



**Rapport de projet tuteuré (2016/2017)**

« Créer un jeu alliant hardware et software afin d’apprendre la programmation Arduino de manière ludique »

**Tutrice : Amélie Cordier**

Adrien Techer

Valentin Guevara

Jérémy Duval

Fanny Velien

# Table des matières

[Table des matières 1](#_Toc477388066)

[1 Introduction 3](#_Toc477388067)

[1.1 Contexte & Tuteur 3](#_Toc477388068)

[1.2 Présentation du projet 4](#_Toc477388069)

[1.3 Organisation 4](#_Toc477388070)

[1.3.1 GitHub 4](#_Toc477388071)

[1.3.2 Responsabilités des membres 4](#_Toc477388072)

[1.4 Droits 5](#_Toc477388073)

[1.4.1 Contenus multimédias et code non créés 5](#_Toc477388074)

[1.4.2 Licence 5](#_Toc477388075)

[1.5 Préconception 5](#_Toc477388076)

[2 Technologies utilisées 5](#_Toc477388077)

[2.1 La librairie LibGDX 5](#_Toc477388078)

[2.2 La librairie RXTX 6](#_Toc477388079)

[3 Mécaniques du jeu 6](#_Toc477388080)

[3.1 Organisation du code et logique 6](#_Toc477388081)

[3.1.1 Design Pattern 7](#_Toc477388082)

[3.1.2 Logique du flux de données 8](#_Toc477388083)

[3.2 Menus et changement d’écran 8](#_Toc477388084)

[3.3 Le monde virtuel 9](#_Toc477388085)

[3.3.1 Physique 9](#_Toc477388086)

[3.3.2 Génération du monde 9](#_Toc477388087)

[3.3.3 Gestion des contacts 10](#_Toc477388088)

[3.3.4 Les entités vivantes 11](#_Toc477388089)

[4 Arduino 12](#_Toc477388090)

[4.1.1 Pourquoi Arduino ? 12](#_Toc477388091)

[4.1.2 Relation Software-Hardware 12](#_Toc477388092)

[4.1.3 Partie programmation 13](#_Toc477388093)

[4.1.4 Grove 15](#_Toc477388094)

[4.1.5 Amélioration : La création d’un parseur 16](#_Toc477388095)

[5 Système d’apprentissage 17](#_Toc477388096)

[5.1.1 Cours 17](#_Toc477388097)

[5.1.2 Défis 17](#_Toc477388098)

[6 Contenu Multimédia 18](#_Toc477388099)

[6.1 Gestion dans le Jeu 18](#_Toc477388100)

[7 Graphismes 18](#_Toc477388101)

[7.1.1 Généralités 18](#_Toc477388102)

[7.1.2 Choix des polices 21](#_Toc477388103)

[7.1.3 Outils 21](#_Toc477388104)

[7.2 Musiques & Sons 22](#_Toc477388105)

[7.2.1 Musiques 22](#_Toc477388106)

[7.2.2 SFX 23](#_Toc477388107)

[8 Les obstacles rencontrés 24](#_Toc477388108)

[8.1 Organisation 24](#_Toc477388109)

[8.2 Intégration Continu 24](#_Toc477388110)

[8.3 Débogage 25](#_Toc477388111)

[9 Futur Développement 25](#_Toc477388112)

[9.1 Interactions avec Arduino & Défis 25](#_Toc477388113)

[9.2 Objectifs de Jeu 27](#_Toc477388114)

[9.3 Univers 27](#_Toc477388115)

[10 Bilans Personnels 27](#_Toc477388116)

[10.1 Jérémy 27](#_Toc477388117)

[10.2 Valentin 28](#_Toc477388118)

[10.3 Fanny 29](#_Toc477388119)

[10.4 Adrien 30](#_Toc477388120)

[11 Conclusion 30](#_Toc477388121)

[11.1 Bilan Général 30](#_Toc477388122)

[11.2 Remerciements 30](#_Toc477388123)

[12 Annexes 30](#_Toc477388124)

[12.1 Diagrammes UML 30](#_Toc477388125)

# Introduction

## Contexte & Tuteur

Malgré une image parfois controversée, le jeu vidéo est un divertissement extrêmement populaire. Réunissant de très nombreux aspects techniques et artistiques, sa richesse lui a permis de s’approprier peu à peu une nouvelle place : celui de support d’apprentissage. Les jeux vidéo ayant cette vocation ont été popularisés sous l’appellation de Serious Game (ou « Jeu sérieux » en français).

Novatrice, la transmission de connaissances via un Serious Game permet d’allier l’amusement et le plaisir du jeu à l’acquisition de connaissances nouvelles. Malgré le fait que le support ait encore des choses à prouver (car il est transgressif), nous avons choisi de réaliser un tel projet afin de s’inscrire dans une dynamique en faveur du Serious Game et de sa marginalisation. Puisque nous sommes tous trois étudiants en Informatique ainsi qu’amateurs de jeux vidéo et de créations originales, il s’agissait pour nous d’un challenge à la convergence de nos intérêts personnels.

Nous souhaitions avant tout développer l’intérêt pour les nouvelles technologies de l’informatique et de l’électronique. Ayant déjà expérimenté la programmation Arduino au lycée et connaissant les possibilités offertes par le microcontrôleur, nous l’avons logiquement choisi.

L’idée était de ne pas nécessairement viser les amateurs de jeux vidéo mais également les néophytes curieux de programmation et de réussir à créer l’interaction. La prétention du projet n’est pas d’offrir une formation complète mais une initiation avancée d’une forme nouvelle.

Notre tutrice est Amélie Cordier. Lorsque nous avons défini les grandes lignes du projet, nous nous sommes naturellement renseignés auprès d’elle avant de lui soumettre le concept. C’est une professeure qui nous avait déjà témoigné un intérêt fort pour la robotique et Arduino et nous souhaitions fortement pouvoir travailler avec elle.

## Présentation du projet

Le projet consiste à développer un jeu vidéo en 2D afin d’enseigner les bases de la programmation Arduino. Le jeu est découpé en différents niveaux qui mettrait le joueur face à des défis de programmation qu’il faut obligatoirement résoudre pour débloquer de nouvelles fonctionnalités et compléter la partie.

Nous avons imaginé un concept de level design où seule l’interaction entre hardware et software permettrait au joueur d’avancer, la carte Arduino étant en permanence connectée. Cela consiste à intégrer peu à peu de nouveaux composants (comme un potentiomètre, des LEDs…) et à transférer des données virtuelles dans le monde physique, par exemple en modélisant la barre de vie grâçe à un bargraphe. L’intégration de modules complémentaires passe par la lecture d’un petit cours, leurs déclarations, et l’écriture de quelques lignes afin de le rendre fonctionnel dans le jeu. L’idée est de laisser peu à peu la main à l’utilisateur dans l’écriture de programmes de plus en plus complexes.

## Organisation

### GitHub

Figure -Logo Github

Dans le but d’optimiser nos phases de développement, nous avons mis en place un CVS[[1]](#footnote-1). Il nous a notamment permis de travailler chacun de notre côté et de pouvoir mettre en commun les parties de code au moment où celles-ci ont été nécessaires.

<https://github.com/Jeremy-Duval/Genuini_s_Journey_Ruins_Of_Ivrea>

### Responsabilités des membres

* Adrien : Mécanique du jeu
* Jérémy : Cours et défis
* Fanny : Graphismes
* Valentin : Arduino

## Droits

### Contenus multimédias et code non créés

Initialement, nous considérions que la réalisation des sprites permettrait un design plus proche de l’idée que nous avions du jeu.

