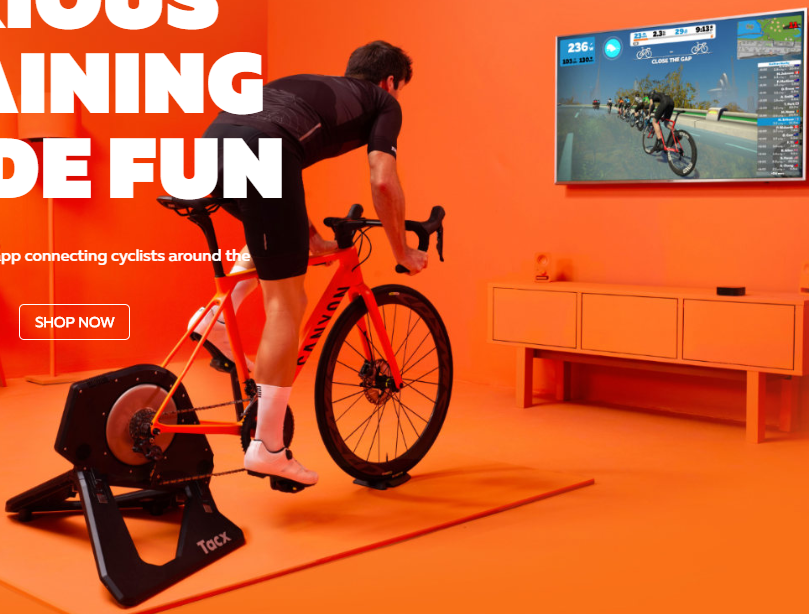


Figure Zwift

Zwift se veut le spécialiste du cyclisme en intérieur. Il propose des parcours totalement virtuels. Il est gratuit pendant 7 jour et passe ensuite à 15$ par mois. Des programmes spécifique et réfléchi sont proposé. Ils utilisent un home trainer pour adapter la résistance. La partie ludique arrive grâce à la personnalisation du vélo virtuel. Il faudra parcourir de nombreux kilomètre afin de pouvoir personnaliser son vélo et le rendre de plus en plus performant en jeu.

### Arcade Fitness

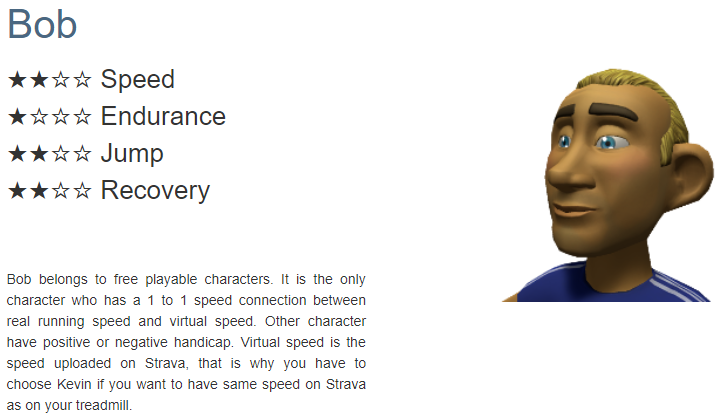


Figure Bob personnages jouable dans Arcade Fitness

Arcade Fitness est une application supportée sur Android et Windows 10 permettant de faire des courses de vélo sur des pistes virtuels. Le tout en utilisant un tapis de course ou un vélo d’intérieur. Il est possible de choisir divers types d’entrainement (endurance, 5 \* 90 secondes, 2 \* 5 sprints etc). La grande force de cette application est qu’on peut y choisir un avatar avec 4 caractéristiques (vitesse, endurance, saut, récupération). Chaque caractéristique allant de 1 à 4 étoiles. Les personnages ne sont pas équilibrés afin de permettre à deux sportifs de divers niveaux de jouer entre eux. Le site conseil d’utiliser une manette 8BitDo Zero Gamepad pour simplifier les manipulations. L’application est donc plutôt familiale et entièrement dédiée au plaisir et offre quelques aspect entraînement physique.

### Fitness gaming

Fitness Gaming est un site web rassemblant tous les produits, solutions et services concernant le marché grandissant des jeux sportifs. Je conseil fortement d’y faire un tour pour découvrir les nombreuses technologies innovante et unique en leurs genres qui existe déjà : <https://www.fitness-gaming.com/>. Je retiendrais particulièrement infinity treadmill avec leurs tapis de course personnalisé. Protégeant les utilisateurs d’une chute et avec des manettes intégrées au barres de maintiens :



## Post analyse du concept

Après avoir épluché une partie des technologies déjà existantes, on se rend vite compte que nous sommes dans une période de grande créativité dans le domaine. Que par conséquent, les lignes que j’écris en ce moment ne seront peut-être même plus d’actualité au moment de la reddition de se projet. Les applications vont du simple moniteur sportif permettant juste d’avoir une belle image en courant, à des concepts les plus imaginatifs de jeux mélangé à des technologies jamais vu. Il sera donc difficile d’apporter une pierre à l’édifice qui n’existe pas déjà probablement en mieux. Je pense la partie la plus novatrice que nous pouvons apporter, c’est de trouver une formule peu chère et qui ne demande que très peu de matériel (éviter les casques virtuels par exemple). Car toutes les applications vues jusqu’à présent demande soit d’acheter du matériel relativement coûteux, soit ce sont des application unique ou peu rependu et donc peu accessible pour la population. Avec les Bulles d’énergies, nous aurions une accessibilité géographique, il faudrait donc réussir à faire un prototype peu coûteux et sans trop de matériel supplémentaire. Sans oublier que la partie « adaptative » au sportif, n’existe pas encore et donc le machine Learning pourrait être la pierre innovante que nous recherchons. Les concurrents jusqu’ici on résolut ce problème en le contournant. Ils offrent des programmes sportifs sélectionnable, ajoutent des malus et bonus aux avatars, mais aucuns n’ont vraiment cherché à détecter la qualité du sportif et adapter la difficulté de leurs applications en fonction de ce résultat. Pour ainsi diriger le sportif vers un entrainement sain pour son corps et dans le but d’améliorations quotidiennes. Étant donné que notre idée de concept ressemble à certaines applications existante (en beaucoup plus simple évidement) on sait que l’idée n’est donc pas loin du but recherché. On sait aussi, qu’on ne pourra pas concurrencer des grandes entreprises à gros budget et que par conséquent avoir un petit jeu concept pour mettre en avant « l’adaptation via du machine Learning » est notre priorité. Mais ce n’est qu’une fois réalisé que nous saurons vraiment si le concept est faisable et adapté aux bulles d’énergie.

# Conception & implémentation du prototype

Dans cette partie nous allons voir le code que j’ai écrit en Java afin de créer le prototype dont nous aurons besoin pour une analyse plus approfondie du concept.

## Boucles de jeux

La première étape pour la création d’un jeu est de se focaliser sur la boucle principale. Si on schématise cette boucle elle consiste à alterner organiser trois mécanismes fondamentaux :

* La réception des manipulations de l’utilisateur
* La logique du jeu
* L’affichage de l’état

### Méthode naïve

La première idée intuitive est donc d’écrire :

1. **while** (**true**)
2. {
3. processInput();
4. update();
5. **render();**
6. }

processInput servirait à interpréter les entrées des utilisateurs depuis le dernier appel à la fonction.

Update permet de faire avancer l’état du jeu, comme si nous avancions le temps d’un « pas ».

Render permettrait d’afficher l’état actuel à l’écran et donc s’occuperait de dessiner les éléments visibles selon l’état actuel du jeu.

L’inconvénient étant que la vitesse du processeur ne peut pas être déterminé et qu’il n’y a aucune corrélation entre le temps de cette boucle et le temps réel. Si le boucle a peu de calcul à faire et peut donc être calculé très rapidement, alors le jeu va avancer très vite de manière fluide. Mais potentiellement trop vite par rapport à la réalité. Si les calculs prennent trop de temps, alors l’image sera saccadée et on aura l’impression de visionner un film en stop motion.

