Rapport intermédiaire

Table des matières

[2. Problématique : 4](#_Toc14909826)

[3. Cahier des charges : 4](#_Toc14909827)

[4. Introduction 5](#_Toc14909828)

[5. Analyse des technologies exploitables 5](#_Toc14909829)

[Les détecteurs de mouvements 5](#_Toc14909830)

[La Kinect : 5](#_Toc14909831)

[Le protocole FTMS 6](#_Toc14909832)

[Shimmer3 IMU 7](#_Toc14909833)

[La Wiimote 8](#_Toc14909834)

[Joy-con 9](#_Toc14909835)

[Phidgets 9](#_Toc14909836)

[Les outils de création de jeux 10](#_Toc14909837)

[Unity 10](#_Toc14909838)

[C++ 10](#_Toc14909839)

[Java 10](#_Toc14909840)

[Autres 10](#_Toc14909841)

[Bluetooth 10](#_Toc14909842)

[Moteur 3D de Marc Hugi et Christian Ammann 11](#_Toc14909843)

[Choix des technologies 11](#_Toc14909844)

[6. Concepts de prototype 12](#_Toc14909845)

[Contraintes & démarche 12](#_Toc14909846)

[Concept du jeu 14](#_Toc14909847)

[Jeu solo 15](#_Toc14909848)

[Objectifs 16](#_Toc14909849)

[Jeu multi 16](#_Toc14909850)

[Serveurs mondiaux 18](#_Toc14909851)

[La concurrence 20](#_Toc14909852)

[Kinomap 20](#_Toc14909853)

[Run social 21](#_Toc14909854)

[IConnect 21](#_Toc14909855)

[Exercube 21](#_Toc14909856)

[Zwift 22](#_Toc14909857)

[Arcade Fitness 23](#_Toc14909858)

[Fitness gaming 23](#_Toc14909859)

[Post analyse du concept 24](#_Toc14909860)

[7. Conception & implémentation du prototype 24](#_Toc14909861)

[Boucles de jeux 24](#_Toc14909862)

[Méthode naïve 24](#_Toc14909863)

[Boucle avec délayage 25](#_Toc14909864)

[Boucle avec étape à taille variable 25](#_Toc14909865)

[Boucle à délayage sur la logique 26](#_Toc14909866)

[Scrolling vertical infini 27](#_Toc14909867)

[Objets et collisions 27](#_Toc14909868)

[Méthode naïve 27](#_Toc14909869)

[Broad Phase 28](#_Toc14909870)

[Résolution de collision 29](#_Toc14909871)

[Rochers & Cœurs 29](#_Toc14909872)

[Boucliers 30](#_Toc14909873)

[Vitesse de scrolling 30](#_Toc14909874)

[Mesureurs d’efforts 31](#_Toc14909875)

[EffortCalculator 31](#_Toc14909876)

[Mesureur souris 32](#_Toc14909877)

[Mesureur d’accélération 33](#_Toc14909878)

[Mesureur de fréquence 34](#_Toc14909879)

[Mesureur de vitesse de course 36](#_Toc14909880)

[Déplacements horizontaux 41](#_Toc14909881)

[Clavier 41](#_Toc14909882)

[Wiimote 42](#_Toc14909883)

[Joy-con 42](#_Toc14909884)

[Phidgets 42](#_Toc14909885)

[Programme sportif 42](#_Toc14909886)

[L’entraînement chez les professionnels 43](#_Toc14909887)

[Le programme temporel 44](#_Toc14909888)

[Gamification 46](#_Toc14909889)

[Les objectifs 46](#_Toc14909890)

[Le manager d’objectifs 46](#_Toc14909891)

[Les évaluateurs 48](#_Toc14909892)

[L’évaluateur sportif 48](#_Toc14909893)

[L’évaluateur de jeu 49](#_Toc14909894)

[Le mode deux joueurs 49](#_Toc14909895)

[L’interface Utilisateur 50](#_Toc14909896)

[Menu 50](#_Toc14909897)

[Menu JPanel 51](#_Toc14909898)

[Aide 51](#_Toc14909899)

[Création de programme sportif 53](#_Toc14909900)

[Le jeu 54](#_Toc14909901)

[La fin de partie 57](#_Toc14909902)

[Effets Sonores 57](#_Toc14909903)

[Le pattern Observer 58](#_Toc14909904)

[8. État actuel 58](#_Toc14909905)

[Prototype 58](#_Toc14909906)

[9. Bug connus 58](#_Toc14909907)

[Recommencer des parties 59](#_Toc14909908)

[Dépassement du bouclier 59](#_Toc14909909)

[Baisse de fréquence trop forte 59](#_Toc14909910)

[10. Problèmes rencontrés 59](#_Toc14909911)

[Valeurs erronées de la Shimmer3 59](#_Toc14909912)

[Calibration 59](#_Toc14909913)

[La Wiimote obsolète 59](#_Toc14909914)

[Conception seul 59](#_Toc14909915)

[11. Amélioration possible 60](#_Toc14909916)

[Mode réseau 60](#_Toc14909917)

[Factorisation 60](#_Toc14909918)

[Format d’image conservé 60](#_Toc14909919)

[Détecteur d’effort 60](#_Toc14909920)

[Base de données 60](#_Toc14909921)

[Login 60](#_Toc14909922)

[Machine Learning 60](#_Toc14909923)

[Homogénéiser le style de l’UI 61](#_Toc14909924)

[12. Remerciements 61](#_Toc14909925)

[13. Conclusion 61](#_Toc14909926)

[14. Table des illustrations 62](#_Toc14909927)

[15. Bibliographie 63](#_Toc14909928)

# 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

Ce projet va servir de point de départ au projet de Mr Grégory Vincent. Le but sera de vérifier la faisabilité d’un prototype reliant activité physique et interface graphique en y ajoutant du machine learning pour que l’application ce calibre selon son utilisateur actuel.

Le projet commençant à peine, il faudra définir nous-même le type de gamification, les limites de l’exploration ainsi que les priorités. Le tout en prenant en considération les moyens matériel et financier.

Le prototype final visera tout type d’utilisateurs et devra donc être aussi intuitif que possible. Concrètement, le prototype sera écrit en Java et utilisera une Shimmer3 et un Joy-con pour la détection de mouvement.

# 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)

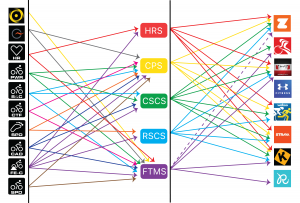


Figure 3 fonctionnalité -> services Bluetooth -> applications

### 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

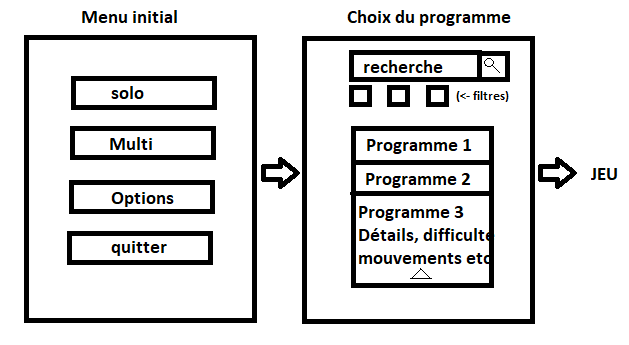
Finalement pour imposer un peu de variété, nous ajouteront des changements d’arrière-plan. Chaque arrière-plan possèdera son propre ensemble de 3 mouvements/positions pour ainsi forcer le sportif à travailler divers muscles pendant une séance. Grâce à ces changements nous pourrons de manière transparente 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 :

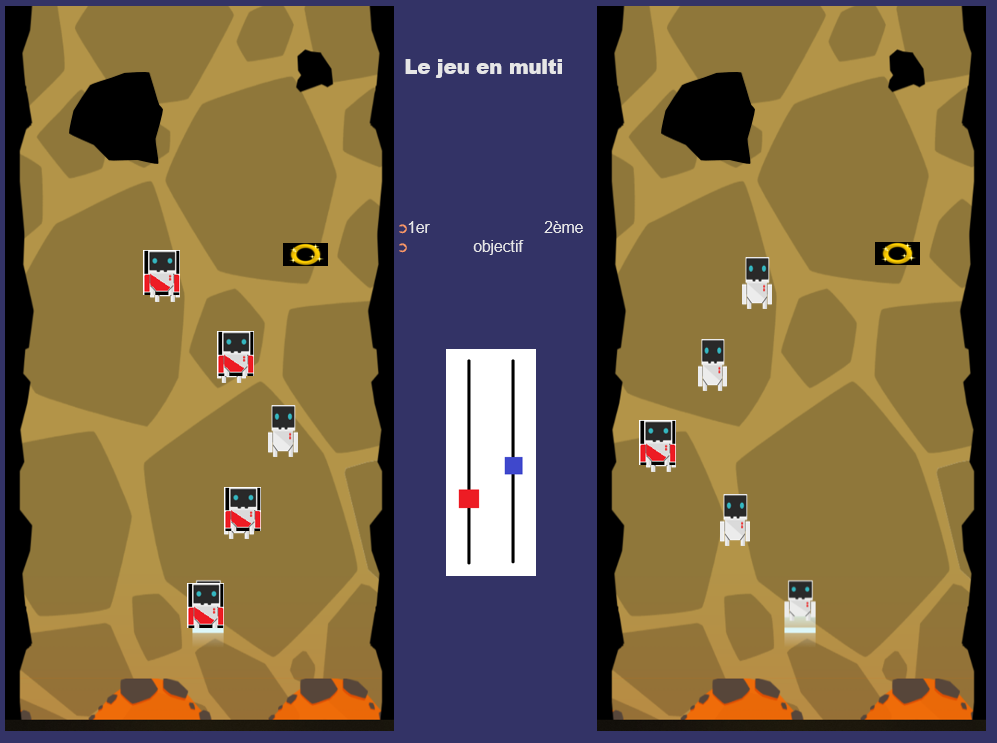


Figure Jeux en compétitif

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 :