Après quelques recherches en ligne, nous avons découvert un pack contenant la totalité des éléments graphiques nécessaires à la création de l’univers Genuini. De plus, il s’agit de ressources libres de droit (appartenant au domaine public).

<http://opengameart.org/content/platformer-art-complete-pack-often-updated>

### Licence

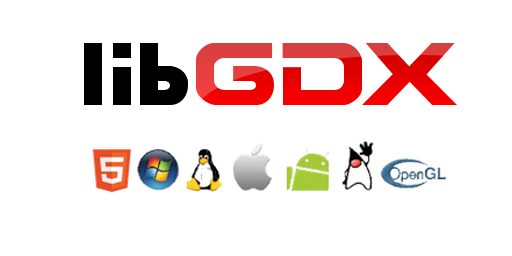
Genuini’s Journey : Ruins of Ivrea est un logiciel libre sous licence CeCILL-C.

Le dépôt contenant l’intégralité des codes sources mais également les différents médias est déclaré sous licence par l’intermédiaire d’un fichier se situant à sa racine (annexe – Licence).

## Préconception

# Librairies

## La librairie LibGDX

[](https://libgdx.badlogicgames.com/)

LibGDX est une bibliothèque libre de droit et multiplateformes basée sur OpenGL et utilisable sous Java, Android et HTML5.

Nous l’avons choisie car elle est performante, en constante évolution et documentée. Effectivement, de nombreux témoignages d’utilisation de cette bibliothèque logicielle sur le Web nous ont encouragés à en apprendre un peu plus sur cette librairie.

De par sa réputation et sa documentation, nous n’avons pas regretté ce choix qui s’est avéré être un allié de taille dans la création des graphismes du jeu.

## La librairie RXTX

Afin de permettre un interfacage efficace entre notre carte Arduino et le programme JAVA, nous avions besoin d’une librairie capable d’intéragir directement avec les composants. A ce moment précis, après de nombreuses recherches sur Internet et notamment sur le site internet officiel d’Arduino, nous avons trouvé un ensemble de fonctions permettant de réaliser l’intégralité de ces actions.

Cette librairie se nomme « RXTXComm.jar » .

<http://playground.arduino.cc/Interfacing/Java>

Elle nous permet premièrement d’interroger les ports séries(cf…) à travers des méthodes déjà écrites mais également d’envoyer des informations ou en recevoir plus facilement.

L’utilisation de ces librairies s’est avérée essentiel pour nous permettre un développement optimisé et plus rapide.

Un des nombreux avantages de cette bibliothèque logicielle était son universalité, elle est compatible avec tous systèmes d’exploitation et est capable de trouver les ports de communication de tous les SE. De plus, de par sa célébrité, bon nombre d’exemples existent en ligne et nous ont permis d’être plus efficace dans notre phase de développement.

# Mécaniques du jeu

## Organisation du code et logique

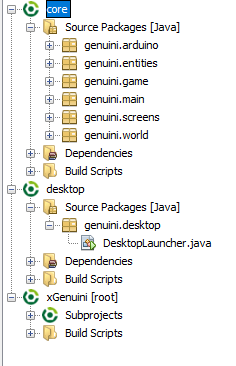
Naïvement d’aucuns pourraient penser que le code d’un jeu vidéo s’articule de la même manière qu’un programme informatique classique. Or, la création d’un jeu vidéo dans son intégralité requiert l’intégration de composants multimédias, la génération d’un monde virtuel avec en son sein des créatures et décors interactifs, des menus, la gestion d’interactions utilisateurs etc…

L’organisation du code joue donc un rôle primordial dans la capacité des membres de l’équipe à construire le jeu efficacement et sans être tourmenté par les éventuelles erreur de régression (cf. 5.3 ).

Nous avions conscience de l’enjeu de la clarté de code, d’autant que nous allions être amenés à réutiliser le code des autres. Nonobstant, nous nous sommes vite aperçu que nous n’avions aucune idée de comment organiser le code pour notre jeu. Nous l’avons compris qu’après avoir recréé la base (classes essentielles qui permettent chacune des mécaniques du jeu) trois fois puis avoir factoriser, épurer et simplifier le code à chaque étape de création.

La description suivante fait état de notre compréhension sur cette articulation issue des spécificités de LibGDX, de Java et de nos connaissances.

### Design Pattern

Un projet LibGDX avec Gradle sépare de manière inhérente le code lié au jeu, du code lié au lancement de celui-ci. Le lancement peut ainsi s’adapté aux mobiles et aux différents systèmes opérationnels sur lequel il est lancé.

Dans notre cas, le lancement se fait donc sous Windows à partir de la classe DesktopLauncher.

Cette dernière lance la classe principale du jeu, qui initialise le contenu (cf. 4.1) et le premier écran (i.e. menu principal).

Les noms de packages résume assez bien la séparation logique, comme nous allons le voir ci-dessous avec la traduction de la JavaDoc.

**Package Arduino**

Ce package regroupe toutes les classes faisant le lien avec Arduino.

**Package Entities**

Ce package contient tout le code associé aux éléments du monde qui agissent sur celui-ci ou sur le joueur.

**Package Game**

Ce package contient tous les gestionnaires pour charger le contenu, modifier les écrans et créer les caméras, les skins (textures appliquées aux boutons), le texte, etc….

**Package Main**

Ce package contient uniquement la classe principale du jeu, qui s’occupe de créer l’écran de démarrage, de charger le contenu (en appelant le gestionnaire de contenu) et de mettre à jour l’écran de jeu en cours de rendu.

**Package Screens**

Ce package contient tous les écrans utilisés lors du jeu comme le menu principal, l’écran de chargement, l’écran de jeu etc…

### Logique du flux de données

A chaque fois qu’un écran est chargé, les boutons, les écouteurs (détecteurs d’entrée utilisateurs), le texte, la musique sont créés.

Le jeu est rafraichi grâce à la méthode *render()* native de LibGDX. Cette méthode est appelée par la classe principale (*MainGame)* 60 fois par seconde. Elle dessine toutes les images/textures etc… et appelle la méthode *update()* de chaque élément ayant besoin d’être mise à jour.

Cette dernière fait passer le monde virtuel (cf. 3.3) au stade suivant calculant ainsi toutes les valeurs des objets subissant la physique du jeu. C’est également en son sein, que le traitement d’éventuelles actions venant de l’utilisateur sont traitées.

**Sauvegarde des données**

Les données relatives au jeu comme la position, la vie du joueur, son état d’avancement sont stockées dans un fichier XML. Le gestionnaire de préférences permet d’accéder et de modifier ces données.

## Menus et changement d’écran

Tous les écrans sont des extensions de notre classe abstraite d’écran qui elle-même est une implémentation de l’interface *Screen* de LibGDX. C’est aussi une extension de la classe *Stage* qui gère le placement des boutons et du texte fixe ainsi que leurs actions en fonction de l’entrée utilisateur. C’est donc au travers de la scène (*Stage)* que nous avons effectué la création et le placement des boutons des menus.

Le changement d’écran est natif de libGDX nous avons juste eu à implémenter la suppression de l’ancien écran. De plus nous avons voulu créer un « Splash-screen » ou écran de chargement. Pour cela nous avons créé un processus en arrière-plan (thread) chargeant l’écran de jeu alors que l’écran de chargement boucle sur une animation.

## Le monde virtuel

### Physique

LibGDX intègre le moteur physique Box2D qui permet de simuler une physique réaliste au sein d’un monde 2D.