Nous allons donc introduire deux concepts, le premier est la quantité de travail qu’il faut effectuer pour faire une itération. Et le second la vitesse de la plateforme. Car tous les pc/langage/code ne se valent pas en termes de vitesse. Tout deux influence le Frame rate qui est le nombre d’image que le peut afficher par second. Dans la boucle un frame rate correspond à une itération.

Dans les vieux jeux, les codeurs écrivaient du code spécifiquement pour une machine et donc ne se souciaient pas du second facteur. Leurs solutions étaient donc de faire exactement la bonne quantité de travail à chaque itération pour que le jeu semble avancer au rythme du temps réel. De nos jours la travail clé de cette boucle est donc de permettre de garder une vitesse consistante en dépit du matériel sur lequel tourne le code.

Derniers détails importants avant d’aller plus loin dans les patterns existants : Il faut absolument choisir précautionneusement le pattern utiliser afin d’optimiser cette partie du code. Car dans un jeu, c’est dans ce bout de code que va passer la plupart du temps de calcul. Et finalement, il faut aussi faire très attention lorsqu’on écrit du code comportant une boucle d’affichage afin que les deux puissent travailler sans s’interférer.

### Boucle avec délayage

L’étape suivante est de d’ajouter à chaque itération un délayage afin de rendre les temps entre chaque itération constante. Le code ressemble à ça :

1. **while** (**true**)
2. {
3. **double** start = getCurrentTime();
4. processInput();
5. **update();**
6. render();
8. sleep(start + MS\_PER\_FRAME - getCurrentTime());
9. **}**

On règle ainsi le cas où la boucle itère trop vite, mais dans le cas où nous avons trop de travail à effectuer lors d’une itération n’est pas résolu.

### Boucle avec étape à taille variable

Pour pallier au problème lorsque le temps de calcul est trop long, il est possible de déterminé combien de temps c’est découlé depuis la dernière iteration et donc d’avancer l’état du jeu en fonction du temps écoulé depuis la dernière itération. Voici ce que donne cette nouvelle version :

1. **double** lastTime = getCurrentTime();
2. **while** (**true**)
3. {
4. **double** current = getCurrentTime();
5. **double elapsed = current - lastTime;**
6. processInput();
7. update(elapsed);
8. render();
9. lastTime = current;
10. **}**

Avec ce code, nous avons un jeu qui itère avec une vitesse constante qu’importe le matériel sur lequel il tourne. Et les machines puissantes ont un visuel plus lisse. Malheureusement, le jeu est devenu non-déterministe.

Si un objet se déplace, que deux machines utilisent des floatting points (donc un nombre approximé). Alors si un ordinateur puissant permet 50 itérations en un certain temps, un autre ordinateur moins puissant permettra moins d’itération et donc l’erreur d’approximation sera différente entre les deux PC. En conséquence l’objet ne sera pas au même endroit à la fin de la trajectoire. Ce qui pour un jeu multijoueur pourrait poser problème.

### Boucle à délayage sur la logique

Pour éviter d’être non-déterministe, nous allons ajouter une boucle sur update (donc la mise à jour de l’état du jeu) afin de mettre à jour en fonction du temps écoulé depuis la dernière itération. Si un long moment c’est passé avant la dernière itération alors nous effectuons plusieurs fois la fonction update. Ainsi un ordinateur lent à l’occasion de « rattraper » son retard sans perdre du temps avec l’affichage. Évidement on souhaiterait avoir une granularité de mise à jour aussi petite que possible, mais il faut s’assurer qu’elle soit suffisamment grande pour que les ordinateurs les plus lent puisse quand même rattraper leurs retard. Le code ressemble à ceci :

1. **double** previous = getCurrentTime();
2. **double** lag = 0.0;
3. **while** (**true**)
4. {
5. **double current = getCurrentTime() ;**
6. **double** elapsed = current - previous;
7. previous = current;
8. lag += elapsed;
9. **processInput();**
10. **while** (lag >= MS\_PER\_UPDATE)
11. {
12. update();
13. **lag -= MS\_PER\_UPDATE;**
14. }
16. render();
17. }

Un dernier souci se pose avec ce type de boucle. Lors de l’affichage, on peut se trouver entre deux intervalles de la mise à jour de la logique de jeu. Et donc pour afficher un objet en mouvement entre deux points de mise à jour il faut encore utiliser une extrapolation lors du rendu. Ce qui permet de gagner en précision. La fonction render prendra en paramètre l’extrapolation et dessinera les objets avec en la considérant. Une bonne implémentation de ce dernier type de boucle peut-être trouvé sur <http://www.java-gaming.org/index.php?topic=24220.0> et c’est ce que j’ai utilisé dans mon projet actuellement en modifiant ce qui m’arrangeais. J’ai notamment dû modifier la fonction draw en paint justement pour éviter les interférences entre la boucle du jeu et la boucle d’affichage de swing.

## Scrolling vertical infini

Afin de simuler un trajet infini, J’ai implémenté un effet de scrolling vertical infini. A chaque mise à jour de l’état du jeu, je déplace vers le bas deux images qui sont de la taille de l’écran. Lorsque l’image du bas sort de l’écran, Je la replace au-dessus de l’autre image. Ainsi de manière transparente à l’utilisateur on peut se déplacer de manière infinie vers le haut sans sortir de l’image. Mais surtout sans devoir instancier et lire les fichiers images en permanence pendant le jeu. Car les lecture mémoire peuvent être très coûteuse en temps et pourrait faire survenir du lag.

Voici un petit schéma expliquant où se trouve le code concernant cette fonctionnalité dans mon projet.

La classe GameLoop va donc appeler la méthode updateGame de la classe GameEngine. La classe GameEngine qui contient la position actuelle des murs, va faire avancer d’un « pas » l’état du jeu. Au moment du rendu graphique, la classe GameLoop va appeler la méthode repaint (propre à Swing) du JPanel (conteneur graphique de Swing) contenant le jeu avec l’interpolation précédemment calculé. Ce qui aura pour effet de décaler l’arrière-plan vers le bas, le tout de manière fluide indépendamment de la puissance de l’ordinateur sur lequel on tourne.

Lors de l’appel à la méthode updateGame, tous les objets supposés se déplacer sont bouger d’un « pas ». Et donc peuvent potentiellement se collisionner. Il faudra donc prendre en compte ces collisions, les calculer de manière efficace en évitant une surcharge du travail de la boucle de jeu.

## Objets et collisions

Nous allons donc tout d’abord étudier les divers moyens de calculer une collision, choisir la plus adaptée, puis l’implémenter dans notre projet. Deux étapes sont nécessaires dans une collision. La première est la détection. Elle permet de déterminer si deux corps (dans notre cas rigide) s’intersecte. La seconde étape consiste à appliquer un effet sur les deux corps qui se chevauche afin de revenir à un état logique pour la physique connue. Puisque notre objectif est de prouver un concept, cette partie du projet n’est pas la plus importante car dans un outil comme Unity ce type d’interaction est quasiment géré automatiquement. Si toutefois le projet par la suite ne sera pas écrit avec un outil dédié, je conseil fortement de lire ce site expliquant comment détecter et résoudre les collisions entre objets solide : <https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects>

## Méthode naïve

La façon la plus simple de détecter une collision consiste à considérer un objet dans un rectangle (ou une sphère) circonscrit. Deux objets sont considérés comme chevauché si les deux rectangles se chevauchent. Cette méthode est pratique car elle diminue grandement la complexité de calcul de la collision. Malheureusement, elle est fausse dans de nombreux cas on détecte une collision sans qu’elle ait lieu :

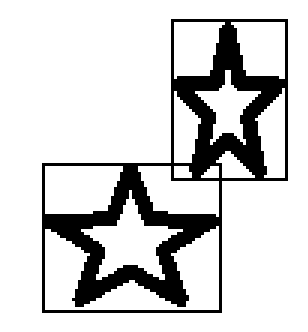


Figure détection fausse d'une collision avec une méthode naïve

Pour ces qualités et défauts, la méthode naïve est souvent utilisée pour choisir s’il faut approfondir la détection avec un calcul plus complexe. Si les rectangles ne s’intersecte pas, il ne se collisionnent pas. Si les rectangles se chevauchent, on utilise une méthode plus précise de détection de collision entre les deux objets. Ces deux phases sont appelées respectivement « Broad phase » et « Narrow phase ».