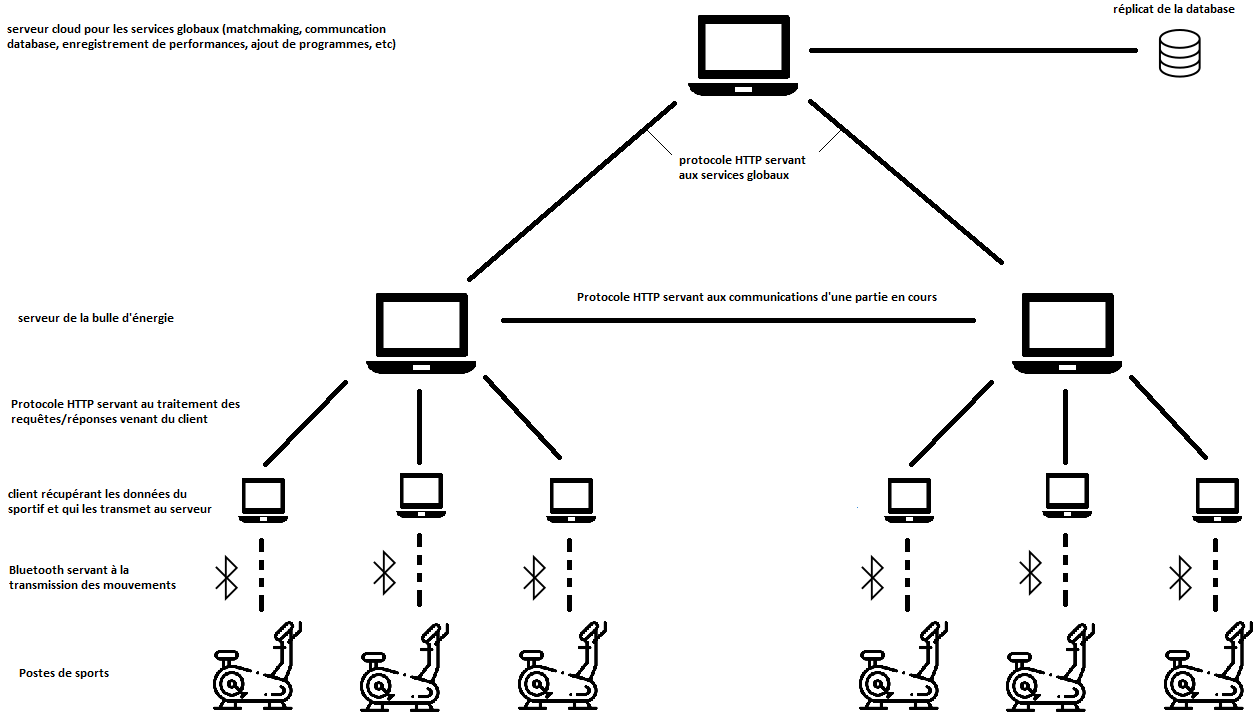


Figure architecture globale

## 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

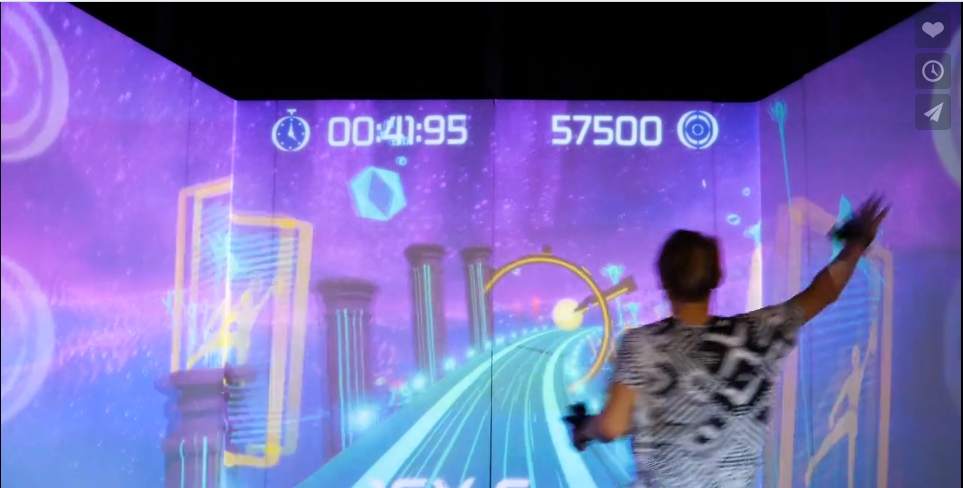


Figure Exercube

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

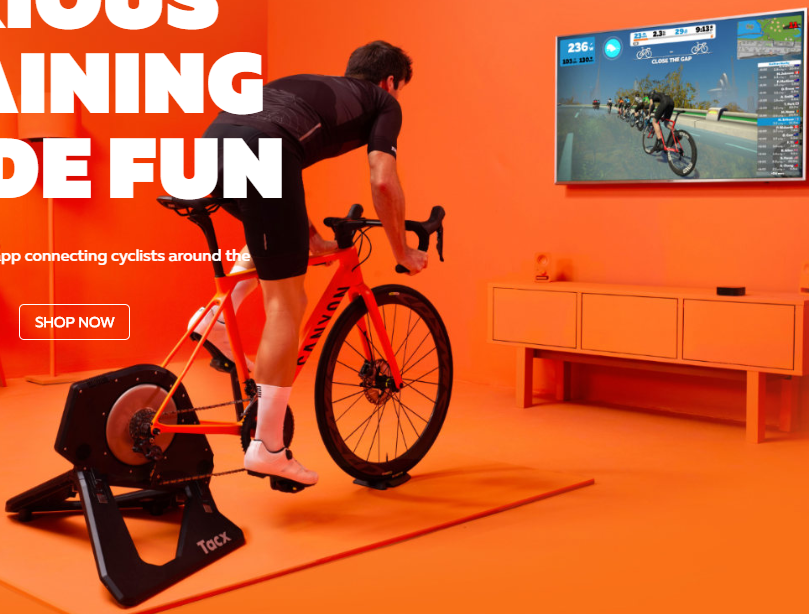


Figure Zwift

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

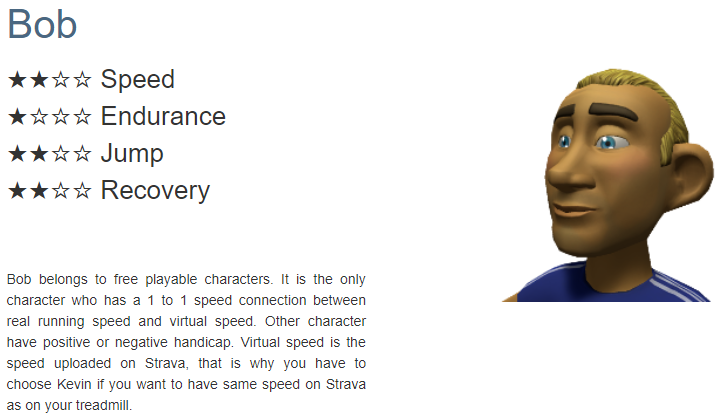


Figure Bob personnages jouable dans Arcade Fitness

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 :

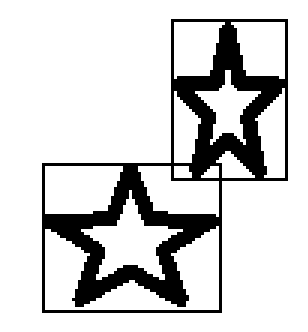


Figure détection fausse d'une collision avec une méthode naïve

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.

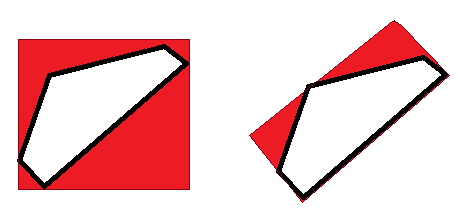


Figure aire d'erreur, méthode naïve contre axes alignés

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 17 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 :

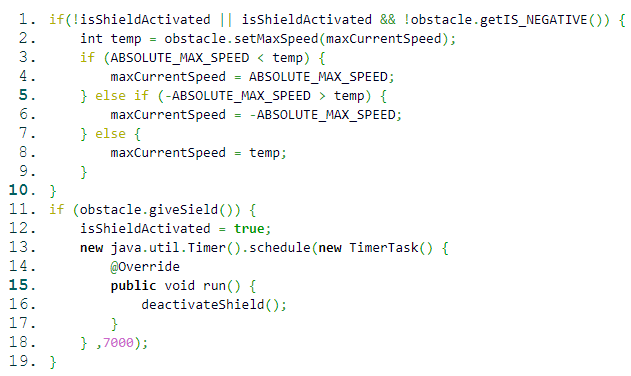


Figure resolveCollide méthode

## 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 :

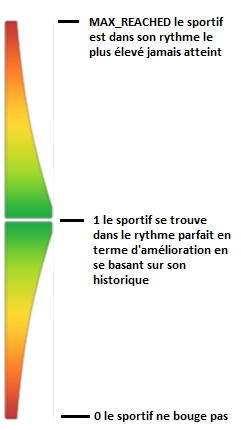


Figure Barre d'effort, en vert l'arrière-plan se déplace à vitesse maximum. En rouge l'arrière-plan ne se déplace pas.

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, puisque le détecteur possède des imprécisions au démarrage on obtient des valeurs trop élevées ce qui rend la deuxième partie de la barre impossible à remplir. J’ai donc décidé de pour l’instant réinitialiser la valeur de MAX\_REACHED à chaque démarrage du programme à 1.5 \* le rythme parfait.

Nous verrons aussi dans la partie « Programme sportif » qu’un entraînement est composé de plusieurs partie. Chaque partie va donc posséder son propre « rythme parfais » et en parallèle son propre MAX\_REACHED.

### 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 :

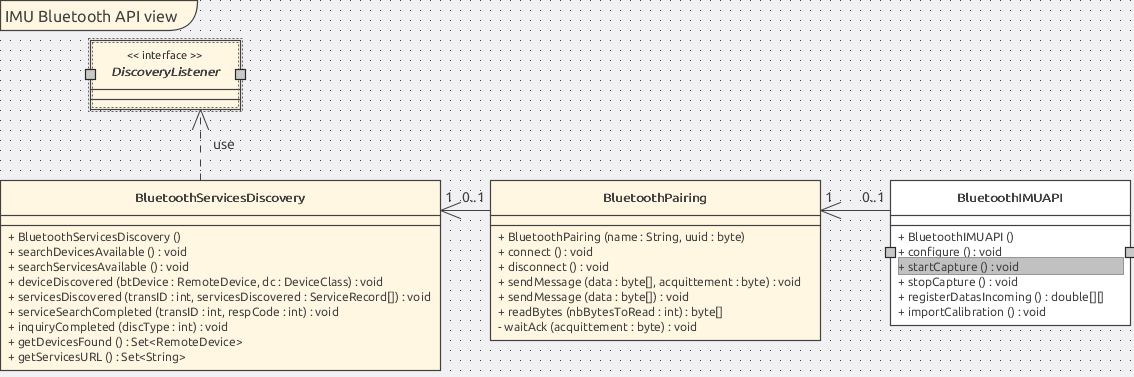
La première implémentation naïve pour un mesureur avec accéléromètre a été d’utiliser une liste chaînée. À chaque nouvelle donnée reçue, je l’ajoute à la fin de la liste et je supprime le premier élément (donc le plus anciens). Le calcul de l’effort ce faisait ensuite en parcourant la liste, en faisant 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 20 Vue de la hiérarchie de classe pour la connexion Bluetooth avec la Shimmer3

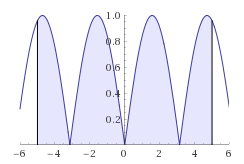


Figure Résultat fictif de l'implémentation naïve si le mouvement était parfait

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 :

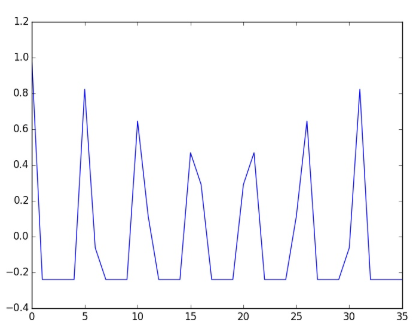


Figure résultat fictif de fast Fourier transformation. Chaque pic serait une fréquence plus ou moins présente dans nos données d’accélération.

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 :

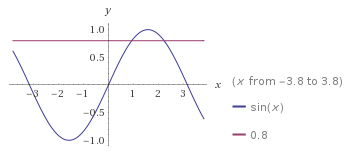


Figure Résultat fictif de la méthode par seuil. Ici en x = 1 on compterai un cycle.

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 :

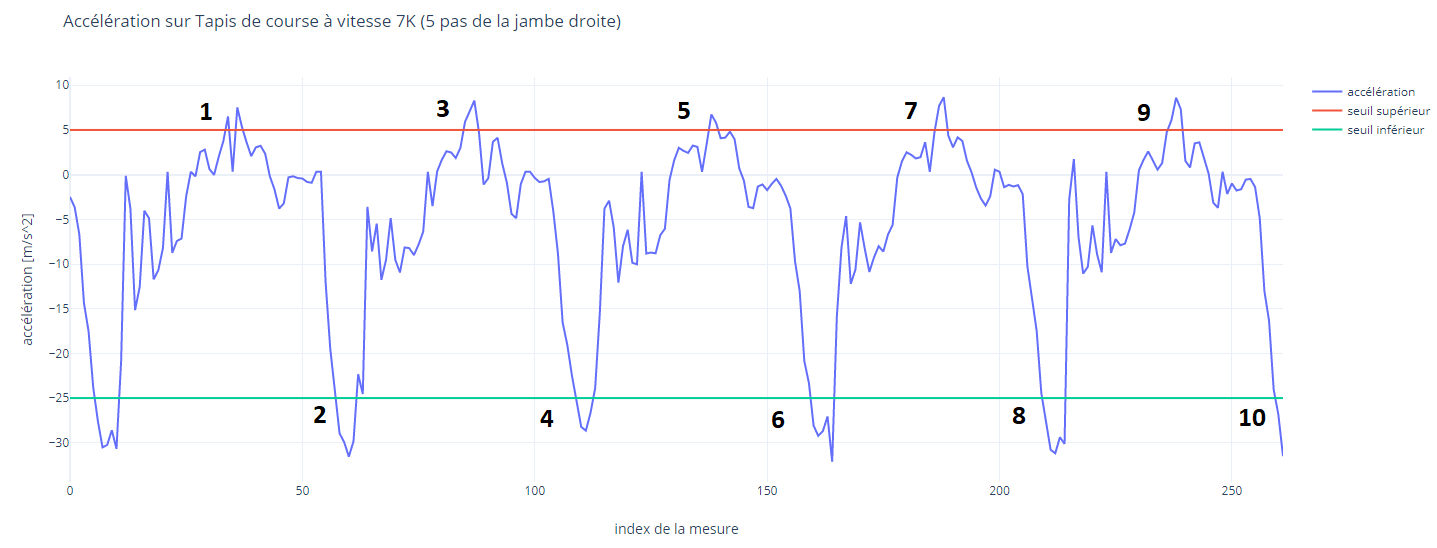
Le deuxième problème qu’il a fallut résoudre est de déterminer la valeur de ce seuil. L’idée de base à été de se dire que nous allions prendre la valeur maximum, la valeur minimum et que nous placerions les deux seuils à un certain pourcentage entre ces deux valeurs. La formule donne par exemple pour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

Figure 25 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 26 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.