Le monde virtuel où évolue le joueur est composé de « corps » qui sont des objets géométriques localisés ayant de multiples propriétés. Leur forme et leur densité permet au moteur de calculer leur masse et donc la manière dont ils se meuvent au sein du monde vis-à-vis de la valeur du champ gravitationnel choisi. Les déplacements des corps se font au moyen d’impulsion linéaire au niveau des centres de masses des objets.

Ils possèdent également un coefficient de friction, une résistance à l’air qui permet d’avoir un rendu plus réaliste et des attributs permettant de les identifier et de décrire leur solidité, relative aux autres corps.

Ces propriétés sont fixées lors de la génération du monde.

### Génération du monde

Le monde est généré à partir d’une carte de tuiles (« tiled map » en Anglais). Cette carte qui correspond à un « niveau » du jeu, a été dessinée en posant différentes tuiles (chaque tuile étant une image) et « objets de carte ». Ces objets sont en fait des indications pour le gestionnaire du monde comme nous le verrons par la suite.

La génération du monde est gérée exclusivement par la class *WorldManager*. Cette séparation permet de limiter la taille de la classe *GameScreen* pour qu’elle ne serve dans la mesure du possible que de nœud central aux composants du monde.

Le gestionnaire commence son activité par charger la carte que lui fournit l’écran de jeu au travers du gestionnaire de préférences. Après l’avoir chargé, il récupère chaque couche (« layer » en Anglais) puis créer les corps correspondants. Cette étape est détaillée ci-dessous.

#### Terrain

Le terrain du monde est composé exclusivement de tuiles. Un corps est généré à partir de la position et de la forme de la tuile.

On a donc une génération de la forme : *Tuile 🡪 Corps* , le corps se superpose dons au dessin de la tuile.

#### Objets

Les couches objets permettent une génération de la forme *Objet de carte 🡪 Entité (Corps + Sprite).*

N.B : Un Sprite est une image en deux dimensions qui peut être déplacée par rapport au fond de l'écran.

La méthode créant chacune des types d’objets parcours les objets de la carte de la couche correspondante.

**Décors interactifs**

En l’état, nous avons implémenté trois types de décors interactifs :

* Tourelles pouvant envoyer des boules de feu à une vitesse et dans une direction variable.
* Ressort faisant rebondir le joueur
* Bouton : lié à un objet, permet de l’activer/désactiver (ex : désactivation d’une tourelle)