## Broad Phase

Une manière un peu plus élégante d’élaguer les potentiels collision est d’aligner l’axe des objets avec leurs rectangle circonscrit. Ce qui permet de diminuer l’aire d’erreur.

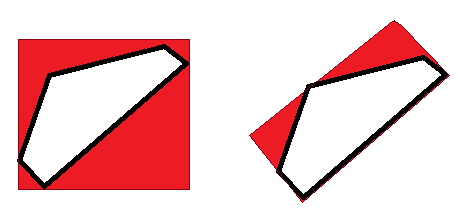


Figure aire d'erreur, méthode naïve contre axes alignés

Pour implémenter cette fonctionnalité quelques mathématiques basique seront nécessaire. Mais on se rend vite compte, que c’est du travail non nécessaire car dans notre concept les objets à collision potentiel seront relativement rectangulaire (personnage, et rocher). Et une petite approximation pour les collisions avec les objets est acceptable pour une preuve de concept.

On peut donc se permettre d’éluder ainsi complètement la Narrow phase et simplifier le problème pour un gros gain de temps. Si le projet évolue et que mon successeur souhaite implémenter lui-même des collisions plus évoluées. Je le renvoie à nouveau sur le site cité précédemment qui va beaucoup plus loin dans la compréhension des collisions. Voici quelques pistes évoquées par exemple :

* Le tri sur projection d’un axe
* Tri par arbre de volume englobant
* Traitement spécial des objets concaves
* Théorème de l’axe séparant
* Calcul de la distance entre objets
* Etc.

Concrètement, J’ai décidé de tout de même écrire une Broad phase et une Narrow phase. La Broad phase consistera à vérifier si les objets tombant du ciel se situe verticalement sur l’axe dans lequel le personnage peut se mouvoir. Tandis que la Narrow phase va vérifier la collision avec les deux rectangle circonscrit. On perd quelques instructions en temps de calcul car on vérifie deux fois la verticalité de la collision. Mais si quelqu’un reprend le projet, il n’aura qu’à modifier les deux méthodes déjà existantes pour affiner le calcul. Ces deux méthodes se trouve dans la classe GameEngine sous les noms checkCollide et isReallyColliding. Elle parcourt toutes deux la liste des objets tombant que contient la classe GameEngine et vérifient s’il y a collision avec le personnage. Pour simplifier le prototype, je n’ai pas considéré les collisions entre objets puisqu’ils bougent tous dans la même direction à vitesse relativement égales.

## Résolution de collision

Une fois que la Broad phase à détecté une collision potentiel et que la Narrow phase à validé la collision, il faut réagir, repositionner les objets et réadapter leurs vitesses. Étant donné, que la détection de collision est commune à tous les objets (dans notre cas), mais que la façon de réagir dépends du type de l’objet, j’ai profité de la liaison dynamique à travers la hiérarchie des objets tombant du ciel pour implémenter divers comportements lors de leurs collisions avec le personnage.

Figure hiérarchie de collision

Grâce à cette hiérarchie, chaque objet peut définir le comportement qu’il souhaite avoir en influençant la vitesse max qu’il reçoit. L’objet prend la vitesse max actuel annoncé par le moteur de jeu et propose en retour une nouvelle valeur. Le moteur de jeu en considérant les limite maximum et minimum, modifie la valeur. À noter qu’on pourrait améliorer cette hiérarchie en ajoutant une couche d’abstraction entre FallingObject et les objets concrets afin de séparer les objets positifs et les objets négatif et donc éviter de devoir redéfinir à la construction le paramètre IS\_NEGATIVE. Je ne l’ai pas fait car on n’a que deux objets positif (Bonus et Shield).

### Rochers & Cœurs

Les rochers et cœurs modifie donc la vitesse maximum en multipliant par un facteur voici la table de correspondance Classe/facteur.

|  |  |
| --- | --- |
| Nom de classe | Facteur |
| BigRock | -0.3 |
| Rock | 0.25 |
| LittleRock | 0.5 |
| Shield | 1 |
| Bonus | 1.5 |

### Boucliers

Le bouclier en plus d’impacter de manière neutre la vitesse maximale. Il active aussi une protection pendant un temps déterminé. C’est donc la seule classe redéfinissant la méthode giveShield en retournant true afin de signaler au moteur de jeu qu’il doit activer le bouclier, puis démarrer un thread pour le désactiver plus tard.

Au niveau du code la partie du moteur de jeu gérant la collision ressemble à ça :

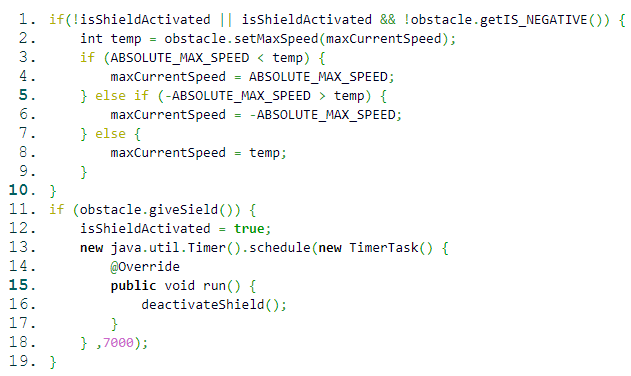


Figure resolveCollide méthode

## Vitesse de scrolling

On vient de le voir avec les collisions, une des missions les plus importantes du moteur de jeu est de contrôler les vitesses des divers objets du jeu. En l’occurrence la vitesse du mouvement du personnage de gauche à droite, le mouvement des objets tombant, et surtout le mouvement de l’arrière-plan. Ce dernier étant un peu plus complexe nous allons expliquer les divers paramètres qui l’influence.

Tout d’abord, il a fallu définir plusieurs constantes ou variable :

* La vitesse actuelle : speed
* La vitesse maximum actuelle maxCurrentSpeed définissant une limite maximum pour speed. En valeur absolue.
* Une limite non franchissable maximum pour les deux valeurs précédentes afin d’éviter qu’en multipliant les bonus on obtienne un jeu à une vitesse absurde : ABSOLUTE\_MAX\_SPEED = 40
* Une valeur de départ pour maxCurrentSpeed vers laquelle on tend lorsqu’aucun effet n’est appliqué par une force externe (principalement les collisions) : MAX\_SPEED = 25
* Une variabilité maximum entre chaque « pas » du jeu pour speed : MAX\_SPEED\_STEP = 3
* La vitesse de restabilisation de maxCurrentSpeed vers sa valeur de départ : MAX\_CURRENT\_SPEED\_STEP = 2
* La vitesse de restabilisation de maxCurrentSpeed vers sa valeur de départ (en milliseconde) : MS\_BETWEEN\_MAX\_SPEED\_UPDATES = 300

La vitesse du mur va donc entièrement dépendre de la variable speed. Pour déterminer sa valeur nous prenons l’effort du sportif qui est une valeur calculée de manière complexe que nous allons voir plus tard. Cette valeur se trouve entre 0 et 1. Ce qui permet de déduire un pourcentage entre 0 et maxCurrentSpeed. La variable speed est en permanence calculée par la boucle de jeu comme un pourcentage d’effort du sportif. Le moteur du jeu utilise ce pourcentage pour donner une vitesse concrète au déplacement de l’arrière-plan.

En parallèle à ce calcul, les collisions vont influencer la variable maxCurrentSpeed pour réduire ou augmenter l’échelle de valeur. Et à chaque itération de la boucle du jeu maxCurrentSpeed va automatiquement se restabiliser vers sa valeur de départ avec des longueurs fixe. Ce qui donne une impression de choc, puis de ré-accélération petit à petit.