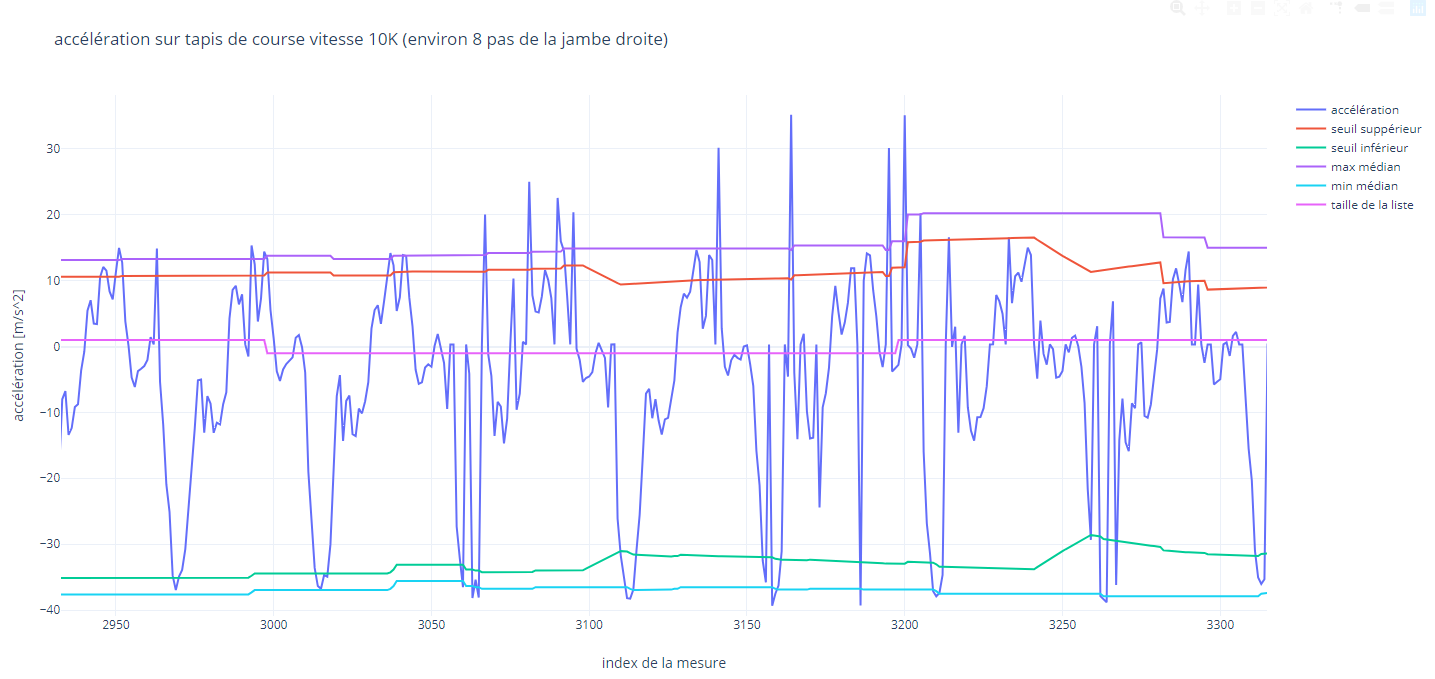
Nous allons donc aussi ajouter un système prévoyant le prochain cycle afin de baisser manuellement le seuil si nous n’arrivons plus à détecter de cycle pendant trop longtemps. En somme, nous détectons un premier cycle qui prend x ms pour parvenir. On s’attend donc à détecter un autre cycle dans environ 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 27 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 :

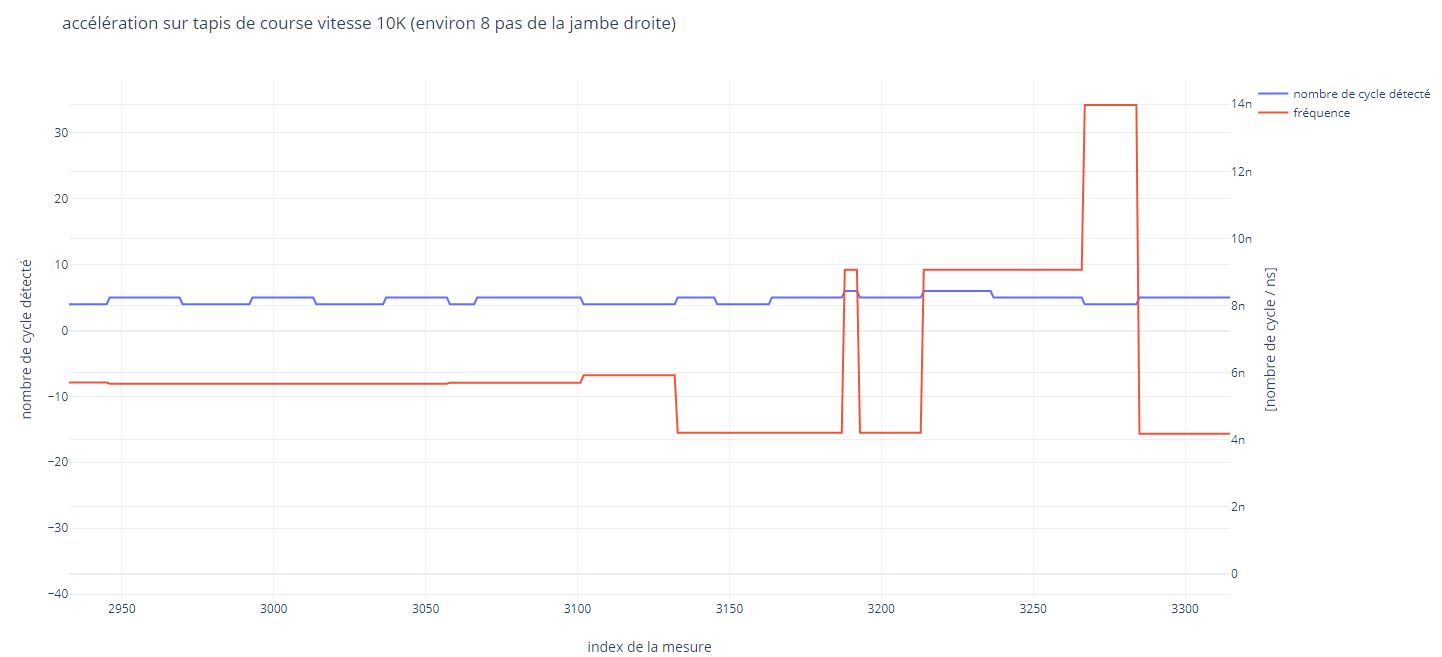


Figure 28 détection de la fréquence actuellement encore instable

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.

Après investigation et grâce à l’aide de l’assistant, nous avons découvert que les données erronées venaient d’une mauvaise utilisation de ma part de la méthode read de la classe DataInputStream. En effet, bien que bloquante, la méthode read peut ne pas retourner toutes les bytes demandées avec les paramètres et retourne le nombre de byte qu’elle a pu lire cette fois-ci. Après l’ajout d’un minimum de logique pour gérer ce cas, les graphiques sont beaucoup plus propres.

Désormais, il ne reste plus qu’à rendre le changement de fréquence moins brusque. J’ai donc ajouté une variable « currentFrequence » qui sera la frequence sur laquelle on se basera pour indiquer à la super classe notre effort actuel. En parallèle, nous faisons nos calculs de frequence momentanné et nous faisons tendre currentFrequence vers la frequence momentanné trouvé. Voici le calcul :

La variable vitesseDeConvergence permet d’ajuster le poids de la nouvelle frequence sur le calcul. On peut voir ainsi sur le prochain graphique que la fréquence ne change plus par saut d’une valeur a l’autre mais de manière beaucoup plus fluide :

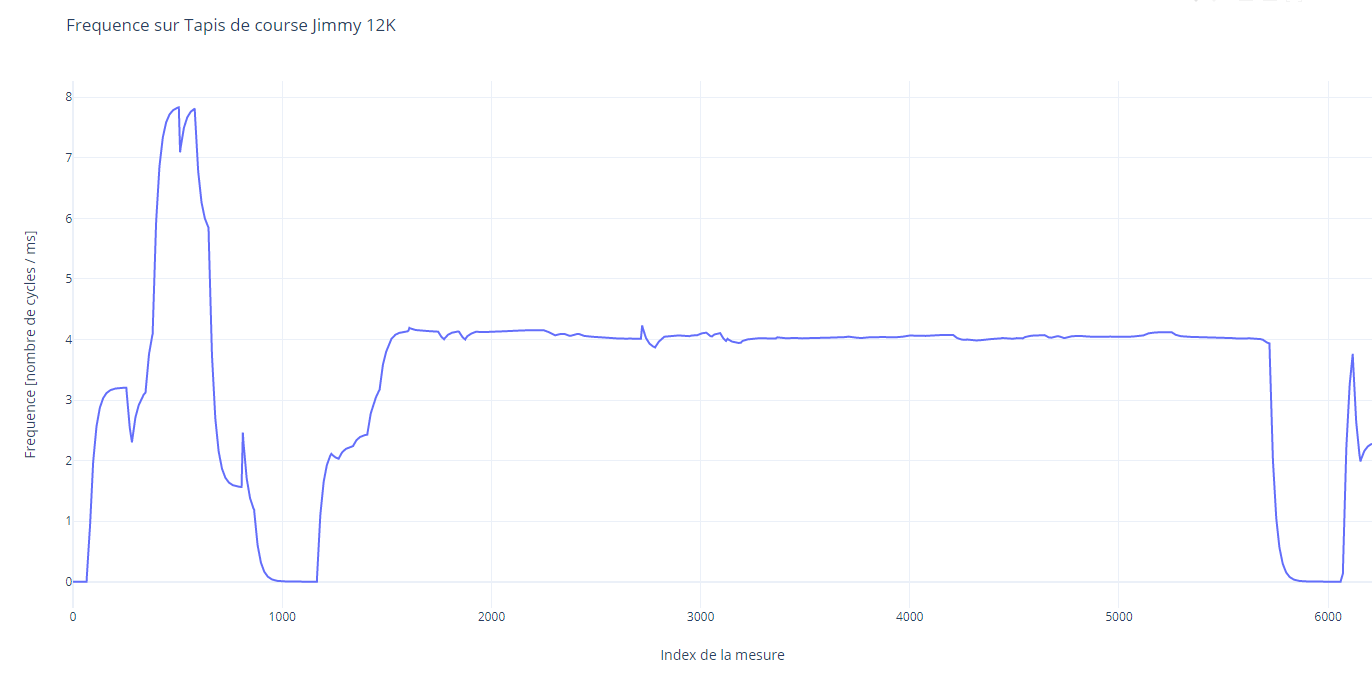


Figure frequence fluide

Désormais nous avons un détecteur de fréquence utilisable. Je l’ai calibré pour pouvoir être utilisé sur un tapis de course par moi-même. Donc un homme de 29 ans d’1m73. Il a fallu par exemple rallonger la taille de buffer de donnée car pour une vitesse très lente (3K) le mouvement de 2 pas rentre dans environ 300 cases du buffer.

La dernière fonctionnalité testée avec le détecteur d’effort était de considérer que l’effort n’est pas une fonction linéaire. Il a fallu réfléchir à comment mapper la fréquence des pas de course à un réel effort. Si on passe de 1 pas la minute à deux pas la minute, on double la fréquence mais pourtant nous ne sommes toujours pas fatigués. J’ai donc défini une méthode mappingFunction servant à faire la transposition. J’ai testé 3 fonctions mathématique.

La première était celle qui me semblait le plus correct. Une fonction en deux parties. De 0 à la fréquence visée puis de la fréquence visée jusqu’au maximum jamais atteint (MAX\_REACHED). Voici ce que donne la fonction si on considère la fréquence visée = 8 et MAX\_REACHED = 11 :

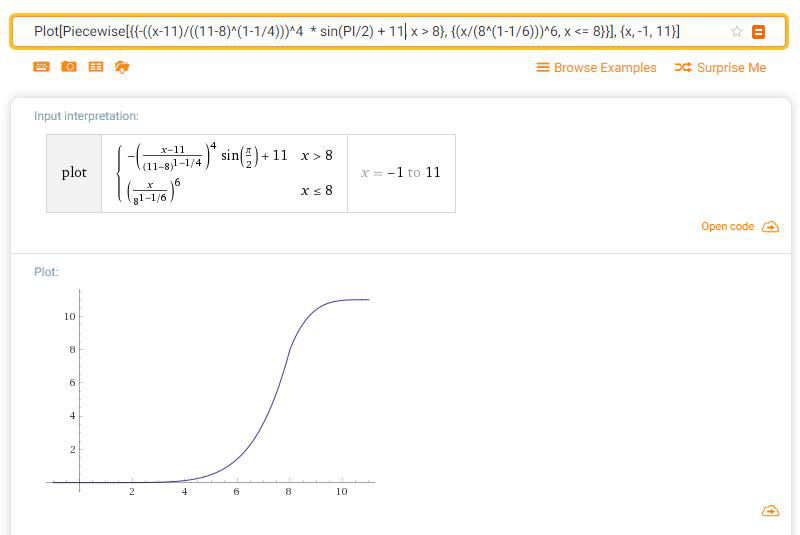


Figure fonction de mapping complexe

Malheureusement, les imprécisions du détecteur sont décuplées avec cette fonction autour de la valeur visée et donc expérimentalement n’est pour le moment pas viable. J’ai donc testé une fonction plus simple qui est une fonction linaire en plusieurs partie qui n’a pas donné de meilleur résultat. Je suis donc resté à une fonction de mappage linéaire simple ou x = y en attendant d’améliorer la précision des calculs. Néanmoins, les fonctions sont en commentaires dans le code, il suffira donc de les décommenter pour les utiliser.

## 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.

## Programme sportif

Maintenant que nous avons un mesureur d’effort fonctionnel, il va sans dire qu’il ne suffit pas de dire à l’utilisateur de rester à une fréquence qui pour lui est parfaite et de faire tourner le jeu pour que son entraînement soit efficace. De nombreuses études sont faites chaque année pour comprendre et améliorer l’efficacité des entraînements des sportifs de haut niveau. Nous allons voir dans ce chapitre quels moyens utilisent les sportifs pour parfaire leurs entrainements afin d’offrir la même logique dans notre prototype.