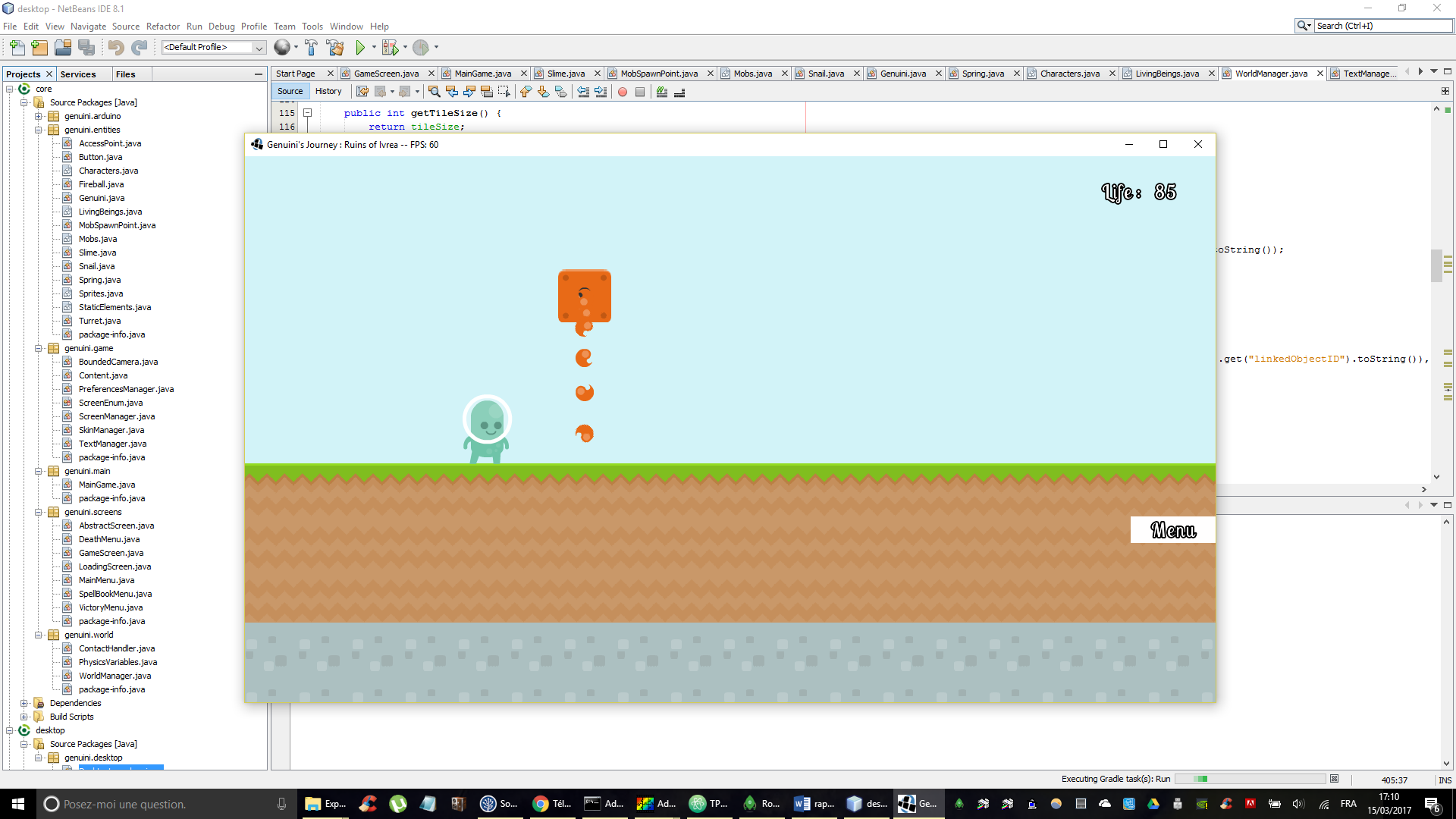
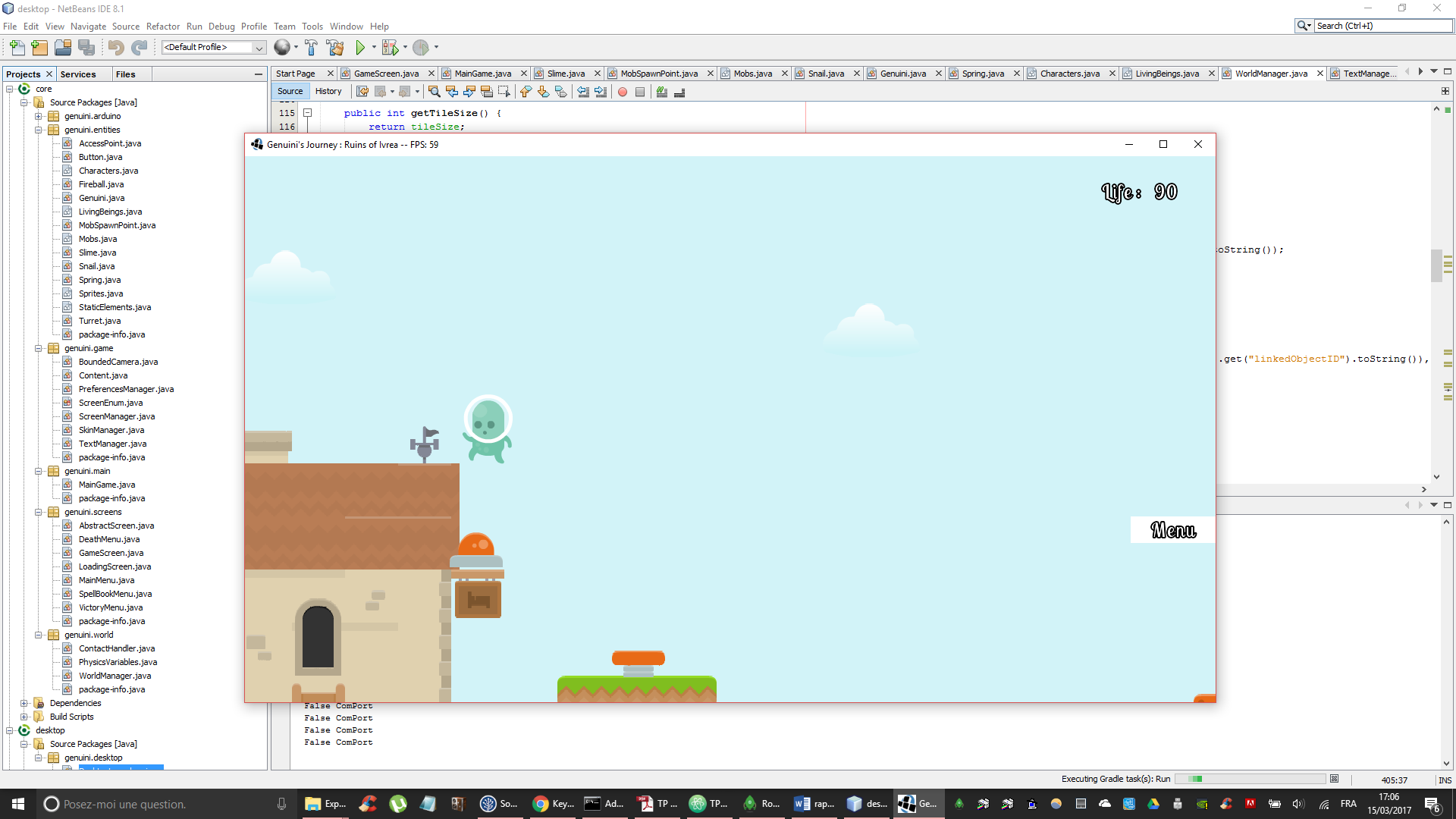
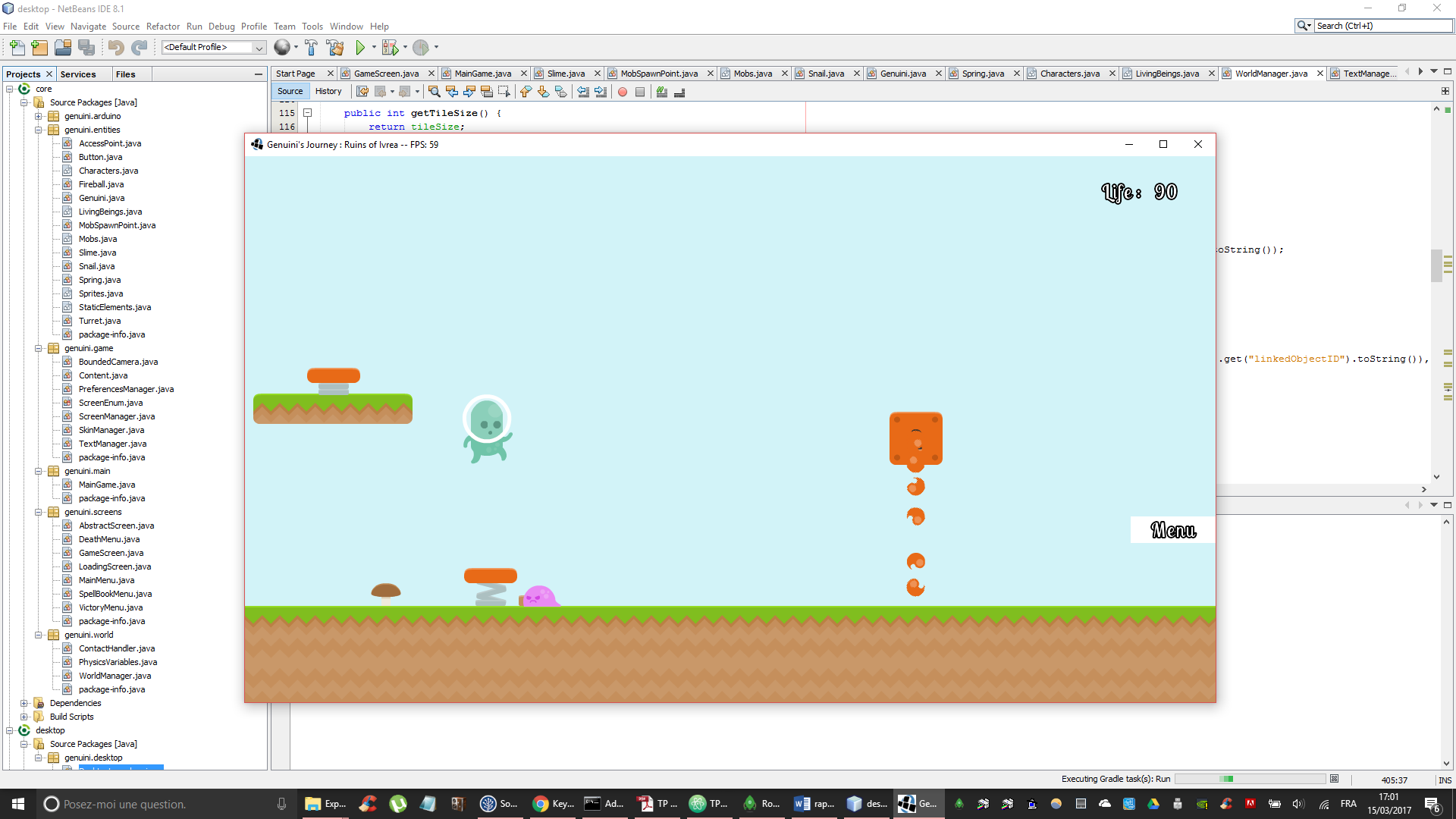


Figure - Bouton

Figure - Tourelle

Figure – Ressort

**Points de générations de créatures**

Ce sont des lieux non-dessinés qui servent à placer une zone d’apparition de créatures.

**Points d’accès**

Ce sont des lieux non-dessinés qui servent à accéder à d’autres niveaux/cartes ou qui fixe le lieu d’apparition du joueur.

### Gestion des contacts

Box 2D permet l’implémentation d’une interface gérant les contacts. Lorsque deux corps entrent en collision le gestionnaire de contact effectue un traitement en deux temps : au début de la collision et lorsque les deux objets se séparent. En fonction des données dites « utilisateurs » des corps, une procédure est appliquée à celui-ci comme par exemple une impulsion linéaire ou la modification des dîtes données « utilisateurs ». Un exemple de cette modification est le marquage du corps afin de signifier au gestionnaire de monde de supprimer détruire ceux-ci.

### Les entités vivantes

Les entités vivantes sont animées.