Tout ce système est ensuite protégé par des valeurs limite qu’est ABSOLUTE\_MAX\_SPEED.

# Mesureurs d’efforts

Comme on l’a vu dans la partie précédente, l’effort du sportif est mesuré pour finalement donner un pourcentage entre 0 et 1 afin de faire mouvoir l’arrière-plan. Nous allons voir ici les méthodes que j’ai utilisé pour mesurer l’effort du sportif.

## EffortCalculator

Le premier mesureur à servit à vérifier que tout jusqu’ici fonctionnait bien (boucle de jeu, affichage et déplacement du mur). C’était une implémentation la plus simple qu’il soit avant même de recevoir du matériel adapté. Nous avons donc utilisé une souris pour mesurer un effort en fonction du déplacement vertical de la souris. Puisque le mesureur allait changer et que comme nous l’avons vu, les technologies sont nombreuses pour capter un mouvement de sportif, il fallait rendre le type de mesureur rapidement changeable. C’est pour cette raison que nous avons écrit la classe abstraite EffortCalculator qui sera un modèle sur lequel construire un mesureur. Il suffira d’étendre de cette classe et avec peu de notion, utiliser son comportement pour implémenter son propre mesureur qui fonctionnera immédiatement avec la boucle de jeu en changeant très peu de code.

La classe EffortCalculator est basé sur une variable particulière : expectedMaxAverage qui représente la valeur attendue par le mesureur comme étant l’effort optimale du sportif. C’est probablement la variable la plus importante de ce projet. Il est donc bien important de comprendre à quoi elle sert. Lorsqu’une sous-classe d’EffortCalculator appelle le constructeur de sa super-classe (EffortCalculator donc). Il devra lui spécifier cette valeur pour indiquer quel sont les valeurs attendues par le mesureur pour une séance « parfaite ». C’est donc cette valeur qui devra plus tard être précalculé avec du machine Learning en se basant sur les autres séances du sportif.

À côté de cette variable, nous avons la variable effort qui sera une variable avec des setters/getters thread safe qui sera fixé par le mesureur lui-même. Et qui pourra être lu par la boucle de jeu de manière asynchrone pour adapter la vitesse de l’arrière-plan.

La classe EffortCalculator offrira aussi une classe interne observable qui permettra à l’UI d’afficher une barre d’effort en temps réel.

Finalement, l’effort ne possède pas de valeur maximum. Il est donc impossible de donner une valeur maximum à la construction. Nous avons donc considéré le maximum de la barre d’effort comme la valeur maximum jamais atteinte par le sportif. En résumé la barre d’effort ressemble à ceci :

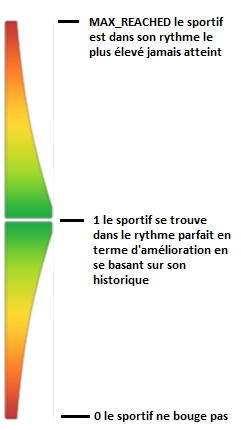


Figure Barre d'effort, en vert l'arrière-plan se déplace à vitesse maximum. En rouge l'arrière-plan ne se déplace pas.

MAX\_REACHED est dynamique si à tout moment on appelle la méthode setEffort avec une valeur gigantesque d’effort, alors cette valeur sera considérée comme le nouveau MAX\_REACHED. À terme, cette valeur devra être stockée dans une base de données sous plusieurs formes selon les types d’appareil, le type de programme, le type de mouvement et ce pour chaque utilisateur. Mais pour le moment, elle est stockée dans le fichier CycleEffortConfig.properties

### Mesureur souris

Le mesureur via la souris aura donc pour but de mapper le signal du mouvement vertical de la souris vers les valeurs de la barre d’effort. Nous avons donc utilisé un buffer circulaire qui retient les x dernières valeurs de déplacement de la souris (x étant une valeur arbitraire de la taille du buffer). Plus le buffer est grand, plus nous somme précis dans le calcul d’une moyenne. Mais plus le buffer est petit et plus nous somme rapide à détecter un changement de rythme (puisqu’on met moins de temps à reremplir le buffer des nouvelles valeurs).

Ce mesureur n’ayant servit qu’à prouver que d’autres parties du code fonctionnaient bien, je ne m’attarde pas dessus. Le code n’a même pas été gardé, il faudra remonter dans les commits du git si vraiment on souhaite revoir cette partie.

### Mesureur d’accélération

Une fois que la boucle de jeu, que le moteur de jeu et que le premier mesureur était fonctionnel. Il a fallu faire évoluer le mesureur. Nous avions décidé d’utiliser un accéléromètre Shimmer3 et pour communiquer avec l’appareil, il nous faut du Bluetooth.

Plusieurs classes ont été écrite pour la communication. La première couche permet d’identifier les périphérique alentours et ensuite de demander leurs services disponibles. BluetoothServicesDiscovery est la classe en question et implémente l’interface DiscoveryListener. Pour mieux comprendre comment fonctionne le Protocol, le site d’oracle est très bien fait : <https://www.oracle.com/technetwork/articles/javame/index-156193.html>. Chaque périphérique possédant son adresse Bluetooth (un string l’identifiant), on peut retrouver la Shimmer3 avec son code écrit sur l’appareil (RN42-B86D). La recherche des services se faisant via leur UUID, on peut trouver la liste sur <http://www.bluecove.org/bluecove/apidocs/javax/bluetooth/UUID.html>. Le service que nous cherchons est OBEX pour « object exchange » qui permet d’échanger des données entre appareils. En l’occurrence ici entre mon ordinateur et la Shimmer3. Plus précisément nous avons utilisé le service RFCOMM qui sert de couche de transport pour OBEX en Bluetooth. Le code binaire de RFCOMM est 0x0003

Un fichier de configuration dans le package IMU nommé IMUConfig.properties contient l’adresse bluetooth et le code binaire pour RFCOMM. Ainsi, si on souhaite changer d’appareil, il suffira de modifier ce fichier. À posteriori, il faudra évidement permettre de sélectionner l’appareil via une interface graphique.

Afin de simplifier l’écriture d’une classe se connectant en Bluetooth avec la découverte de service. J’ai écrit une classe générique (BluetoothPairing) permettant d’obtenir un manager de connexion Bluetooth. Concrètement cette classe permet en spécifiant l’adresse Bluetooth et le service qu’on cherche à utiliser. De se connecter à se service et d’offrir une API simple pour échanger des données.

Une dernière classe (BluetoothIMUAPI) est la classe spécifique à l’IMU et utilise la classe générique pour se connecter au service RFCOMM de la Shimmer3. C’est cette classe qui va lire le fichier de configuration, et utiliser l’API générique pour obtenir les données des accélérations. Elle s’instancie facilement et offre des méthodes pour configurer la Shimmer3, lancer la capture des données, la stopper et récupérer les accélérations. Toute la logique de cette classe est une retranscription en java d’un script Python que Mr. Satizabal Mejia Hector Fabio à bien voulut me partager.