### L’entraînement chez les professionnels

Afin de simplifier le problème, j’ai commencé par regarder les entraînements des coureurs. La plupart des forums discutant d’entraînements de course se base sur la VMA ou la FCM qui sont deux estimateurs.

VMA pour vitesse maximale aérobie est la plus petite vitesse de course consommant le maximum d’oxygène pour un coureur. Connaître sa vitesse aérobie permet ensuite de calibrer son entraînement en pourcentage de cette vitesse. Comme le montre l’image suivante, la VMA est une vitesse se situant entre le rythme 5km et le rythme en côte :

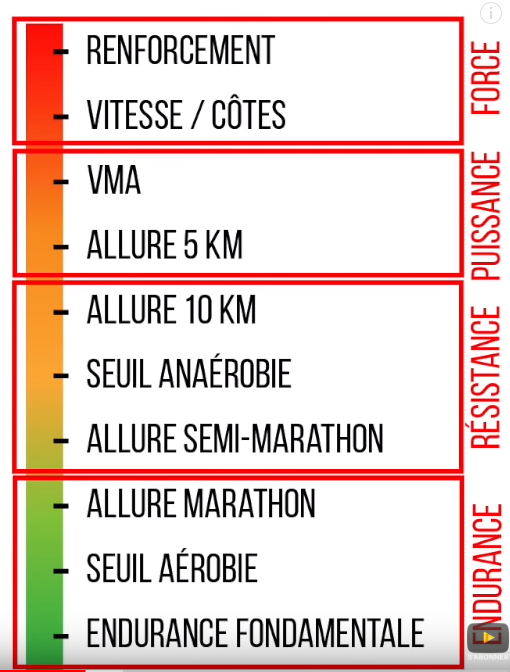


Figure hiérarchie des vitesses de course

FCM pour fréquence cardiaque maximale est la fréquence maximale que peut atteindre le cœur lors d’un effort maximum pour un individu. On pourrait donc utiliser cette fréquence pour mesurer l’effort de la personne et cette fois aussi varier le pourcentage requis pour varier le type d’entraînement.

La différence entre VMA et FCM est que la VMA est spécifique au sport donné et donc plus précise. Tandis que la FCM permet d’être générique.

J’ai donc choisi d’utiliser le système de VMA et donc d’implémenter une fréquence « optimale » pour chaque type de mouvement. On pourra ensuite calibrer chaque mouvement en indiquant la VMA de la personne selon le mouvement quel fait.

Une fois la VMA connu tous les entraîneurs vont décrire un programme sportif en indiquant :

* Une durée
* Une intensité (en pourcentage de la VMA ou équivalent)
* Un mouvement

Et vont répéter ce schéma plusieurs fois pour décrire chaque partie du programme sportif. On peut retrouver plusieurs programmes de course « par intervalle » sur ce site : <http://www.cs13etoiles.ch/old/documents/theorie/Intervalles-exemples.pdf>

Chaque entraînement aura pour but d’améliorer une ou plusieurs caractéristiques. On n’utilisera donc pas le même entraînement pour améliorer son endurance ou pour améliorer sa vitesse de pointe. La règle de base est plus on court lentement, plus on entrainera son endurance, mais plus il faudra donc courir longtemps. À l’inverse plus on courra vite plus on améliorera sa vitesse de pointe et plus il faudra fractionner son entraînement avec des sprints et des pauses.

Il est aussi important de noter que souvent les coureurs débutants on tendance à surentraîner la « puissance ».

### Le programme temporel

Sur la base de ces quelques connaissances, j’ai donc implémenté une hiérarchie de classe permettant de créer un programme sportif. Un Workout comme je l’appelle dans le code est composé de WorkoutParts qui possède les trois propriétés mentionnées précédemment (un mouvement, une intensité, une longueur). J’utilise le terme de longueur qui est plus générique intentionnellement car il pourrait bien s’agir autant d’une durée temporelle que spatial voir autre chose. Voici l’UML qui en découle :

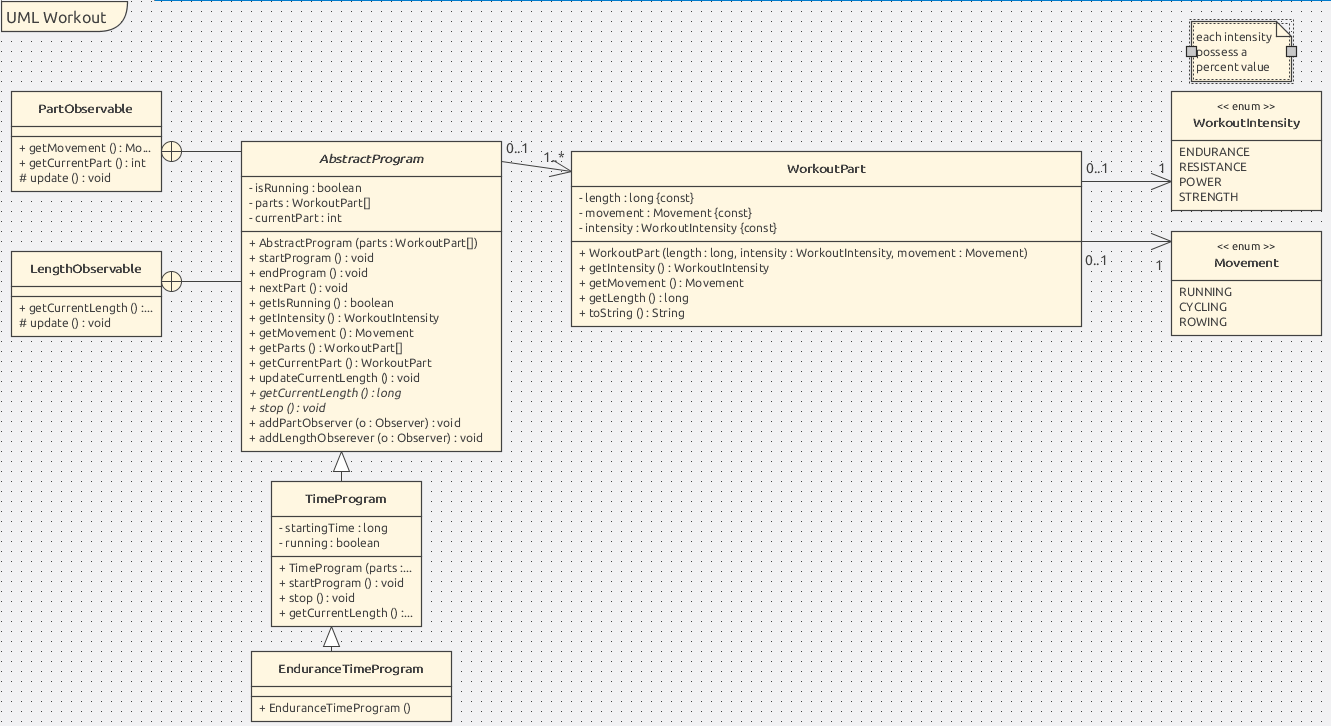


Figure UML des programmes sportifs

La classe AbstractProgram va servir de modèle pour que les sous-classes puissent définir leurs programmes sportifs. Les sous-classes doivent juste définir ce qu’est une « longueur » en redéfinissant les deux méthodes abstraites getCurrentLength et stop. Puis elles peuvent définir le comportement en cours de programme en redéfinissant startProgram. Enfin elles doivent aussi spécifier les diverses parties en donnant un tableau à la construction de WorkoutPart.

Les classes externes vont pouvoir observer le comportement du programme en souscrivant aux deux Observable fournit par l’interface de AbstractProgram. PartObservable va notifier chaque changement de partie tandis que LengthObservable va notifier à chaque instant ce qui est plus coûteux.

Le principal intérêt est que la GameLoop puisse souscrire à ces changements de partie et donc modifier le comportement du détecteur d’effort et modifier l’intensité et donc la fréquence visée.

## Gamification

Maintenant qu’il est possible de mouvoir le background au rythme d’un programme sportif, il faut surtout captiver l’utilisateur pour l’amuser et lui faire oublier ces futures courbatures. Nous allons donc voir dans cette partie quels seront les objectifs du sportif qui désormais est surtout un joueur.

### Les objectifs

6 types d’objectifs sont implémenté et fonctionnels.

* Éviter N rochers
* Détruire avec un bouclier N rochers
* Rester dans le rythme de course parfait pendant x secondes
* Zigzaguer entre N rocher (c’est-à-dire passer à gauche d’un rocher puis à droite du suivant etc)
* Ramasser un boost de vitesse lorsqu’un autre est déjà actif
* Avancer de N pixels le background

Tous ces objectifs sont de simples compteurs avec un noms. Une classe Rule construite à partir d’un nom et d’une valeur à atteindre commence simplement à compter à partir de 0. Elle offre des méthodes d’incrément/soustraction/reset afin de manipuler le compteur du dit objectif. Une fois la valeur de l’objectif atteint elle le signal en mettant un boolean à true.

### Le manager d’objectifs

Afin d’initialiser et manipuler les divers objectifs sans surcharger la classe GameEngine, une classe RulesManager à été écrite afin de manipuler facilement les règles. Ainsi le GameEngine doit juste indiquer dans sa logique à quel moment tel ou tel compteur doit être modifié.

Typiquement, le GameEngine appellera par exemple la méthode addObjectifs(RUN\_N\_PIXELS, speed); qui permet d’ajouter speed à la valeur du compteur de la règle RUN\_N\_PIXELS. Le manager se chargera lui-même de savoir si ce compteur est actuellement actif ou non et si l’objectif est réussi ou non. Un seul objectif est actif à la fois et il est pris aléatoirement parmi les objectifs existants créé par le RulesManager à l’initialisation de la partie.

De plus, deux classe interne à RulesManager servent d’observable soit pour être notifié à la réussite d’un objectif, soit pour être notifié à la modification de l’état d’un objectif. Le second observable est donc plus lourd. Voici ce que donne cette hiérarchie en UML :

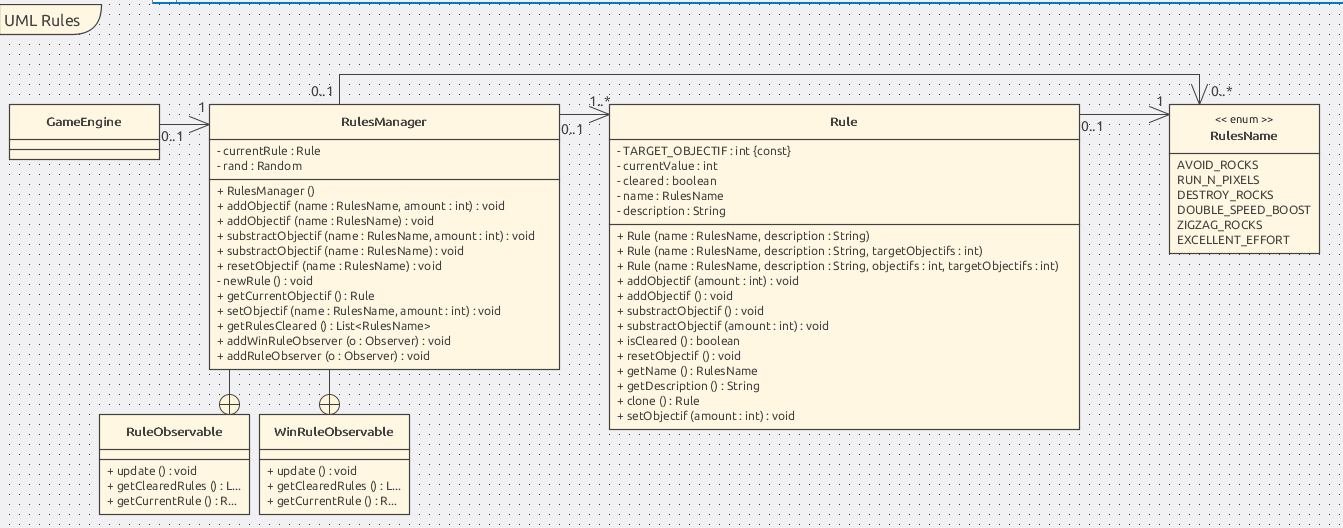


Figure UML Gamification

## Les évaluateurs

Afin de rendre l’application impactante, il est important de donner à l’utilisateur un feedback sur son expérience. Nous avons déjà des feedback sensoriel grâce aux déplacements du personnage à ces collisions avec le décor mais il nous manque encore beaucoup de feedback notamment au niveau de l’interface graphique que nous verront plus tard. Mais tout d’abord, je veux parler de l’évaluation de son niveau sportif et de l’évaluation de son niveau de jeu. J’ai décidé de bien séparer les deux afin que chaque utilisateur puisse décider ce qu’il souhaite prioriser lors d’une séance.