En fait, les images associées aux différentes créatures personnages sont indexées et compressées dans un fichier (atlas de textures).

La manipulation de ce dernier permet d'accéder aux images de manière chronologique. Ainsi une procédure utilisant un compteur de temps, permet de récupérer l'image correspondant à l'état dans lequel se trouve l'entité. Les états comme « marche », nécessitant une animation renvoi l'image correspondant au temps écoulé.

#### Créatures mobiles

Les créatures sont générées par l’objet correspondant à leur zone d’apparition. Elles ont un comportement interactif avec le joueur en fonction de l’état dans lequel elles se trouvent.

slimeWalk_1Nous en avons pour le moment implémentées deux :

Les slimes (littéralement substance visqueuse) glissent sur le terrain et attaquent le joueur à partir d’une certaine distance.

snailWalk_1snailShellLes escargots sont des créatures endormies s’ils sont trop loin du joueur et passes à l’attaque lorsque le joueur est près.

#### Genuini

Le joueur possède des caractéristiques qui sont sauvegardées : sa position et sa vie. Ses mouvements sont déterminés par l’entrée de l’utilisateur (touches ZQSD). Il a également des capacités spécifiques. Nous n’avons pour le moment qu’implémenter la capacité de lancer des boules de feu.

# Arduino

Figure -Logo Arduino

### Pourquoi Arduino ?

L’utilisation de la technologie Arduino était une nécessité. Afin de familiariser l’utilisateur avec les concepts initiaux de la programmation de composants électroniques, nous étions à la recherche d’un système adaptable et ludique. En tenant compte de ces attentes, nous avons entamé une recherche de périphériques facilement programmables et compatibles avec la carte Genuino 101 que nous allions utiliser pour le projet.

Ce choix était, avant tout, une décision commune. Durant notre parcours pré-universitaire, nous avons tous côtoyé à un moment ou un autre le monde merveilleux d’Arduino. Le projet tuteuré du semestre 1 était pour la plupart des membres du groupe une première approche au sein de l’IUT à cet environnement électronique que nous voulions absolument découvrir en profondeur.

Jérémy et Valentin avaient déjà travaillé avec cette technologie sur des projets pré-baccalauréat et avaient par conséquent une première expérience et certains réflexes de programmation qui se sont avérés forts utiles lors de la phase de développement de la partie électronique. Pour toutes ces raisons, Arduino nous semblait un choix judicieux et plus particulièrement une carte : Genuino 101. Elle a pour avantage d’intégrer nativement un gyroscope mais aussi une possibilité de connexion Bluetooth. Pour son côté pratique et son aspect « plug-and-play », ce microcontroleur s’est imposé comme une évidence.

### Relation Software-Hardware

Nous étions guidés par cet objectif de transmettre la connaissance nécessaire au joueur afin de s’amuser comme nous l’avons fait avec ces différents outils Arduino. Nous n’avions pas pour ambition de le porter à un haut niveau mais de lui permettre d’acquérir un niveau débutant suffisant pour commencer à tester les différents composants.

Pour parvenir à cet objectif-là, notre plan d’action était le suivant :

* Donner la possibilité de lire des cours
* Appliquer l’apprentissage à travers des questions en jeu
* Les bonnes réponses permettront de débloquer des parties du jeu

La finalité étant de pouvoir demander à l’utilisateur de saisir un petit programme qui permettrait par exemple d’allumer et éteindre une LED.

A travers la démonstration, nous avons permis un premier contact entre ces deux univers. Effectivement, le joueur doit pouvoir répondre à une question portant sur un programme Arduino afin de débloquer l’inversion de gravité et lui permettre de terminer le premier niveau.

Notre second objectif était de développer l’expérience en jeu pour surprendre l’utilisateur et lui montrer que les possibilités apportées par ce microcontroleur sont infinies. Nous avons alors mis en place une liaison en continue avec les périphériques Arduino qui permettent de :

* Tenir informé de la vie de Genuini[[2]](#footnote-2)
* Afficher sur l’écran LED l’écran actif
* Permettre une interaction immédiate entre le monde physique et logiciel

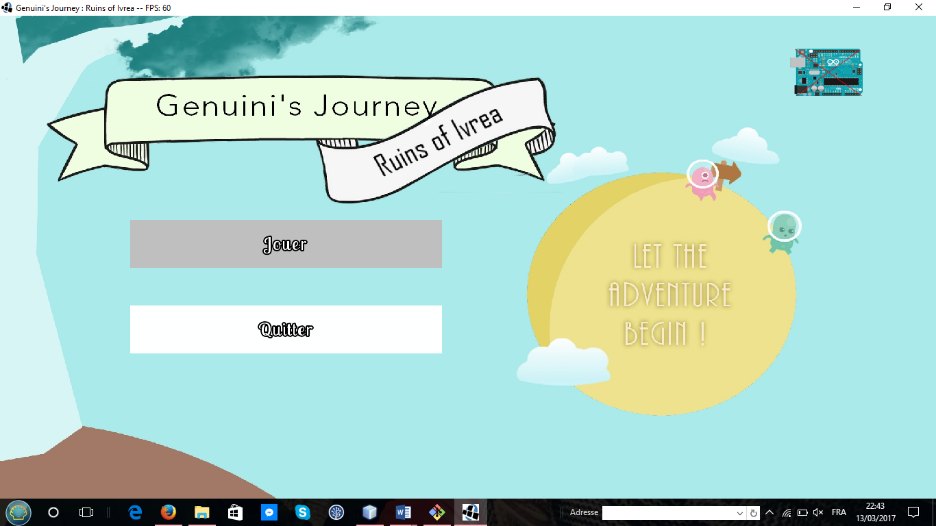
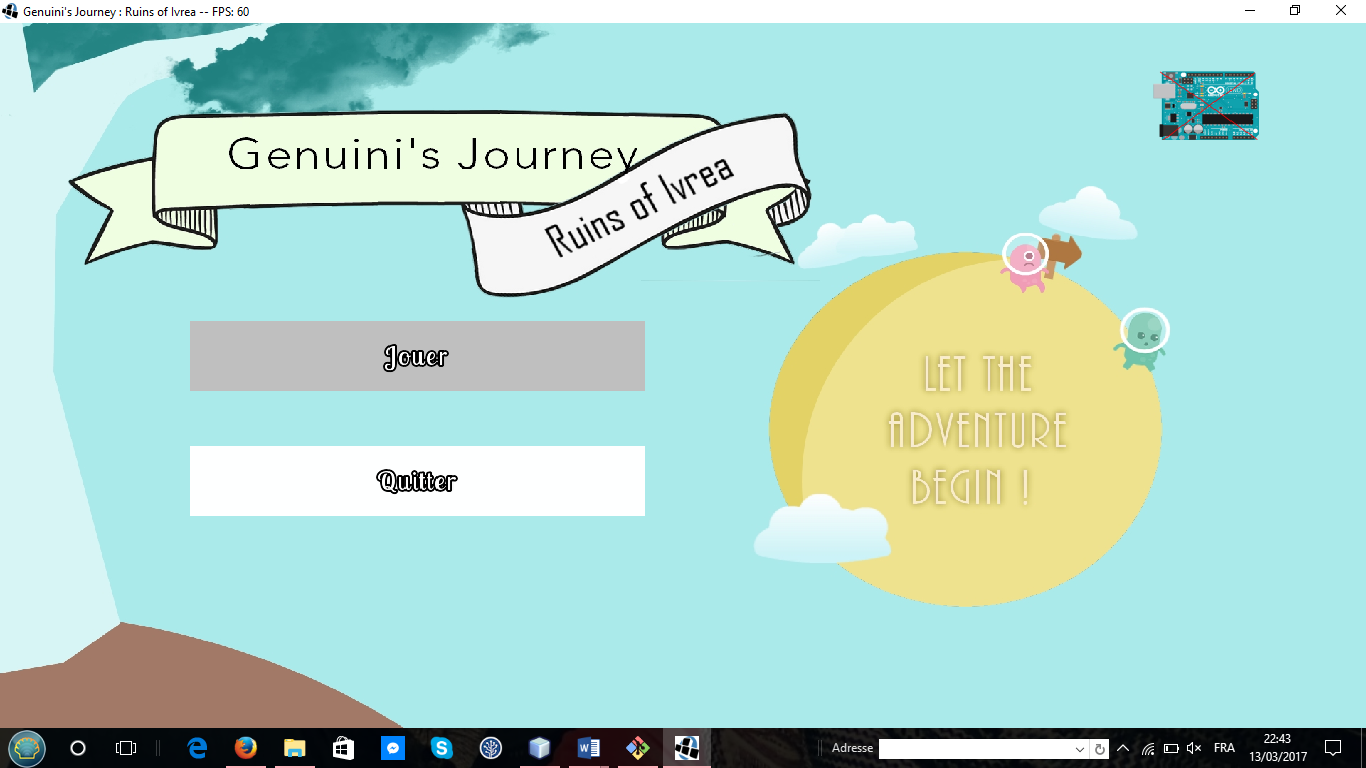
Le jeu devait pouvoir être également utilisable sans connexion entre les composants et l’ordinateur, nous avons alors implémenté la détection des périphériques et les actions qui seront effectuées en conséquence.