Voici un résumé en image des classes impliquées pour l’utilisation d’une IMU dans mon code :

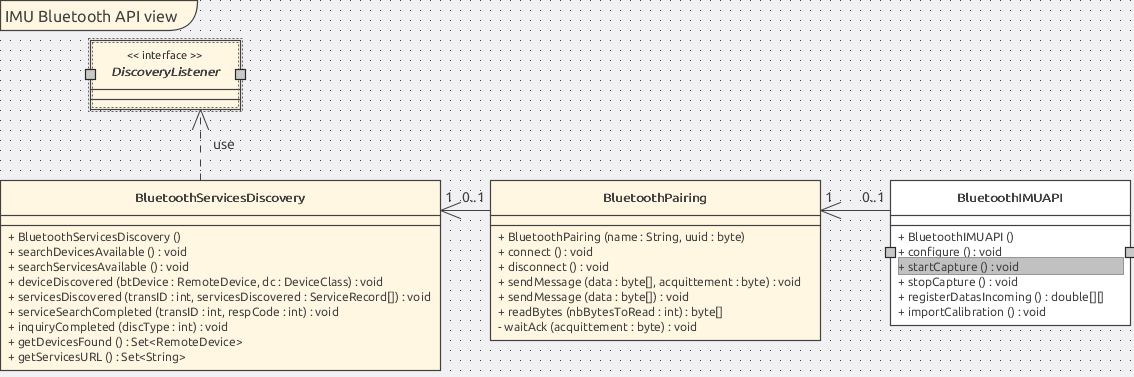
La première implémentation naïve pour un mesureur avec accéléromètre a été d’utiliser une liste chaînée. À chaque nouvelle donnée reçue, je l’ajoute à la fin de la liste et je supprime le premier élément (donc le plus anciens). Le calcul de l’effort ce faisait ensuite en parcourant la liste, en faisant une moyenne et en divisant par la moyenne que l’on vise. Le tout sur les 3 axes et en valeurs absolues. Ce calcul peut être associé au calcul de l’air sous la courbe de la fonction abs(sin(x)) :

Figure Vue de la hiérarchie de classe pour la connexion Bluetooth avec la Shimmer3

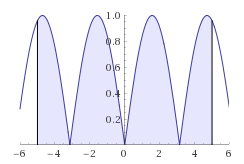


Figure Résultat fictif de l'implémentation naïve si le mouvement était parfait

Le problème avec cette méthode est qu’elle n’est pas pertinente. Le mouvement étant cyclique, l’accélération va passer régulièrement par la valeur 0. On considérera cette entrée comme si l’utilisateur ne fait pas d’effort à ce moment alors qu’il est pourtant en mouvement. La méthode fonctionne uniquement parce que l’amplitude de l’accélération augmente avec la fréquence et donc la moyenne augmente. La classe implémentant ce mesureur est IMUEffortCalculator.

### Mesureur de fréquence

Afin d’augmenter la pertinence de notre mesureur, nous allons calculer la fréquence du mouvement cyclique. Deux méthodes m’ont semblé possible. La méthode par seuil ou fast Fourier transformation. J’ai commencé par cette deuxième méthode. Malheureusement, cette méthode donne des pics pour chaque fréquence trouvée dans notre signal ce qui donne à peu près :

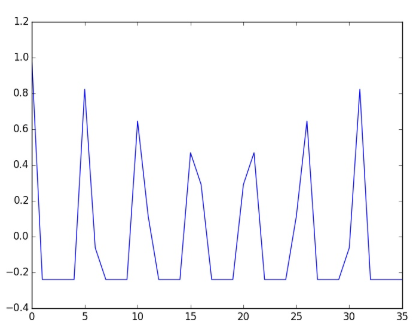


Figure résultat fictif de fast Fourier transformation. Chaque pics seraient une fréquence plus ou moins présente dans nos données d’accélération.

Le souci était qu’il était difficile de retrouver la bonne fréquence parmi les bruits (lequel de ces maximums locaux choisir). Mais même en supposant que j’y arrive, les changements de rythme auraient été trop brusque avec cette méthode. Il aurait fallu attendre que la nouvelle fréquence soit majoritaire et le détecteur aurait « sauté » d’une fréquence à l’autre. Alors que nous souhaitons une variation souple.

J’ai tout de même essayé d’implémenter cette méthode car j’ai trouvé un code source en java disponible <https://stackoverflow.com/questions/3287518/reliable-and-fast-fft-in-java>. Le résultat était trop variable comme attendu. J’ai donc abandonné cette méthode pour utiliser la méthode par seuil.

La méthode par seuil consiste à décider d’une valeur seuil. Pour chaque donnée d’accélération qui passe de dessous ce seuil à au-dessus, nous comptons un cycle. Cette méthode fonctionne bien si le mouvement cyclique est propre et se rapproche d’un sinus :

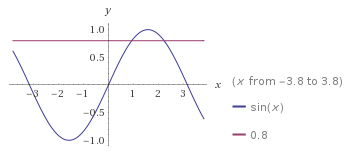


Figure Résultat fictif de la méthode par seuil. Ici en x = 1 on compterai un cycle.

Avec cette méthode, J’étais capable de détecter un mouvement de la main cyclique. Plus le mouvement était rapide, plus je comptais de cycle. Je divisais mon nombre de cycle obtenu par le timestamp entre ma donnée la plus récente et la plus ancienne de ma liste chaînée. Et ainsi j’obtenais une fréquence. C’est avec cette méthode que nous avons enfin essayé en condition réel l’application.

Nous avons utilisé un tapis de course, placé la Shimmer3 sur la cheville, testé et ça ne fonctionnait pas aussi bien qu’à la main. Nous allons voir avec ce dernier mesureur quels sont les soucis principaux rencontré et comment en résoudre certains.

### Mesureur de vitesse de course

Après avoir mesuré mon mouvement sur un tapis de course, nous nous sommes rendu compte qu’avec les chocs du pied sur le sol, la fonction résultante ressemble beaucoup moins à une fonction sinus :

Figure accélérations sur tapis de course et choix du seuil

On se rend compte qu’il est impossible de choisir un seuil à cause des chocs. Vous pouvez voir sur l’image ci-dessus deux choix de seuils pertinent qui détecterait deux cycles à un endroit ou il devrait n’y en avoir qu’un.

La première idée pour résoudre ce problème est d’utiliser deux seuils. Un seuil supérieur et un seuil inférieur. Ainsi un double pic d’accélération dû à un choc passant deux fois le seuil supérieur ne sera compté qu’une fois tant qu’il n’aura pas passé le seuil inférieur. Exemple :

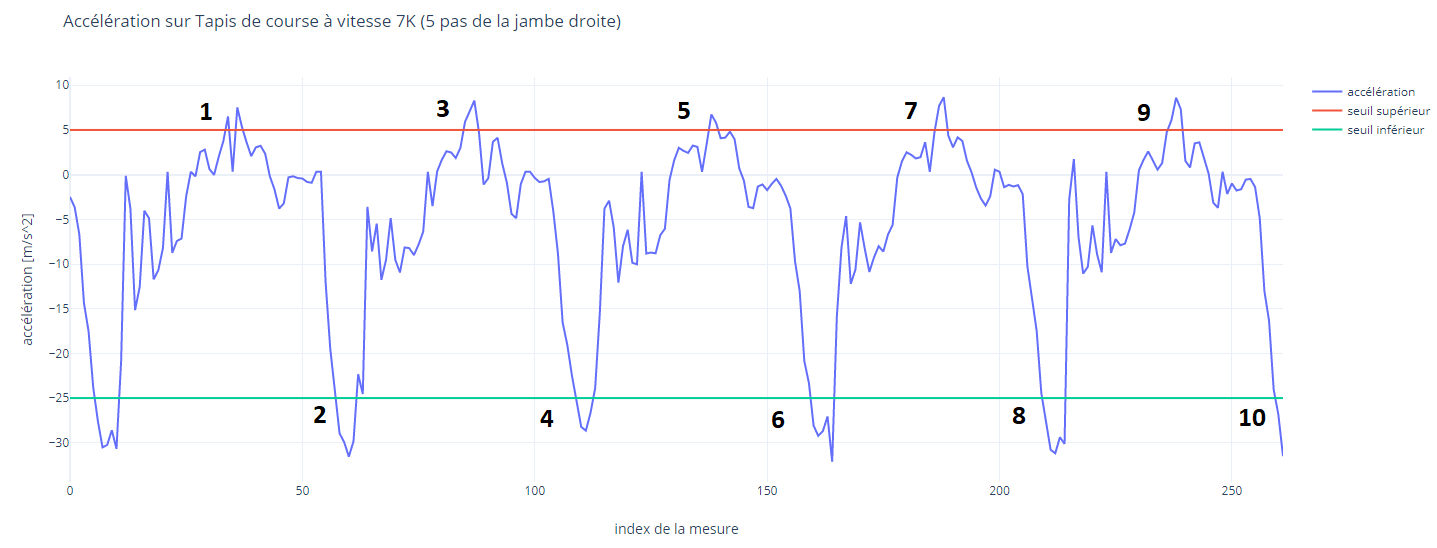
Le deuxième problème qu’il a fallut résoudre est de déterminer la valeur de ce seuil. L’idée de base à été de se dire que nous allions prendre la valeur maximum, la valeur minimum et que nous placerions les deux seuils à un certain pourcentage entre ces deux valeurs. La formule donne par exemple pour le seuil supérieur :