### L’évaluateur sportif

L’évaluation sportive se fera en cours de jeu. A chaque fois que l’utilisateur termine une partie de son programme sportif, il sera évalué au moyen d’un smiley. S’il a garder un rythme moyen correspondant à l’intensité du programme alors le smiley sera un smiley vert content. Sinon, plus il s’éloigne, plus le smiley sera triste. Voici un tableau des résultats d’évaluation possible :

|  |  |  |
| --- | --- | --- |
| **appréciation** | **Pourcentage d’erreur accepté** | **Icone reçue** |
| En cours | - |  |
| excellente | <=20 |  |
| Bonne | <=40 |  |
| moyenne | <=60 |  |
| mauvaise | >= 60 |  |

Afin d’évaluer le pourcentage d’erreur, il a fallut ajouter un peu de logique au programme. Ainsi la classe EvaluationPart permet de stocker un a un les valeurs d’effort retenu par le détecteur. Le total s’accumule et lorsque la partie sportive est terminé, il suffit de faire une simple division pour connaître son effort moyen. Une classe EvaluationManager permet de gérer quel EvaluationPart doit recevoir les données en cours de partie. Il permet aussi de décider quand calculer le résultat de l’évaluation. Enfin les 5 résulats d’évaluation possible son stocké dans un enum. Voici l’uml obtenu :

### L’évaluateur de jeu

Afin d’évaluer le niveau du joueur, un simple score à été ajouter. Pour augmenter le score, il existe deux moyens. Soit avancer le background via la course ou l’activité physique que nous avons choisi. Soit via les objectifs. Chaque objectif possède une valeur de score récupérable via son enum RulesName. En fin de partie chaque score d’objectif réussit est ajouté au score de déplacement pour donner un score total final.

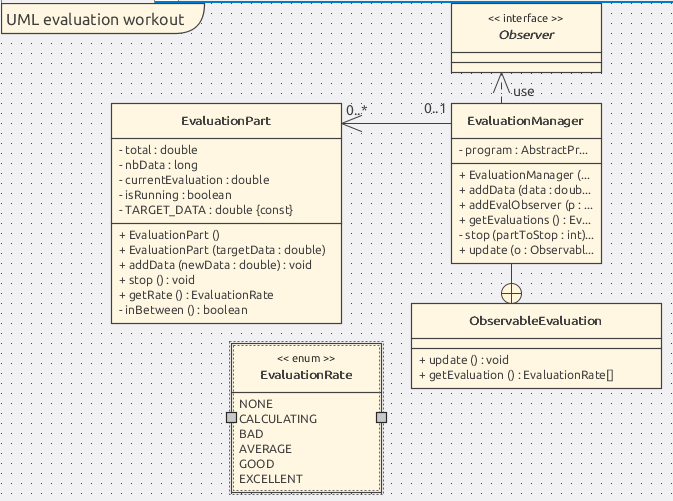


Figure uml évaluation sportive

La classe EvaluationManager est donc notifié à chaque enregistrement d’effort. Elle se charge d’indiquer à la bonne EvaluationPart le nouvel effort. Et elle est aussi notifiée lorsqu’on change de partie. Elle peut donc indiquer lors d’un changement de partie d’évaluer la partie terminée et elle notifie ces observateurs que l’état de l’évaluation à changé.

## Le mode deux joueurs

Le mode deux joueurs n’en est pour l’instant qu’au début de l’implémentation. L’ajout d’un personnage à simplement été fait en transformant la logique de GameEngine en changeant l’attribut Character en Character[]. Il a ensuite suffi d’étendre de la classe GameEngine en GameEngineDuo et de modifier une méthode. En effet, la méthode setSpeed doit désormais prendre en compte la distance entre les joueurs pour simuler un effet d’aérodynamisme. Le calcul est un simple calcul de dispersion. Plus la dispersion est grande, plus je ralenti la vitesse. Voici la formule de ralentissement :

Graphiquement on obtient une courbe descendant lentement vers 0 :

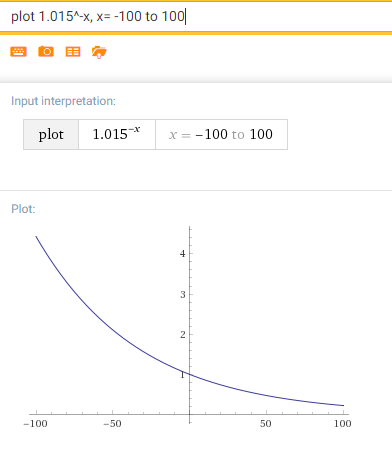


Figure graphique du ralentissement de l’aérodynamisme

La valeur 1.015 à été obtenu expérimentalement pour donner une bonne sensation de ralentissement à deux joueurs.

## L’interface Utilisateur

Maintenant que toutes les pièces du puzzle ont été créé, il ne reste plus qu’à assembler le tout à travers une interface graphique intuitive. L’interface à été écrite rapidement sur les 2 dernières semaines du projet. C’est donc probablement la partie qui manque le moins de défauts. Malheureusement, c’est aussi une des parties les plus importantes sans quoi l’utilisateur ne peut comprendre ce qu’il est en train de faire. J’ai divisé les classes créées pour l’UI en deux catégories. Les classes représentant une page entière ou une grosse partie de celle-ci et les composants que j’ai dû redéfinir pour les spécialiser. Par exemple, la classe MenuPanel représente la page affichée au lancement de l’application. Tandis que la classe Slider est un slider dessinant deux images superposées pour simuler une barre de chargement.

Nous allons voir dans ce chapitre les diverses pages dans lesquels l’utilisateur peut naviguer, quelles sont ces possibilités sur chacunes d’elles et enfin comment est construit la page par derrière les décors.

### Menu

La classe Menu étend la classe JFrame c’est le point d’entrée du programme où il est possible de naviguer d’une page à l’autre. Toutes les pages que nous allons voir, vont-elles étendre de JPanel et prendront comme parent cette fameuse classe Menu. Connaissant leurs parents, ils pourront ainsi appeler directement ces méthodes pour que lors d’un clique sur un panneau annexe on puisse tout de même passer d’une page à l’autre. C’est aussi cette classe qui est en charge de la création des divers composant nécessaire à une partie (Par exemple l’instanciation de la boucle de jeu et des détecteurs). Comme amélioration possible, il faudrait extraire cette deuxième fonction dans une classe GameManager.

### Menu JPanel

En plus de la classe Menu, une des pages étendant de JPanel s’appelle MenuPanel et représente le menu visible dès le lancement du programme. Voici un visuel (les chiffres rouges n’existe pas):



Figure menu de départ

Le layout utilisé est une BoxLayout qui va empiler les éléments de haut en bas. Afin de créer l’espacement entre les boutons, des Box vide ont été ajouté entre chaque bouton. J’imagine qu’une autre manière de faire aurait été de définir un padding sur les boutons.

Les boutons « play » permettent de lancer une partie avec le programme sportif actuellement défini. Le bouton « create workout » permet de définir se programme et le bouton « how to play » permet de comprendre comment jouer, enrobé par la petite histoire de Buddy. Et « quit » quitte proprement l’application.

### Aide

La page d’aide accessible via le bouton « how to play » permet de lire des explications sur le jeu, l’histoire de Buddy le petit robot et plus ou moins ce que je suis en train d’expliquer dans ce chapitre. Le tout afin de pallier aux imperfections de l’UI actuel. C’était pour moi un impératif à avoir tant que l’UI n’est pas parfaite et que chaque action de l’utilisateur n’est pas instinctive. Voici un visuel :

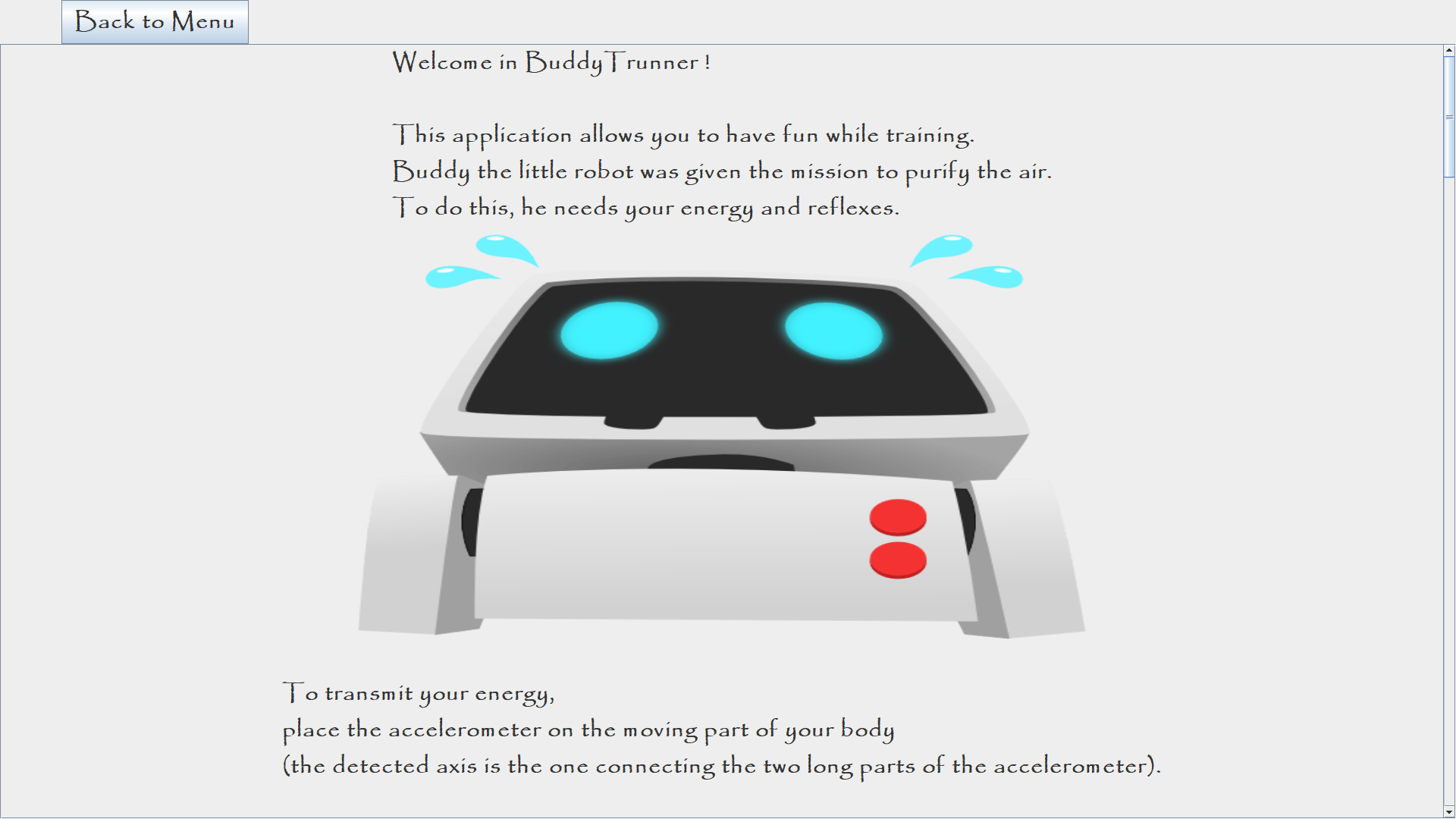


Figure menu d'aide

Ici deux BoxLayout ont été nécessaire. Le premier pour stocker le bouton de retour et le JScrollPane. Un JScrollPane permet de stocker des composants sa propre gestion de layout. Malheureusement, le layout par défaut d’un JScrollPane choisissait de stocker horizontalement mes labels. J’ai donc défini un JPanel avec un Boxlayout vertical pour stocker des labels. Les divers textes et images ne sont donc qu’une suite de JLabel stocké verticalement. Les images sont en fait les icones du label concerné mais placé judicieusement en dessous du texte pour donner l’impression d’en être séparé.

Voici ce que donne la même image vue avec les layouts :



Figure menu d'aide, point de vue avec layouts

Rien de spécialement intéressant à ajouter pour les composant puisque le bouton « back to menu » revient simplement au menu vu précédemment et le reste n’est qu’un affichage scrollable.