Figure -Liaison Arduino-Jeu non détectée

### Partie programmation

Comme expliqué précédemment, l’environnement Arduino était un univers à approfondir pour certains et à découvrir pour d’autres. L’objectif premier était une collaboration sur toutes les parties du projet par tous les membres du groupe. Le développement Hardware n’a donc pas dérogé à la règle et a interessé de par son aspect ludique toutes les parties.

Le langage de programmation utilisé dans l’univers Arduino est le C/C++. Ayant déjà suivi des cours sur ce langage, la manière de développer n’a donc pas été une surprise pour nous.

Pour une réalisation optimale, nous avons travaillé sur l’IDE du constructeur nommé Arduino IDE. Son apparence ne diffère guère de nombreux autres environnements de développement, il a pour avantage de mettre en avant une interface claire et épuré permettant une compréhension rapide des différentes fonctionnalités du logiciel.

Son principal inconvénient reste la non présence de coloration syntaxique. A ce problème, une solution existe. Il est possible de coder sur un environnement différent appelé SublimeText qui lui prend en compte de nombreuses aides aux développeurs. Un plugin présent dans sa librairie de contenu additionnel permet à l’utilisateur de compiler et téléverser son programme par l’intermédiaire de ce logiciel à son microcontroleur.

<http://eskimon.fr/2224-arduino-mini-tuto-utiliser-sublime-text-ide>

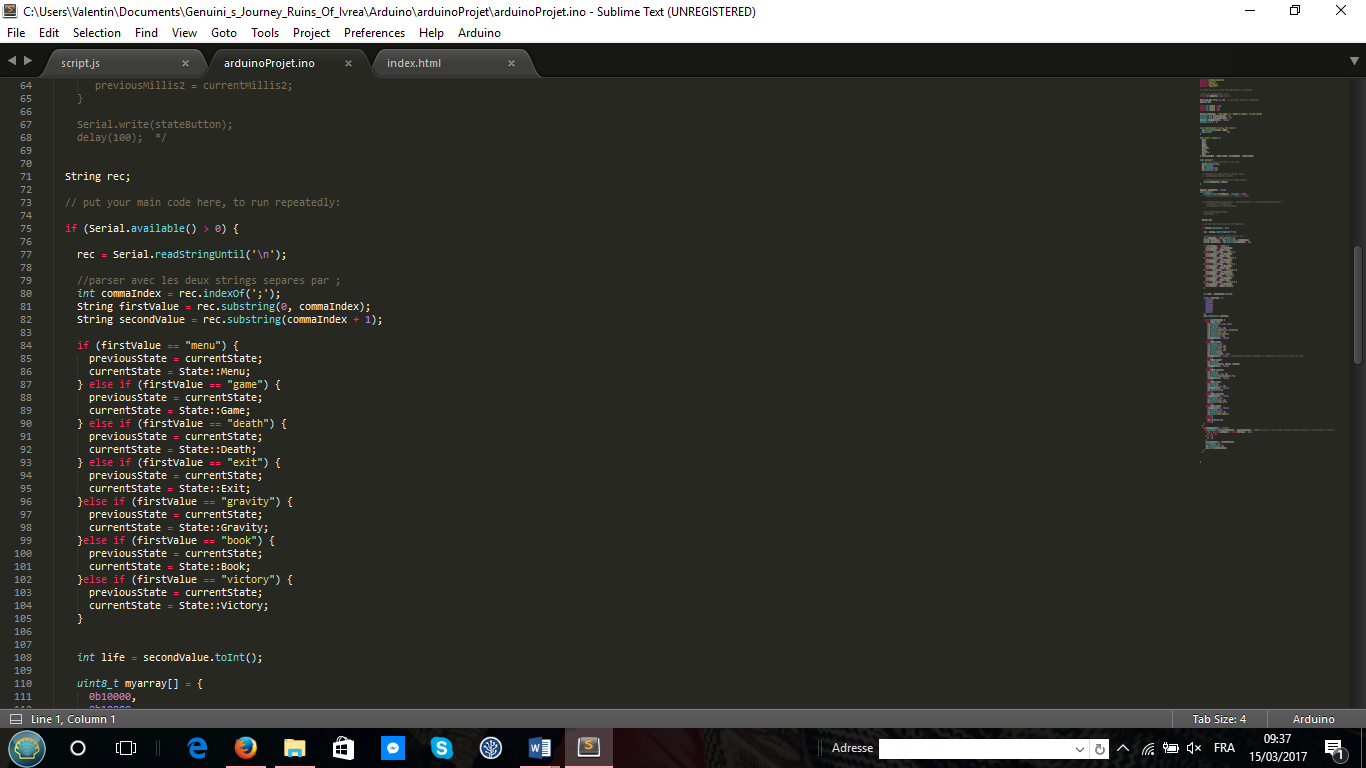
Le point essentiel du développement était la réussite de l’interfacage entre notre code Arduino et le jeu exécuté sur un ordinateur. Il s’est alors posé la question de la manière d’obtenir un résultat satisfaisant entre ces deux appareils.

Un port permettant la liaison entre le microcontroleur et le PC existe, il se nomme « Port COM ». La connexion série permet une relation d’entrée-sortie entre les deux appareils aux extrémités de la liaison. C’est une norme disponible sur tous les périphériques actuels et est utilisable via les connectiques USB. Une machine comporte plusieurs ports COM mais en attribue un seul pour le périphérique désiré.

*« Notre programme devait pouvoir faire face à tous types de situations. »*

Une de ces difficultés était la détection de la connexion entre le module Arduino et l’ordinateur. Pour parer ce problème, nous avons donc parcouru un tableau contenant tous les ports COM de la machine et avons vérifié si le microcontroleur était présent. La suite allait de soi, si le module est connecté nous le préparons pour recevoir et envoyer des données sinon le jeu informe l’utilisateur de la non détection du périphérique.