Figure comptage des demi-cycle avec deux seuils

Deux problèmes interviennent avec cette formule. Tout d’abord, l’appareil parfois envoie de fausses valeurs dont certaines dépassent le maximum ou le minimum. Ensuite cette façon de faire fonctionne très bien, si l’amplitude augmente. Mais si on ralentit et que l’amplitude diminue les nouvelles données peuvent ne plus jamais passer les seuils. Exemple :

Figure Fausse donnée augmentant drastiquement le maximum

On voit donc que les prochains cycles ne seront plus détectés. Afin de rendre plus robuste la détection. Nous avons donc calculé un maximum local et non global. Donc à la place de retenir la valeur maximum, nous calculons le maximum contenu dans le buffer. Ce qui augmente la complexité des calcules en O(n) avec n la taille du buffer car il faut à chaque nouvelle mesure parcourir notre liste chaînée pour déterminer le maximum actuel. Cette méthode permet d’oublier les trop grandes valeurs fausses après un certain temps, mais le temps pour y parvenir est trop long. Nous avons donc finalement utilisé non pas un maximum absolu, mais un maximum médian. Nous prenons la 8ème valeur la plus grande que l’on considère comme le maximum actuel. Ainsi la médiane n’étant pas impacté par des valeurs extrêmes, n’est pas impacté par une fausse valeur extrême. On peut donc avoir jusqu’à 7 valeurs extrême fausses dans notre buffer sans que notre seuil soit trop haut pour ne plus détecter les cycles suivants.

Cependant, bien que maintenant nos seuils soient dynamiques et se calque sur des valeurs maximum/minimum médian, nous avons toujours une lenteur à détecté une baisse d’amplitude. Il faut attendre que suffisamment de nouvelles données aient écrasé les valeurs de notre liste, pour que les seuils redescendent et qu’on puisse à nouveau détecter des cycles.

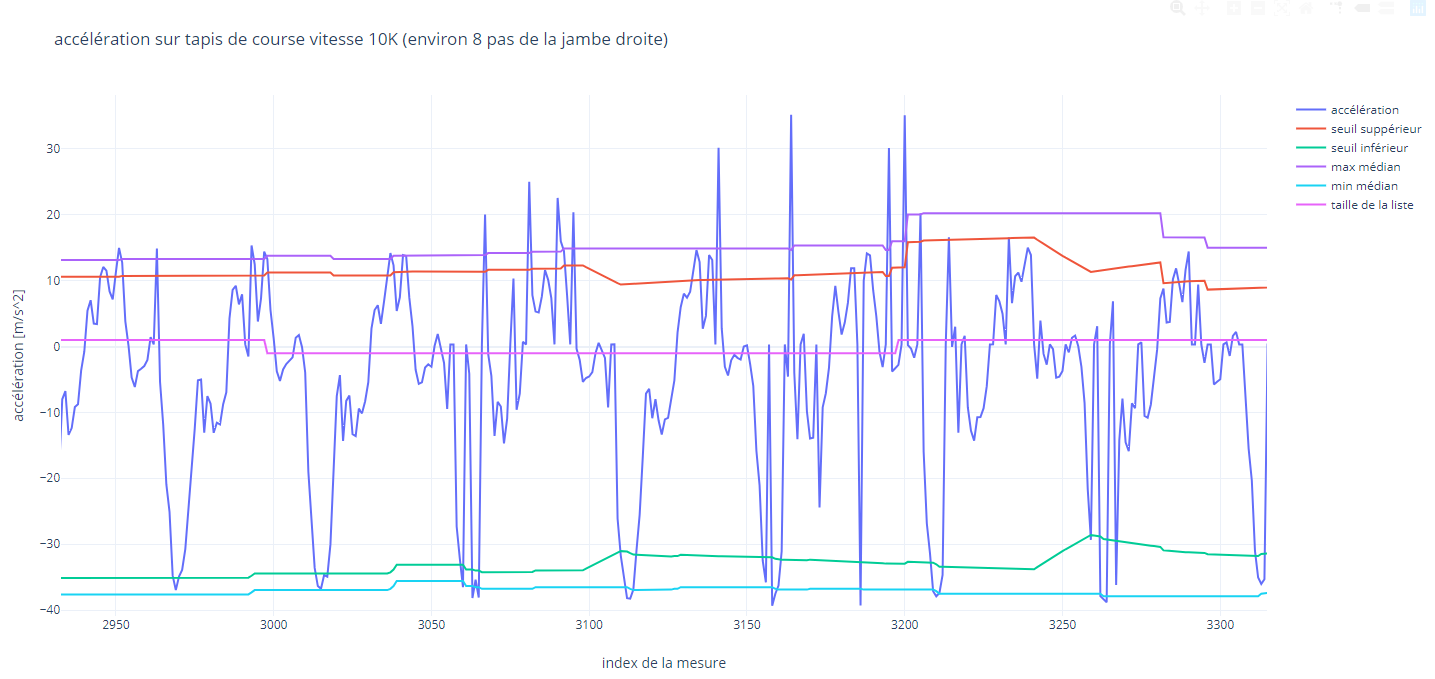
Nous allons donc aussi ajouter un système prévoyant le prochain cycle afin de baisser manuellement le seuil si nous n’arrivons plus à détecter de cycle pendant trop longtemps. En somme, nous détectons un premier cycle qui prend x ms pour parvenir. On s’attend donc à détecter un autre cycle dans environ x ms. Si nous ne détectons pas de cycle à ce moment, nous augmentons le pourcentage de l’équation 1) ce qui augmente la marge d’erreur de nos seuils. Voici ce que donne actuellement les graphiques complets :

Figure graphique complet avec seuil dynamique, maximum médian et prévision du cycle suivant

Actuellement on arrive donc à détecter qu’un cycle n’est pas arrivé à temps. Malheureusement lorsque beaucoup de fausses données s’enchaînent, ça ne suffit pas à maintenir un calcul de fréquence correct comme on le voit dans l’image suivante :