### Création de programme sportif

Accessible depuis le bouton « create workout », le menu de création de programme sportif permet comme son nom l’indique de créer partie par partie, son propre programme sportif. Pour l’instant il ne permet que de définir le programme sportif actuel, mais il serait utile de stocker le programme créé dans une base de données afin de choisir parmi les créations lors de lancement d’une partie. Voici un visuel :

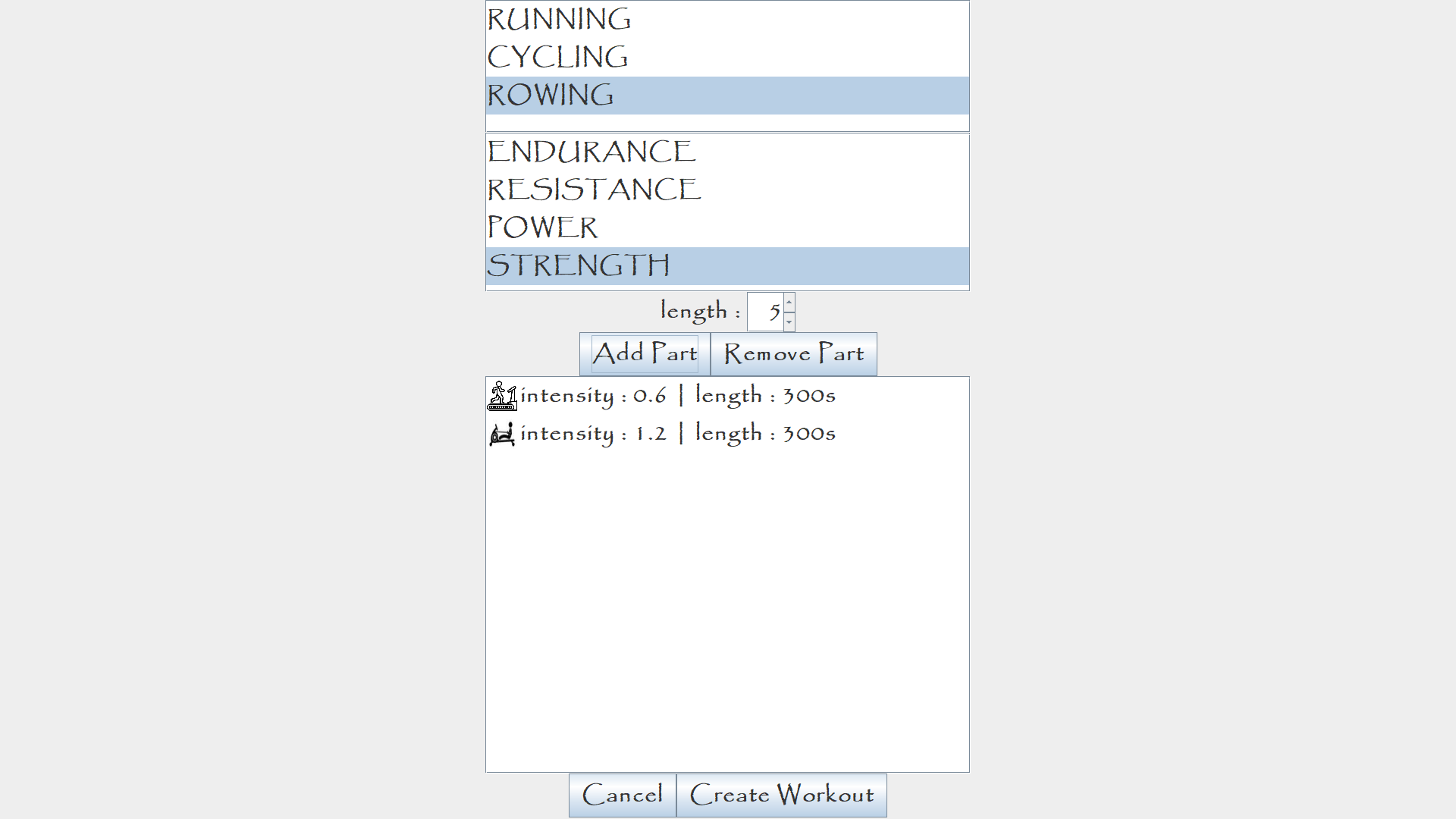


Figure menu de création de programme sportif

Ici, les composants sont stocké dans un BoxLayout vertical, cependant les boutons sont aussi stocké dans un BoxLayout horizontal qui lui empile les composant de gauche à droite. Les liste à choix ont été créé avec le composant JScrollPane. Cependant, un composant spécial à dû être créé pour afficher un WorkoutPart, ainsi ListProgramParts qui étend de la classe JLabel, implémente aussi l’interface ListCellRenderer. Ceci oblige de redéfinir la méthode getListCellRendererComponent qui indique au JScrollPane comment dessiner ce composant à l’écran. En l’occurance, on dessine l’icone correspondant au mouvement puis on écrit le texte du WorkoutPart. Voici à nouveau le même visuel avec les layouts :

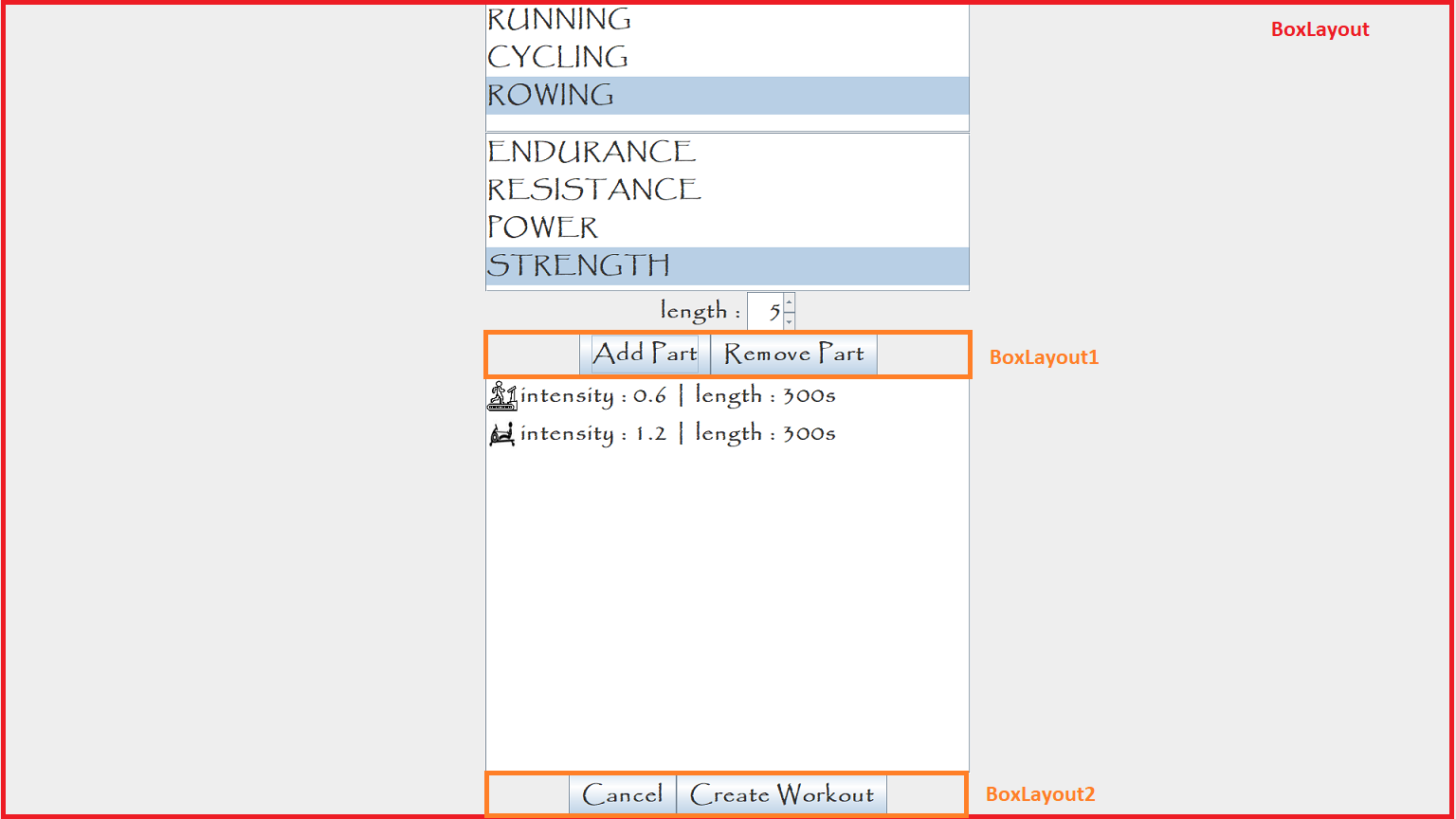


Figure menu de création de programme sportif avec layouts

Pour les composants, il fallait s’assurer qu’au moins un mouvement et une intensité était sélectionné. De plus, la longueur ne devait pas être négative et j’ai arbitrairement mis 60minutes comme valeur maximum. « Remove part » supprime la partie sélectionnée. « Cancel » retourne au menu, tandis que « create workout » met à jour le programme sportif et retourne au menu. J’ai considéré le retour en menu comme un feedback suffisant pour que l’utilisateur comprenne qu’il s’est passé quelque chose. Cependant, il faudrait ajouter un label pour indiquer la réussite de la création. Enfin toute les listes ne peuvent sélectionner qu’un élément à la fois.

### Le jeu

Lorsque l’utilisateur lance une partie, une page éphémère apparaît. Si tout se passe bien, les messages s’enchaînent, puis l’interface du jeu s’affiche. Sinon, un message d’erreur rouge remplace le texte informatif pendant quelques seconde pour laisser le temps à l’utilisateur de lire. La raison principale d’une erreur serait que les accessoires ne sont pas connectés ou non pairé à l’ordinateur. Puis, l’utilisateur est redirigé au menu pour lui laisser le temps de faire les manipulations avant une nouvelle tentative. Voici à quoi ressemble cette page de chargement.

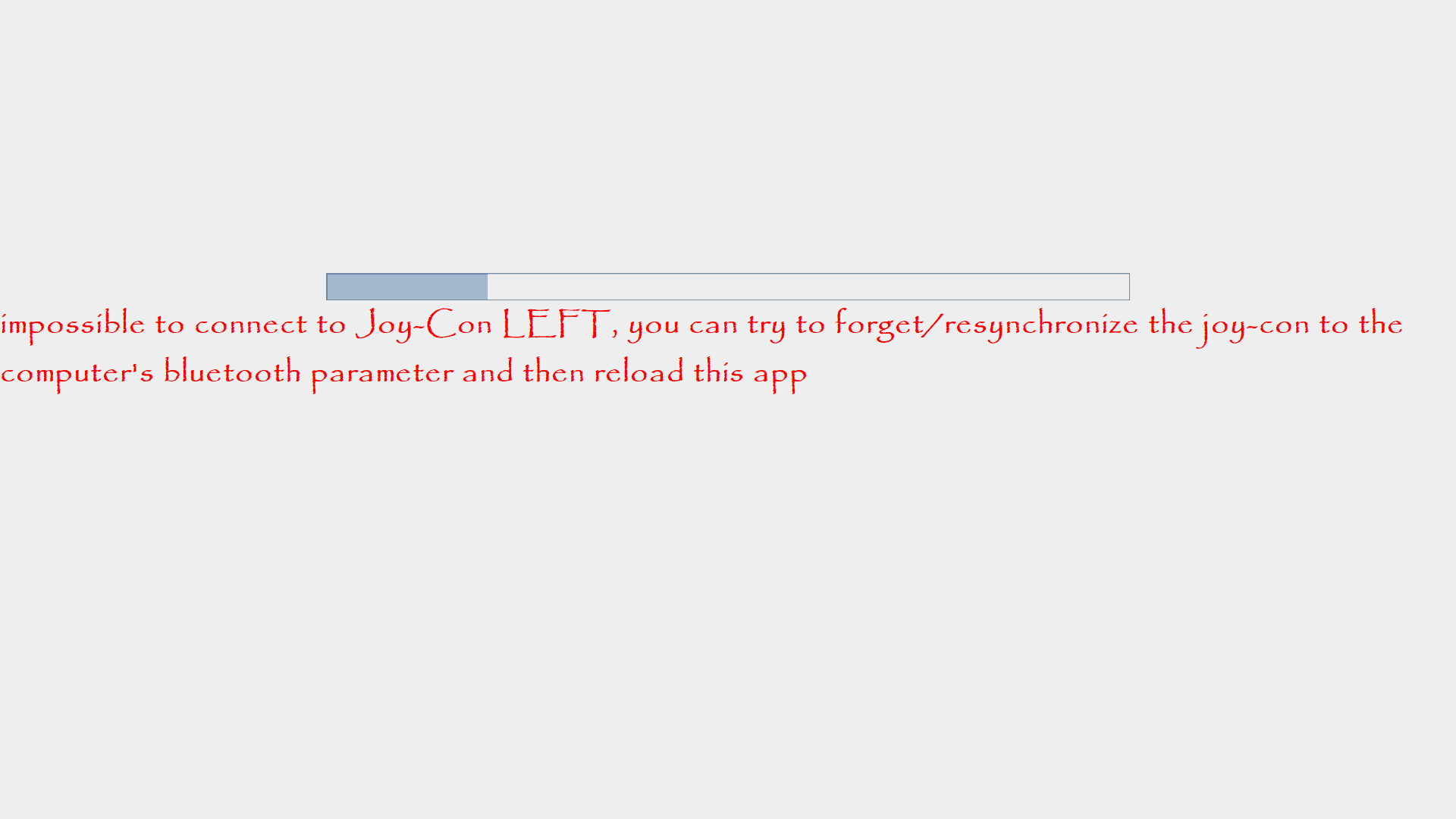
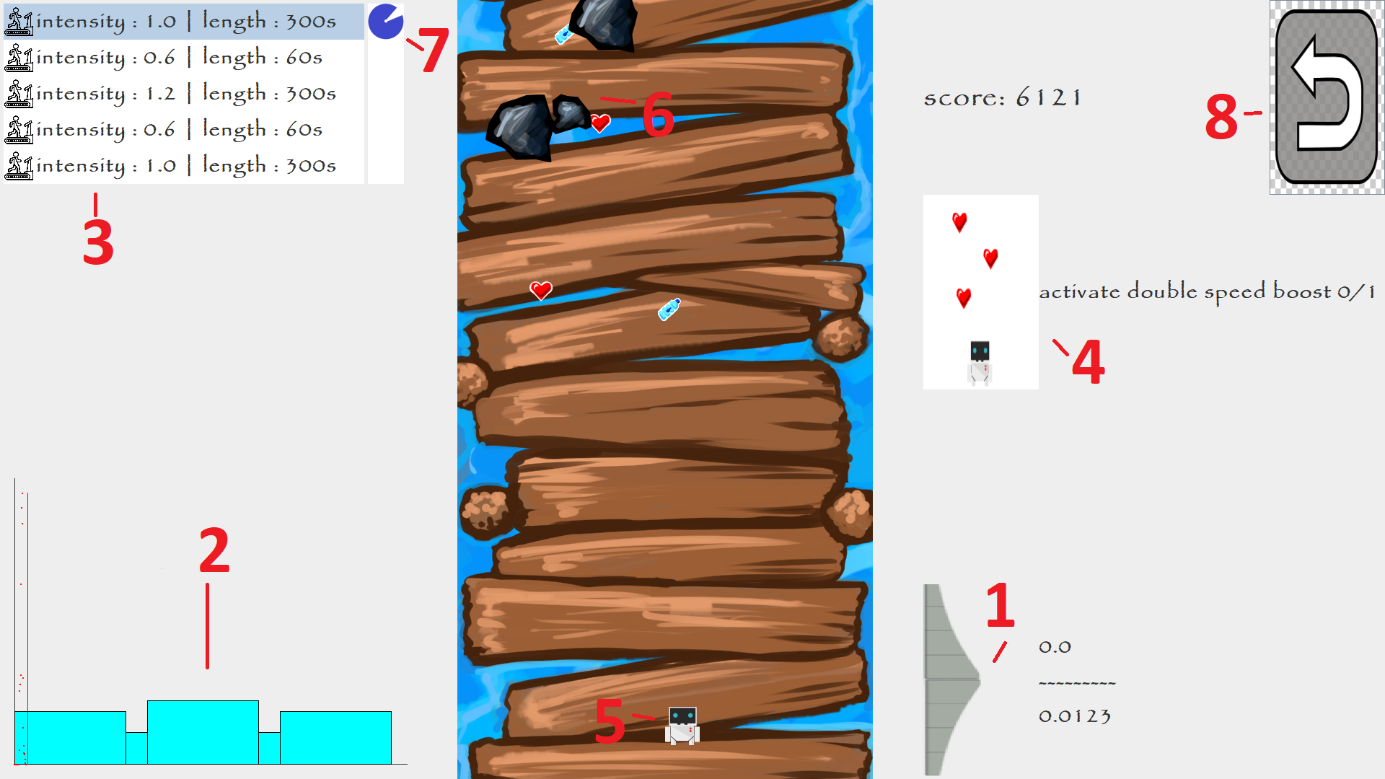