Par conséquent, afin de transmettre des informations de vie du personnage ou bien d’écran actif, nous utilisons une méthode : « InstanceDeLArduino.write(« infos ») » qui envoie sur le port série une séquence de bits correspondant à l’action que le microcontroleur va devoir effectuer. Nous devions rendre possible l’envoi de plusieurs informations en simultanée à l’Arduino. Malheureusement, la liaison série ne peut contenir qu’une seule suite de données, une chaîne de caractères. Il a donc fallu diviser notre chaîne à l’aide de point virgule afin que le programme côté microcontroleur puisse l’interpréter et effectuer les actions en conséquence.

Du côté Hardware, une énumération des écrans ainsi qu’une liste des différentes instructions permettent au code Arduino d’entretenir un contact réel avec le gameplay. Pour traiter les informations reçues, nous avons séparé la chaîne de caractère afin de pouvoir par exemple récupérer la valeur de la vie du personnage et ensuite l’écran actif du jeu.

Afin d’illustrer notre propos, soumettons un exemple concret qui est l’affichage de l’écran de mort. Une fois la vie valant 0 la partie logicielle s’occupe, de changer d’écran en indiquant au joueur la fin de la partie mais également, de lancer une musique particulière. A ce moment précis, un petit écran LED relié à la carte Arduino inscrit un message de mort et change de couleur pour passer en rouge afin d’augmenter le côté dramatique de la situation.

Finalement, de la même manière que la programmation JAVA, le développement Arduino a constitué une part importante dans le processus de réalisation du projet. Nous avons dû nous coordonner afin de parvenir à effectuer toutes les actions voulues.

Pour conclure, le projet à travers un univers Arduino s’est avéré être une excellente idée et ce microcontroleur a su convaincre tous les membres du groupe.

### Grove

Afin de rendre les branchements électroniques plus faciles pour l’utilisateur, nous cherchions des composants additionnels compatibles avec notre solution Arduino et obéissant à ce principe que nous avions fixé : « plug-and-play ». Après de multiples recherches, nous avons découvert l’univers Grove. Il s’agit de modules offrant la possibilité d’accroitre les capacités de la carte en y ajoutant pour l’exemple des capteurs ou bien des écrans. L’avantage de ces outils est la librairie prête à l’utilisation fournie par l’entreprise SeedStudio comportant toutes les fonctions nécessaires à l’exploitation complète de ces modules.

<http://wiki.seeed.cc/Grove_System/>

Afin de proposer au joueur une multitude d’interaction avec le milieu Hardware, nous avons opté pour un pack comprenant les capteurs que nous souhaitions puis nous avons fait l’acquisition également d’une bargraphe LED pour permettre un affichage sur une Interface Homme-Machine de la vie du personnage.



Voici une liste exhaustive des différents modules présents dans ce pack :

* Base shield : A connecter à l’Arduino pour rendre disponible Grove
* LCD RGB BlackLight : Ecran Led pour l’écriture de textes
* Sound sensor : Capte le son ambiant
* Touch sensor : Petite interface tactile
* Temperature sensor : Retranscrit la température extèrieure à l’Arduino
* LED : Une Led de chaque couleur
* Button : Un bouton poussoir pour affecter des actions spécifiques

<http://wiki.seeed.cc/Grove_Starter_kit_for_Arduino_101/>

*Finalement, Grove offre de nombreuses possibilités d’interactions ludiques et immédiates entre les programmes Hardware et Software.*

### Amélioration : La création d’un parseur

Nous avions un idéal dans ce projet, mettre en œuvre un parseur du côté Arduino.

Parseur : « Parser » signifie effectuer une action spécifique adéquate à partir d’un texte reçu dans un langage pré-établi.

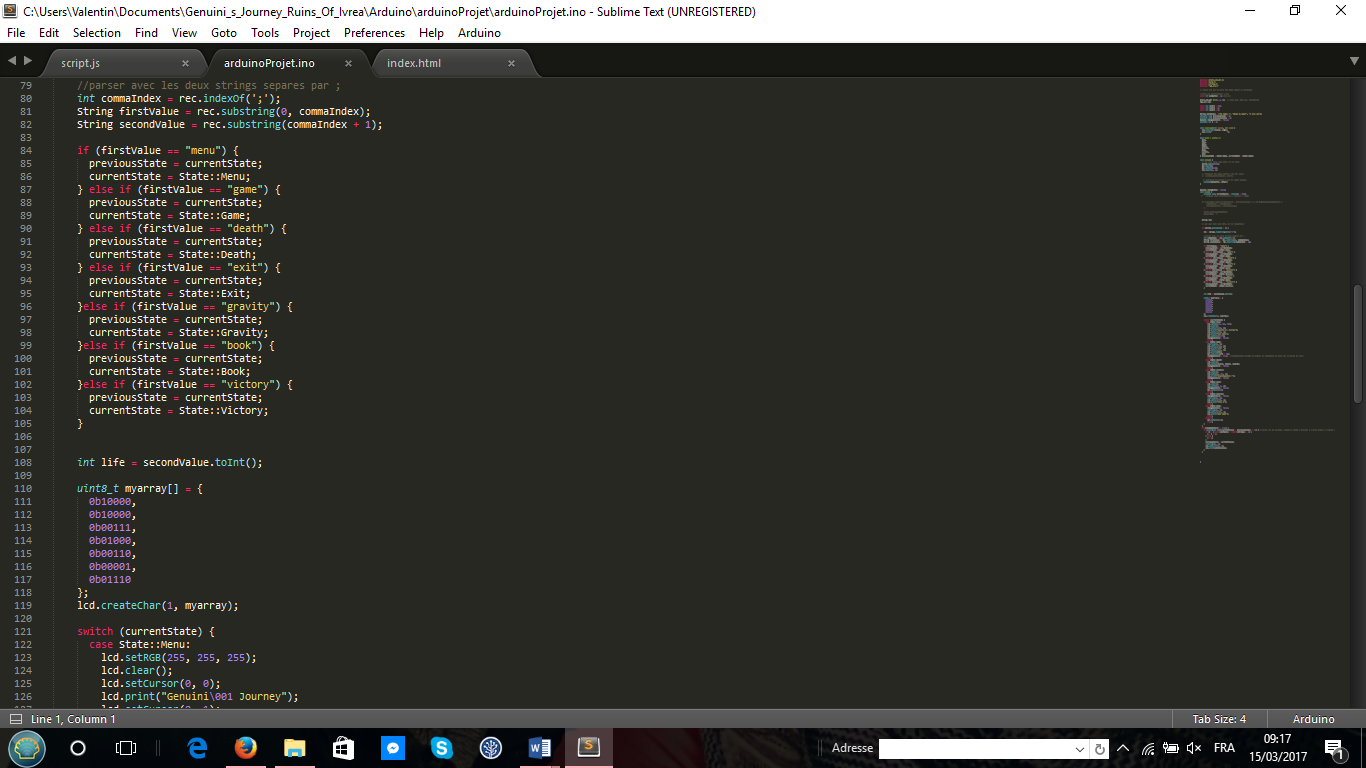
Par la suite, dans la continuité de notre développement, nous mettrons en place dans notre programme Arduino un système permettant la compréhension du langage C/C++ écrit dans le jeu par le microcontroleur. Afin de créer les premières activités d’apprentissage de l’utilisateur, nous avons commencé par vérifier si une ligne saisie par le joueur correspond à la ligne attendue pour compléter le niveau. De cette manière, notre réalisation est en adéquation avec le cahier des charges mais avant tout avec l’univers que nous voulions créer.