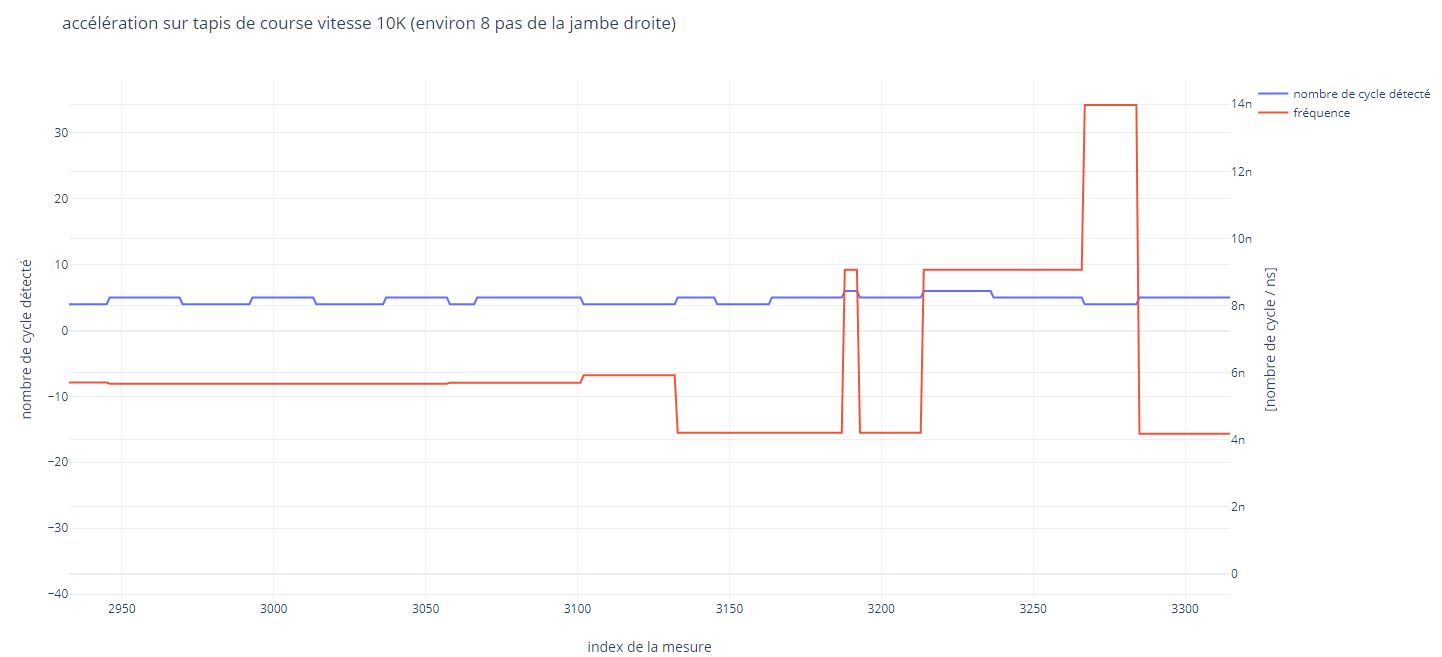


Figure détection de la fréquence actuellement encore instable

Il faut ajouter que nous avons encore une fréquence qui saute d’une valeur à l’autre comme pour fast Fourier transformation. La raison étant que je prends pour l’instant une fréquence médiane plutôt qu’une moyenne. Nous verrons dans la partie « État actuel » les quelques erreurs que nous avons détecté avec ce système et donc ce qu’il reste à faire.

## Déplacements horizontaux

Afin de déplacer le personnage de droite à gauche pour esquiver les obstacles, nous avions besoin de détecter un mouvement non cyclique. En effet, si nous utilisions un accéléromètre uniquement pour déplacer le personnage, nous reviendrions en permanence à notre place de départ. Si nous faisons un mouvement brusque de droite à gauche, l’accéléromètre va détecter une forte accélération, puis une forte décélération. Il nous faudra donc un système permettant de définir le début du mouvement et sa fin pour expliciter quelle partie du mouvement est significatif (son accélération ou sa décélération). À postériori, si nous trouvons une technologie capable de remplacer la Kinect, nous pourrions détecter des positions plutôt que des mouvements. La suite du chapitre explique quelles ont été les technologies utilisées pour déplacer le personnage.

### Clavier

La manière la plus simple utilisée avant même d’obtenir les senseurs était d’utiliser les flèches directionnelles du clavier. Tout comme pour la souris avec les mesureurs, cette façon de faire n’a pas été gardée car elle servait uniquement à vérifier que d’autre parties du code fonctionnaient. En l’occurrence le déplacement du personnage de gauche à droite sans dépasser la taille du mur.

### Wiimote

Comme expliqué dans le choix des technologies, le premier appareil que nous avions choisi d’utiliser pour détecter un mouvement non-cyclique est une Wiimote. Bien que j’aie réussi à détecter l’appareil, je n’ai jamais réussi à échanger de données avec. Le problème était un souci de compatibilité entre windows 10 et Bluecove (la librairie permettant d’utiliser Bluetooth en Java. Voici l’explication en plus détaillé :

« As I suspected the lack of support for L2CAP (Wii remote’s data protocol) in WINSOCK and the fact that BlueCove (an Open Source implementation of JSR-82 Bluetooth for Java which we use for Wrj45 – WiiRemoteJ) uses WINSOCK as a default stack (kernel or resource enumerator) if a Windows computer has the ‘Microsoft Bluetooth Enumerator’ installed (XP can have multiple and Vista has it as standard) the Wii Remote will be visible (Simple Bluetooth etc), but not able to exchange data.

Hence, Microsoft Bluetooth Enumerator (which only supports RFCOMM) needs to be replaced with WIDCOMM provided with Broadcom products and suitable for majority of devices up to version 5.1.0.1100 and exclusive to Broadcom chipset devices after that (I suspect BlueSoleil has the same problem). […] Many non-brand products use Broadcom chipset and will work with newer versions of the software and driver (some may need VID\_xxxx PID\_xxxx hacks). […]»

Après avoir vu cette explication, j’ai donc essayé de modifier mon énumérateur Bluetooth par un énumérateur supportant WIDCOMM. J’ai aussi utilisé des dongles Bluetooth pour remplacer mon matériel. J’ai aussi cherché une librairie qui permettrais de communiquer en Java avec une manette Wiimote qui en plus n’utiliserais pas L2CAP mais toutes l’utilisent. Nous avons donc fini par abandonner cette technologie.

### Joy-con

Au fil de mes recherches pour la Wiimote, je suis tombé sur une librairie très modeste permettant la communication avec un Joy-con. Bien qu’elle fonctionne, que tous les boutons et les joysticks soient détectable, elle n’avait pas encore de fonctionnalité pour l’accéléromètre intégré à la manette.

Heureusement des personnes fournissent sur Github des recherches d’ingénierie inversée sur les Joy-con. J’ai donc pu modifier le code source de la librairie joyconLib pour y ajouter la lecture des données de l’accéléromètre.

### Phidgets

Les phidgets auraient été la solution suivante pour la détection d’un mouvement non cyclique. En se servant d’un phidget « bouton » et du phidget accéléromètre. Je n’ai pour le moment rien intégré dans le projet les concernant car les Joy-con sont acceptable sur un tapis de course. Lorsqu’on utilisera un vélo, il sera toujours possible de ne pas tenir le guidon. Cependant, pour le rameur nous devront probablement trouver une solution alternative pour bouger de gauche à droite.

# État actuel

## Prototype

Le prototype est pour le moment fonctionnel.

* Le mur avance au rythme qu’indique le mesureur.
* Les obstacles tombent aléatoirement à un rythme correct.
* Le joueur peut se déplacer de gauche à droite grâce à la manette Joy-con.
* La jauge d’effort s’affiche correctement selon ce que le mesureur détecte.
* Un menu est implémenté afin de commencer une partie.
* La détection de fin de partie après un laps de temps fonctionne.
* Le score s’affiche en fonction de la distance parcourue.
* Les collisions affectent la vitesse du mur et donc indirectement le score.
* Les bonus fonctionnent (boost de vitesse et bouclier)

Un bug reste à fixer, c’est lorsqu’on ramasse plusieurs fois le bouclier, le délai de disparition du bouclier n’est pas remis à zéro. Ce bug n’est pas une priorité.

Voici un UML simplifié de l’état actuel du projet :

## Mesureur d’effort

Actuellement le mesureur d’effort détectant la fréquence ne fonctionne pas correctement. Les fausses valeurs envoyées par l’IMU impactent mes calculs. De plus, au rythme de marche, la fréquence est arrondie à 0 je ne sais pas encore pourquoi. Le mesureur néanmoins s’approche gentiment de quelque chose d’utilisable pour commencer une collecte de données.

Aucuns tests n’ont pour le moment été fait sur un vélo ou un rameur qui sont les appareils que souhaite utiliser au début notre mandataire. Il reste donc des efforts à faire dans cette partie avant de pouvoir commencer à utiliser du machine Learning afin d’adapter le prototype au sportif l’utilisant. Néanmoins, le calcul de la fréquence est correct sur des données « propres », la prévision du cycle suivant fonctionne.

# Objectifs restants

Ce projet étant gigantesque, il ne sera évidement pas terminé d’ici la fin de mon Bachelor. Voici les quelques points que j’aimerais tout de même avancer avant mon rendu.