Figure menu de chargement d'une partie

Une fois la partie prête l’interface du jeu et le jeu sont lancé. Une amélioration possible serait de rajouter un bouton « prêt » pour laisser l’utilisateur gérer le début de la partie. Voici l’interface d’une partie à un joueur (les chiffre rouge ne sont pas présent):



L’interface est divisée en 3 parties. A gauche nous avons les informations relatives au sport, à droite les informations relatives à la gamification, puis au centre le jeu lui-même. Les composants sont :

1. La barre d’effort indiquant l’effort du sportif. Il faudra chercher à remplir la moitié de la barre (classe Slider).
2. Un Graphique résumant le programme sportif, indiquant la progression actuelle (barre vertical rouge) , puis des points indiquant la performance actuel seront rajouté au fur et à mesure de l’activité (classe GraphPanel).
3. Une liste résumant le programme sportif de manière textuelle et imagée (classe JListObserver + ListProgramParts).
4. L’objectif de jeu actuel, son image (un JPanel), sa description, ainsi que sa progression actuelle (0/1) (un JLabel).
5. Le robot Buddy que l’on contrôle.
6. Les divers obstacles/bonus qui nous arrivent dessus.
7. L’appréciation sportive (classe EvaluationPanel).
8. Un bouton de retour au menu pour pouvoir quitter à tout moment (un JButton).
9. Le score (un JLabel).

Cette interface est évidement la plus complexe. Tout d’abord, les 3 parties ont été divisé grâce à un BorderLayout qui permet de définir des zones. Ici, est, centre ouest.

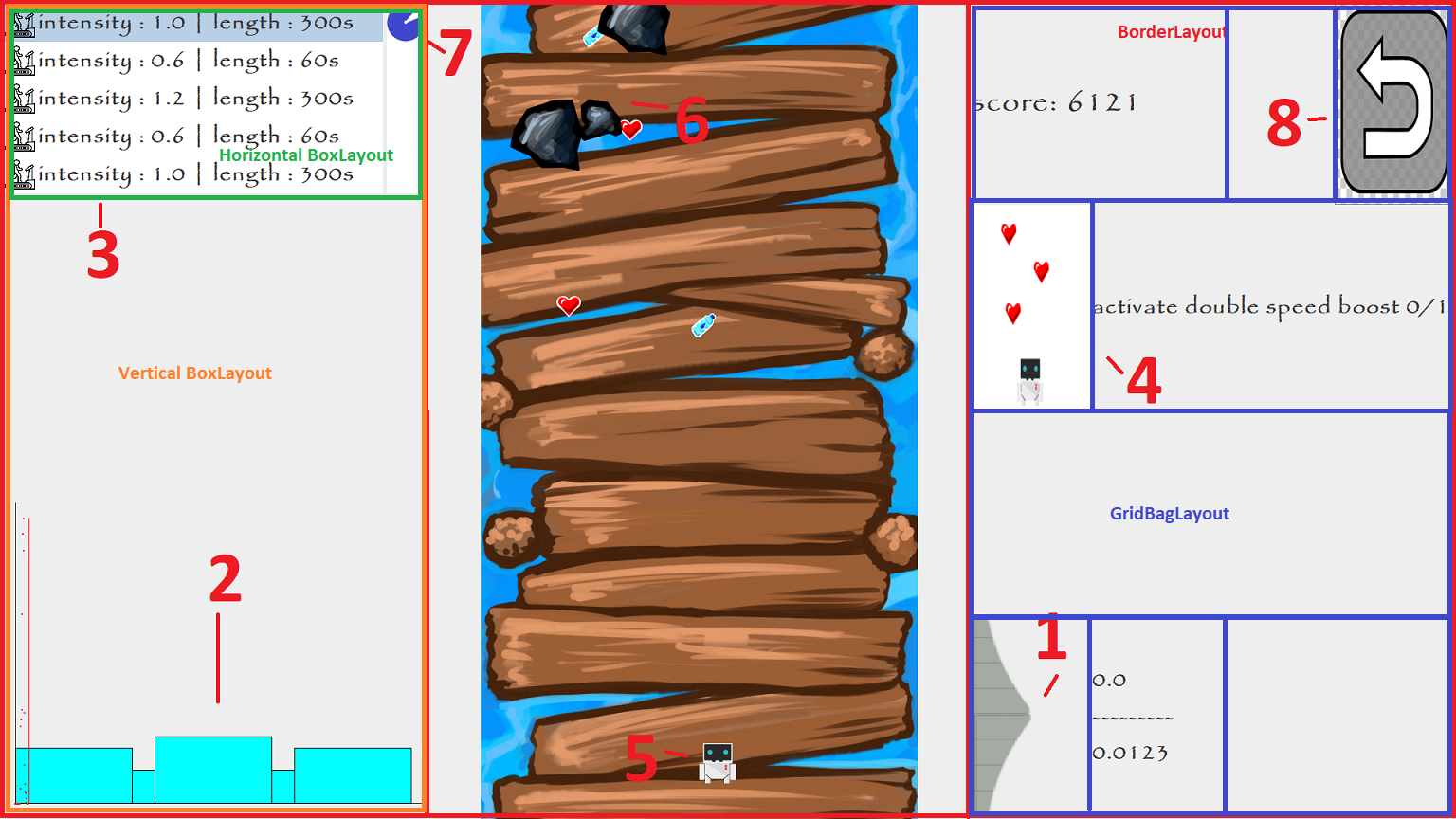
Le centre est implémenté avec la classe GamePanel. C’est un simple JPanel dont ont a redéfinit la méthode paint pour y dessiner l’état du jeu.

La partie gauche se trouve dans la classe ListProgramPanel qui contient deux Boxlayout, un vertical et un horizontal. 3 et 7 sont dans le layout horizontal qu’ensuite on place dans le layout vertical en y ajoutant le graphique.

Finalement, la partie la plus complexe, celle de droite se trouve dans SliderPanel. Tous ces composant sont stocké dans un GridBagLayout qui est la grill la plus flexible que propose swing. Elle permet surtout à un composant en de chevaucher plusieurs cellules de la grill.

Voici ce que donne le panneau de jeu avec les layouts :

Figure 42 Menu de jeu avec layouts



### La fin de partie

Une fois la partie terminée ou quittée, une Popup venant de la classe EndPopUpDialogBox étandant de JWindow apparaît. Elle permet notamment d’admirer sa collection de médailles obtenus durant la partie. Puis de comparer son score aux autres et enfin d’enregistrer son score et son nom dans une liste.

Voici un visuel :

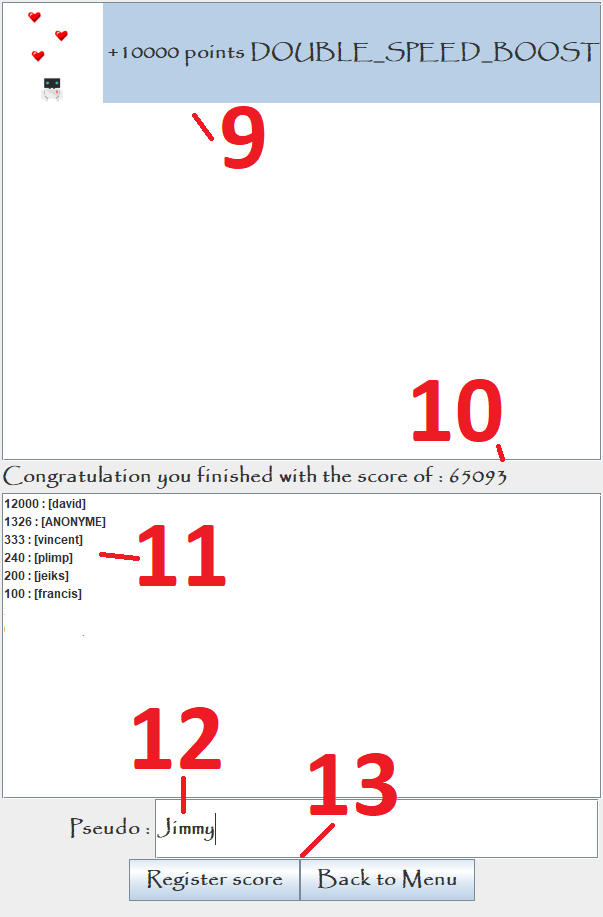


Figure popup de fin de partie

Une fois encore s’est une Boxlayout vertical qui contient le tout. Sauf pour les deux boutons qui sont aussi contenu dans une BoxLayout horizontal. Au niveau du comportement on notera qu’au moment d’enregistrer son nom plusieurs vérification sont faite. La première est que notre nom n’existe pas déjà dans la liste avec un meilleur score. La seconde est lors de l’écriture dans le champ texte, seul les caractères alphanumériques sont acceptés. Les deux boutons retournent au menu, mais seul « Register score » enregistre notre score dans le fichier ranking.json situé à la racine du projet.

## Effets Sonores

En plus, des effets visuels, un son à été mis en place pour signifier la collision. À Chaque collision un fichier son est lu pour faire comprendre à l’utilisateur que quelque chose vient d’arriver. De plus une musique tourne en boucle durant le programme sportif. Il faudrait évidement améliorer les effets sonores en ajoutant de nombreux son discret et en permettant à l’utilisateur de choisir sa musique plutôt que de le lui imposer. Enfin, un menu d’option permettant de couper le son est une priorité à avoir. Pour l’implémentation, la classe SoundPlayer permet de lire un fichier son une ou plusieurs fois, en boucle ou non, grâce à sa seule méthode playSound.

## Le pattern Observer

J’ai utilisé le pattern Observer dans plusieurs cas. Surtout pour permettre à l’interface graphique de se mettre à jour avec la modification du modèle. Mais aussi parfois pour remplacer ce qui aurait dû être une communication réseau (Par exemple l’accéléromètre qui envoie ces mesures à l’évaluateur). Je pense avoir cependant un peu trop utilisé ce pattern qui pourtant est connu pour casser l’encapsulation. J’ai néanmoins toujours implémenté le pattern via une classe interne qui limite les interactions possibles avec l’observable.

Les classe internes Observable existantes sont :

* PartObservable qui notifie les changements entre partie du programme sportif
* LengthObservable qui notifie à chaque instant du programme sportif
* ObservableEvaluation qui notifie lorsqu’une évaluation du résultat sportif est modifié
* EffortObservable qui notifie lorsque le détecteur change l’effort actuel du sportif
* EndObservable qui notifie lorsque le jeu est terminé
* ScoreObservable qui notifie un changement de score en cours de partie
* WinRuleObservable qui notifie lorsqu’un objectif est atteint
* RuleObservable qui notifie lorsque l’état d’un objectif est modifié

# 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).
* On peut définir son programme sportif avant de jouer
* On peut enregistrer son score final
* 6 objectifs de gamification sont fonctionnels et permettent de recevoir des médailles
* Un graphique temps réel résume la progression du sportif
* Un Menu d’aide permet à l’utilisateur de comprendre le but de l’application

# Bug connus

Ayant pour but d’obtenir un prototype fonctionnel afin d’avoir une idée des possibilités, je n’ai pas mis l’accent sur les tests. Cette liste de bug n’est donc absolument pas exhaustive et provient juste d’observations en cours d’implémentation.