## Mesureur pour tapis de course

Le mesureur possède encore quelques défauts :

* À vitesse de marche, la fréquence est mise à zéro, probablement à cause d’une erreur d’arrondie avec les flottants. C’est un bug à corriger.
* Des valeurs erronées sont détectées par l’accéléromètre de la shimmer3. Je n’ai pour le moment aucune hypothèse sur la source de cette erreur. Qui malgré la grande gêne qu’elle procure m’a tout de même forcé à écrire un mesureur robuste aux erreurs de détection de mouvement.
* Le mappage entre la fréquence des pas de course et la barre d’effort se fait de manière linéaire. Or la fréquence entre une personne qui marche, une personne qui court et une personne qui sprint n’est pas linéaire. Les mesures de ma propre course à ces différentes vitesses ont déjà été prise. Il faut donc juste que j’en fasse un graphique, décide d’une équation de mappage adaptée et l’implémente dans le mesureur.

## Mesureur pour vélo

Tous les tests ont pour le moment été fait sur un tapis de course. Il faudra essayer si ce mesureur fonctionne toujours sur un vélo d’intérieur. En ce qui me concerne je m’attends à ce que le mouvement soit même mieux détecté car le vélo implique moins de choc et par conséquent moins d’imprécision.

## Mesureur pour rameur

Pareil pour le rameur, la principale différence est peut-être la lenteur du mouvement. Il faudra donc après test, rallonger la taille du buffer pour ce type d’appareil.

## Machine Learning

C’est une grosse partie qu’il reste à faire, créer suffisamment de données pour pouvoir utiliser du machine Learning. C’est-à-dire courir plusieurs fois 30minutes avec un mesureur entièrement fonctionnel. Puis, faire une analyse complète du type de machine Learning que nous allons utiliser pour se projet. L’implémenter puis tester. Je doute avoir le temps de réussir à commencer cette étape, cependant je la note ici car en commençant se travail c’était mon objectif de réussir à aller jusqu’à la fin de cette partie du projet.

## Transfert de connaissance

En écrivant les dernière ligne de se rapport intermédiaire, je me rends compte à quel point le transfert de connaissance est chronophage. Je pense donc qu’après avoir corrigé les quelques derniers soucis du mesureur. C’est la documentation, les tests fonctionnels et les commentaires de code qui seront ma priorité avant de viser un rendu final.

# Bibliographie

<https://fr.wikipedia.org/wiki/Kinect>

<https://www.01net.com/actualites/microsoft-enterre-kinect-mais-ses-technologies-vont-lui-survivre-1286872.html>

<http://www.xboxygen.com/News/29079-Azure-Kinect-le-nouveau-Kinect-a-399-destine-a-Windows>

<https://www.gamesradar.com/theres-a-new-microsoft-kinect-but-its-not-built-for-gaming/>

<https://www.01net.com/actualites/microsoft-enterre-kinect-mais-ses-technologies-vont-lui-survivre-1286872.html>

<https://www.youtube.com/watch?v=szajkbF9HxM>

<https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=423422&_ga=2.174022691.1206118813.1551355649-912524342.1551355649>

<https://www.shimmersensing.com/products/shimmer3-development-kit>

<https://www.shimmersensing.com/products/>

<http://www.nime.org/proceedings/2018/nime2018_paper0021.pdf>

<https://www.cdiscount.com/jeux-pc-video-console/r-wiimote.html#_his_>

<https://www.google.com/search?q=wii+sortie+date&oq=wii+sortie+date&aqs=chrome..69i57j0l2.2667j1j7&sourceid=chrome&ie=UTF-8>

<https://fr.wikipedia.org/wiki/T%C3%A9l%C3%A9commande_Wii>

<https://books.google.ch/books?id=pN-GDwAAQBAJ&pg=PA297&lpg=PA297&dq=shimmer3+application&source=bl&ots=SUYCEwS0_L&sig=ACfU3U0WY7C0Lq-8-MHOBDcaJV7MByFYMg&hl=fr&sa=X&ved=2ahUKEwjq_IC3sM_iAhX4ysQBHYDtArQQ6AEwCnoECAgQAQ#v=onepage&q=shimmer3%20application&f=false>

<https://www.google.com/search?q=acheter+joy-con&oq=acheter+joy-con&aqs=chrome..69i57.4036j1j9&sourceid=chrome&ie=UTF-8>

<https://fr.wikipedia.org/wiki/Joy-Con>

<https://fr.wikipedia.org/wiki/Dispositif_haptique>

<https://unity3d.com/fr/programming-in-unity>

<https://www.supinfo.com/articles/single/12-developpement-jeux-video-pourquoi-utiliser-unity3d>

<https://unity3d.com/fr/unity?_ga=2.61857260.1718087989.1559647406-681544978.1559647406>

<https://www.unrealengine.com/en-US/>

<https://www.youtube.com/watch?v=mVaf9v2Ya7M>

<https://wireless.wiki.kernel.org/en/users/documentation/bluetooth-coexistence>

<https://superuser.com/questions/1312004/solve-wifi-bluetooth-frequency-conflict>

<https://www.quora.com/Is-it-possible-to-connect-to-wifi-and-bluetooth-at-the-same-time-and-both-would-be-working-as-intended>

<https://www.prixtel.com/decouvrir-PRIXTEL/actualite/news/le-bluetooth-quest-ce-que-c-est-et-comment-ca-marche/>

<https://www.phidgets.com/>

<https://www.phidgets.com/?tier=3&catid=10&pcid=8&prodid=956>

L’Art du game design Par Jesse Schell publié par Pearson Education France en 2010 ISBN : 978-2-7440-2431-3

Icons made by <https://www.freepik.com/>

<https://www.kinomap.com/fr/>

<https://sphery.ch/>

<https://vimeo.com/297562260>

<https://zwift.com/en/?utm_source=google&utm_medium=cpc&utm_campaign=shift_eur_de+fr+it+es+nl+pl+dk+be+ch+lu_cycling_search_zwift_performance_mar19&gclid=CjwKCAjwue3nBRACEiwAkpZhmeEyYiW65q02SQeN1P8PLRM9K8B2JhoqB50ymi6R2jnLs54bKHrbyhoC0JQQAvD_BwE>

<https://www.mantel.com/blog/fr/zwift-how-to/>

<https://bipr.fr/arcade-running-avatars>

<https://bipr.fr/arcade-running-about>

<https://www.youtube.com/watch?time_continue=3&v=Usjh3NR35ng>

<https://www.bluegoji.com/>

<https://www.bluegoji.com/infinity>

<http://gameprogrammingpatterns.com/game-loop.html>

<http://www.java-gaming.org/index.php?topic=24220.0>

<https://www.toptal.com/game/video-game-physics-part-ii-collision-detection-for-solid-objects>

<https://www.oracle.com/technetwork/articles/javame/index-156193.html>

<http://www.aviyehuda.com/blog/2010/01/08/connecting-to-bluetooth-devices-with-java/>

<http://www.bluecove.org/bluecove/apidocs/javax/bluetooth/UUID.html>

<https://en.wikipedia.org/wiki/List_of_Bluetooth_protocols#Radio_frequency_communication_(RFCOMM)>

<https://www.researchgate.net/publication/261067439_Analyzing_Body_Movements_within_the_Laban_Effort_Framework_Using_a_Single_Accelerometer>

<https://xii9190.wordpress.com/2008/04/18/ms-intel-winsock-bluecove-and-wii/>

<https://github.com/dekuNukem/Nintendo_Switch_Reverse_Engineering>

<https://github.com/dekuNukem/Nintendo_Switch_Reverse_Engineering>

<https://github.com/dekuNukem/Nintendo_Switch_Reverse_Engineering/blob/master/imu_sensor_notes.md>

<https://github.com/elgoupil/joyconLib>

<https://www.youtube.com/watch?v=BQyZZARU3dQ>

endurance-> 60-75% FCM(fréquence cardiaque maximal) et on essaie de courir le plus longtemps possible

<http://www.cs13etoiles.ch/old/documents/theorie/Intervalles-exemples.pdf>