## Recommencer des parties

Commencer une partie, puis revenir au menu, puis recommencer une partie peut perturber la Shimmer3. C’est un bug qui pour l’utilisation normal du programme ne devrait pas gêner puisqu’on ne fera qu’une partie par séance. Cependant, pour la journée porte ouverte c’est un souci majeur puisque les utilisateurs vont essayer et recommencer sans cesse.

Le souci vient probablement de ma lecture des derniers bytes lorsque je demande à la Shimmer3 de stopper l’enregistrement. Je n’aurais malheureusement pas le temps d’instiguer avant le rendu final.

## Dépassement du bouclier

Afin de pouvoir déplacer le personnage sur toute la longueur du mur, j’ai fixé sa position minimum et maximum au pixel prêt pour ne pas dépasser le background de jeu. Malheureusement, lorsqu’un bouclier est équipé. L’image du personnage s’agrandissant, si on déplace le personnage aux extrémité, l’image du bouclier dépasse et n’est jamais effacé en dehors du background. C’est un bug vraiment mineur et ne pose aucun problème. Il n’était donc pas dans ma liste de chose à faire pour l’instant.

## Baisse de fréquence trop forte

L’implémentation du détecteur de fréquence perds complètement la détection de dépassement de seuil lorsque l’amplitude de l’accélération baisse fortement. Le fais d’avoir ajouté des seuils avec prévisions et dynamique ne règle pas complètement le problème. Je ne vois pour l’instant pas de solution à ce problème. Heureusement sur le tapis de course, le changement de rythme est relativement faible. Ce bug n’est donc pas chose aisé à produire sur le tapis de course. La détection reprend malgré tout après un cycle complet du buffer, ce qui est néanmoins trop lent à mon avis.

# Problèmes rencontrés

## Valeurs erronées de la Shimmer3

Le plus gros souci a été les valeurs erronées venant de la Shimmer3 qui impactait mon calcul de fréquence. C’est le seul point ou j’ai dû demander assistance. Surtout qu’il était difficile de se voir avancer dans le projet sans un détecteur de mouvement fonctionnel. Il s’est avéré que c’était la méthode read de java qui, bien que bloquante, ne retournait pas forcément tous les bytes qu’on lui demandait.

## Calibration

Un problème majeur a été la calibration générale du projet, que ce soit pour calibrer les accéléromètres, pour calibrer la vitesse de défilement des objets (quelle vitesse est agréable pour l’œil ?). La calibration de divers seuils etc. Finalement, beaucoup de constantes s’entremêle et il est difficile de décider laquelle on devrait baisser ou monter. Je suis arrivé à un résultat, je pense qu’on peut toujours faire mieux. Cependant, je ne vois pas de moyen efficace d’automatiser le processus.

## La Wiimote obsolète

Le souci de compatibilité entre windows 10 et la Wiimote m’a fait perdre beaucoup de temps. Le Joy-con est une laternative tout à fait correcte. Je pense que les librairies vont probablement fleurir dans les années à venir.

## Conception seul

Ayant principalement fait des projets de groupes pendant l’HEIG et des laboratoires dont la conception était relativement intuitive ou spécifique à un problème donné, il est étrange de se voir confier un gros projet et de devoir choisir soit même toute la conception. C’était un exercice instructif, mais je ne le conseillerais pas en entreprise. Avoir des retours et des avis niveau conceptuel permet je trouve d’éviter de tomber dans des pièges de facilité ou des erreurs de conceptions.

# Amélioration possible

Bien que je sois content du résultat, le projet est loin d’être parfais. Voici quelques idées pour des améliorations futur.

## Mode réseau

Par soucis de temps, j’ai immédiatement éludé l’aspect réseau du projet. Il existe des tutoriels Unity expliquant comment faire un jeu multi-joueur en réseau. Si le projet est repris avec ce moteur il sera donc aisé d’implémenter cet aspect.

Dans le cas ou quelqu’un voudrait continuer d’écrire l’aspect réseau en modifiant mon code, il devra séparer les classes qui sont orientés serveur de celles orientés client. Certains patterns Observer sont des remplacements à la communication réseau qu’il y aurait dû avoir. Par exemple entre le détecteur qui transmettra l’effort depuis le client et l’évaluateur sportif qui reçoit ces données pour calculer une appréciation.

## Factorisation

L’UI ayant été faite rapidement sur les dernières semaines, il y aurait une possibilité de factorisation du code non négligeable. Notamment en utilisant une méthode utilitaire static pour le redimensionnement des images. En lisant le code vous pourrez d’ailleurs voir divers niveaux de factorisation qui montre mon évolution dans la compréhension de Swing avec le temps.

## Format d’image conservé

Le format des images n’est pas conservé lorsque je redimensionne. Ce qui donne un effet d’étirement moche. Par exemple pour le résumé des médailles obtenu en jeu ou dans le menu d’aide. Il serait donc intéressant d’avoir une méthode de redimensionnement des images avec conservation du format.

## Détecteur d’effort

La partie la plus dur du projet était d’implémenter un détecteur qui calcul un effort. Autant niveau conception du code qu’au niveau du choix des données utilisés. Je pense qu’un groupe de spécialiste dans la détection de mouvement serait nécessaire pour obtenir un résultat vraiment professionnel puisque c’est un domaine vaste et complexe.

## Base de données

Pour le moment la seule persistance est le résultat des scores des joueurs. Il faudra donc a posteriori rajouter une persistance via une base de données. Surtout si l’entreprise qui nous emploi souhaite toujours quadriller les mégalopoles de leurs concepts.

## Login

Il faudrait ajouter un menu de login au lancement de l’application ou un moyen de se connecter via un badge NFC par exemple afin d’identifier l’utilisateur. Ainsi, il serait possible d’offrir des services plus personnalisés, comme des programmes sportifs favoris ou un aspect communautaire qui relierais entraîneur et sportifs. C’est aussi grâce à ce login qu’on pourra ajouter une couche de machine learning.

## Machine Learning

Peut-être mon plus grand regret concernant ce projet est de ne pas avoir pu commencer la partie machine learning qui est probablement la partie la plus innovante du projet. Le projet à néanmoins été conçu dans l’idée que le détecteur se basera sur les résultats du machine learning pour s’auto-calibrer. Pour toutes personnes souhaitant avancer cette partie du projet, sachez que mon idée de base était de faire varier le premier paramètre du constructeur de la classe EffortCalculator selon le résultat du machine learning. Ainsi la partie serait calibré en début de partie avec un détecteur se basant sur les résultats personnels de la personne loggué. Pour le moment la valeur est en dur dans la sous-classe IMUCycleEffortCalculator.

Dans l’idée je voyais deux types de machine learning à utiliser. Le premier servant à placer un nouvel utilisateur sur une échelle de difficulté. Ainsi via un formulaire, une petite fille de 10 ans ne lui sera pas demandé le même effort qu’un homme de 20 ans sportif. J’imaginais donc demander au minimum, le sexe l’âge et le niveau sportif.

Le deuxième type de machine learning aurait servi au fur et à mesure de l’utilisation du logiciel à adapter selon les résultats de l’utilisateur, la courbe de difficulté afin d’optimiser sa courbe de progression sportive.

## Homogénéiser le style de l’UI

Ayant demander des assets à Céline Paquier, certaines images sont belles et possède un style propre. Malheureusement, je ne pouvais pas trop lui en demander et ai donc dû piocher aussi dans des assets gratuites ce qui casse l’homogénéité de l’UI.

# Remerciements

Je tiens à remercier Mr. Perez-Uribe pour son soutien et avoir supporté la quantité astronomique de mail que je lui envoyais quotidiennement.

Je remercie aussi Mr. Satizabal pour avoir résolu le mystère des fausses données provenant de la Shimmer3.

Un merci tout spécial aussi à Céline Paquier pour avoir accepté de dessiner quelques images de mon projet. Notamment le fond du jeu, qui grâce à elle ne ressemble plus à « un cou de giraf ». Pour ceux que ça intéresse, vous pouvez retrouver son travail sur [www.lepatchi.com](http://www.lepatchi.com).

Merci à mes amis pour avoir supporté mes monologues de « je galère à faire mon tb » sur tout médias confondu (<3 Bader, Benjamin, Camilo, Nadir, Sydney, Miguel, Élodie, Sura), Ainsi que pour l’aide à la relecture.

Et enfin merci à ma Maman pour avoir cuisiner des plats qui variaient entre bon, redondant et bizarre.

# Conclusion

Je suis plutôt satisfait du résultat final, bien que de nombreuses améliorations son possible. Mon plus gros regret étant de ne pas avoir pu commencer la partie machine learning. J’ai aussi hâte de discuter avec l’expert pour connaître son avis et se qu’il aurait suggéré pour parfaire la détection de l’effort. De plus, j’ai le sentiment d’avoir touché à beaucoup de sujets sans avoir pu approfondir complètement chacun d’eux, c’est probablement normal puisque ce travail de bachelor était une phase de test pour le projet ambitieux de notre mandataire. S’il fallait aujourd’hui répondre s’il est possible de créer l’infrastructure complète visée par Mr Vincent, je dirais qu’il faudrait encore tester une version réseau de mon prototype avant de pouvoir donner un avis. La partie machine learning étant je pense faisable mais surtout, il existerait dans le pire des scénarios, des moyens de contourner le problème.

# Table des illustrations

[Figure 1 Kinect 4](#_Toc14908847)

[Figure 2 Azure Kinect 5](#_Toc14908848)

[Figure 3 fonctionnalité -> services Bluetooth -> applications 6](#_Toc14908849)

[Figure 4 IMU Shimmer3 6](#_Toc14908850)

[Figure 5 Wiimote 7](#_Toc14908851)

[Figure 6 Joy-con 8](#_Toc14908852)

[Figure 7 Lava Run Game 13](#_Toc14908853)

[Figure 8 Jeux en compétitif 17](#_Toc14908854)

[Figure 9 architecture globale 18](#_Toc14908855)

[Figure 10 Kinomap UI 19](#_Toc14908856)

[Figure 11 Run social UI en course 20](#_Toc14908857)

[Figure 12 Exercube 20](#_Toc14908858)

[Figure 13 Zwift 21](#_Toc14908859)

[Figure 14 Bob personnages jouable dans Arcade Fitness 22](#_Toc14908860)

[Figure 15 détection fausse d'une collision avec une méthode naïve 27](#_Toc14908861)

[Figure 16 aire d'erreur, méthode naïve contre axes alignés 27](#_Toc14908862)

[Figure 17 hiérarchie de collision 28](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908863)

[Figure 18 resolveCollide méthode 29](#_Toc14908864)

[Figure 19 Barre d'effort, en vert l'arrière-plan se déplace à vitesse maximum. En rouge l'arrière-plan ne se déplace pas. 31](#_Toc14908865)

[Figure 20 Vue de la hiérarchie de classe pour la connexion Bluetooth avec la Shimmer3 33](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908866)

[Figure 21 Résultat fictif de l'implémentation naïve si le mouvement était parfait 33](#_Toc14908867)

[Figure 22 résultat fictif de fast Fourier transformation. Chaque pic serait une fréquence plus ou moins présente dans nos données d’accélération. 34](#_Toc14908868)

[Figure 23 Résultat fictif de la méthode par seuil. Ici en x = 1 on compterai un cycle. 34](#_Toc14908869)

[Figure 24 accélérations sur tapis de course et choix du seuil 35](#_Toc14908870)

[Figure 25 comptage des demi-cycle avec deux seuils 36](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908871)

[Figure 26 Fausse donnée augmentant drastiquement le maximum 36](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908872)

[Figure 27 graphique complet avec seuil dynamique, maximum médian et prévision du cycle suivant 37](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908873)

[Figure 28 détection de la fréquence actuellement encore instable 38](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908874)

[Figure 29 frequence fluide 39](#_Toc14908875)

[Figure 30 fonction de mapping complexe 40](#_Toc14908876)

[Figure 31 hiérarchie des vitesses de course 42](#_Toc14908877)

[Figure 32 UML des programmes sportifs 44](#_Toc14908878)

[Figure 33 UML Gamification 46](#_Toc14908879)

[Figure 34 uml évaluation sportive 48](#_Toc14908880)

[Figure 35 graphique du ralentissement de l’aérodynamisme 49](#_Toc14908881)

[Figure 36 menu de départ 50](#_Toc14908882)

[Figure 37 menu d'aide 51](#_Toc14908883)

[Figure 38 menu d'aide, point de vue avec layouts 51](#_Toc14908884)

[Figure 39 menu de création de programme sportif 52](#_Toc14908885)

[Figure 40 menu de création de programme sportif avec layouts 53](#_Toc14908886)

[Figure 41 menu de chargement d'une partie 54](#_Toc14908887)

[Figure 42 Menu de jeu avec layouts 55](file:///D:\Semestre6\TB\Gamification\Rapport.docx#_Toc14908888)

[Figure 43 popup de fin de partie 56](#_Toc14908889)

# 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>

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