Figure -Conditions liées à la réception des données

*Afin de conclure, nous pouvons tous affirmer avoir apprécié la conception, la programmation mais également l’interaction que nous a procuré cette liaison entre une partie logicielle et une partie Hardware. Dans un futur même proche, nous aimerions travailler dans un secteur proche de celui-ci. Et si jamais cela ne s’avérait pas être le cas, nous expérimenterons de nouveau ce lien à travers des projets personnels qui permettront de maitriser de plus en plus le monde Arduino.*

# Système d’apprentissage

### Cours

Le joueur dispose d’un accès permanent aux cours qui sont consignés dans le grimoire. Ils portent sur des notions élémentaires d’électronique et d’informatique et permettent de comprendre le fonctionnement des différents composants ainsi que les fondamentaux de la programmation.

### Défis

Les défis sont le point clef du jeu.

Ils permettent de tester les connaissances théoriques des cours (vus précédemment par le joueur) et de les mettre en pratique.

Le principe est simple : le joueur se retrouve confronté à un problème dans le jeu, qui interrompt sa progression. Il est alors poussé à ouvrir le grimoire dans lequel l’attend un défi. Une fois ce dernier résolu, une nouvelle fonctionnalité sera alors débloquée ce qui lui permettra de surmonter les difficultés.

On distingue deux types de défi : une question de cours avec plusieurs propositions de réponse (c’est le cas du premier niveau). C’est le plus simple pour le joueur mais aussi pour la programmation. Il permet l’acquisition des connaissances basiques.

Pour le second type, le joueur doit saisir une ou plusieurs lignes de code dans un champ texte (c’est le cas du deuxième niveau).

L’utilisateur doit alors vraiment programmer l’Arduino.

Ces deux solutions ont été construites au fur et à mesure de notre compréhension de LibGDX, de la communication entre Java et Arduino et de notre capacité à structurer et factoriser le code. Le second type de question est plus complexe du fait qu’il nécessite l’analyse du code saisi par le joueur.

# Contenu Multimédia

## Gestion dans le Jeu

La classe « MainGame » créée par la classe main (située dans DesktopLauncher) charge en mémoire toutes les musiques, sons et images en mémoire. Le programme ne créé donc les sons qu’une seule fois, au démarrage, évitant des pertes conséquentes de temps -contrairement au cas où nous les chargerions à chaque appel d’un écran-.

« MainGame » se sert de la classe « Content » (du package genuini.game) afin de gérer les musiques, sons et images. Celle-ci permet d’ajouter, de modifier ou de supprimer les différentes listes (sous forme de HashMap). Elle stocke la donnée à laquelle elle attribue une clef correspondant soit à une clef passée en paramètre, soit au nom de cette donéée (par défaut).

Pour utiliser une musique (ou autres), il n’y a plus qu’à appeler un getter avec en paramètre le nom de celle-ci (la clef) au chargement de l’écran correspondant.

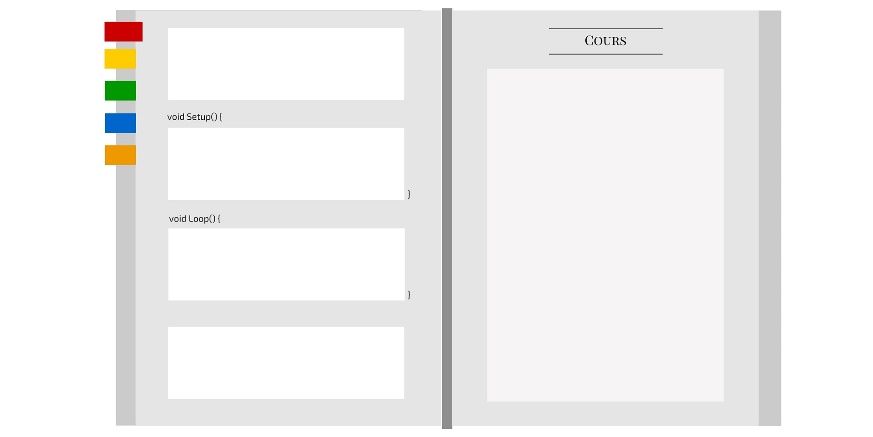
# Graphismes

### Généralités

Le choix de réaliser un jeu vidéo en tant que projet tuteuré va bien au-delà de la programmation pure. Nous attachions une importance particulière à développer une identité visuelle propre. Néanmoins, l’ampleur de la tâche s’est révélée très importante puisque nous souhaitions réellement fournir des graphismes nets, presque professionnels et n’étions pas idéalement formés ou équipés pour. Nous avons donc utilisé des spritesheets (« feuilles de sprites », fichier contenant le nécessaire à la création d’une map ou partie de map) déjà existantes, que ce soit pour le personnage principal ou les maps. Celles-ci proviennent du même auteur : blabla afin de garder une cohérence graphique.

Par la suite nous avons essayé de faire le design d’un premier personnage grâce à Inkscape. Beaucoup d’idées laissées sans suite puisqu’il était alors nécessaire de produire des maps et plateformes cohérentes, dans un style de dessin similaire, alors que nous avions en parallèle beaucoup de défis de programmation que ce soit vis-à-vis du projet ou bien de la charge de travail personnelle à l’IUT. Nous avons donc donné priorité à créer un contenu graphique que l’on pouvait exploiter directement avec le code sans perdre de temps.

En termes de création, nous n’avons également pas disposé des outils nécessaires jusqu’à la fin du projet (S4). Désormais : Krita et Tablette Huion. Nous avons réalisé un premier design d’écran de chargement, de victoire et de défaite avec et comptons étendre l’utilisation de design personnalisés. Encore une fois, l’importance d’une homogénéité de la patte graphique est primordiale et la réalisation de graphismes « sophistiquées » est une tâche, qui, n’entrant pas directement dans le cadre de notre formation de développeur, a été quelque peu réservée en bonus.



***Exemple : l’apparence du grimoire****. Nous avons préféré reprendre la maquette présentée au S2 que d’utiliser un autre skin que nous avions trouvé suite à nos recherches. Moins flagrant sur la miniature, la figure à gauche a un style beaucoup plus réaliste et antique, ce qui cassait avec le reste du jeu. Au contraire, le second grimoire est simple, sans ombres.*

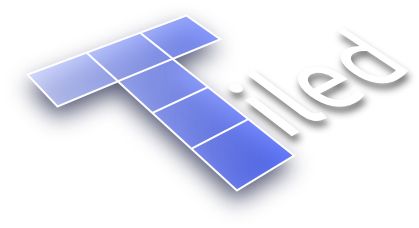
Néanmoins, les choix des sets ont été faits naturellement. S’agissant d’un Serious Game tout public, il nous fallait un design simple et efficace qui puisse plaire au plus grand nombre. De manière analogue pour les polices d’écriture, nous recherchions quelque chose de doux voir d’enfantin pour prendre le contrepied de l’image généralement véhiculée autour de l’électronique et de l’informatique, traditionnellement associés avec une certaine hostilité au grand public, très technique, très austère... Notre but étant d’ouvrir la programmation, la programmation Arduino, nous voulions une image épurée et accueillante pour que tout le monde puisse s’y retrouver sans préjugés particuliers.

### Choix des polices

**Grundshrift** : police utilisée dans le milieu scolaire en Allemagne, semblable à l’écriture manuscrite. Elle nous a plu de par sa simplicité et sa lisibilité et correspond à notre volonté didactique.

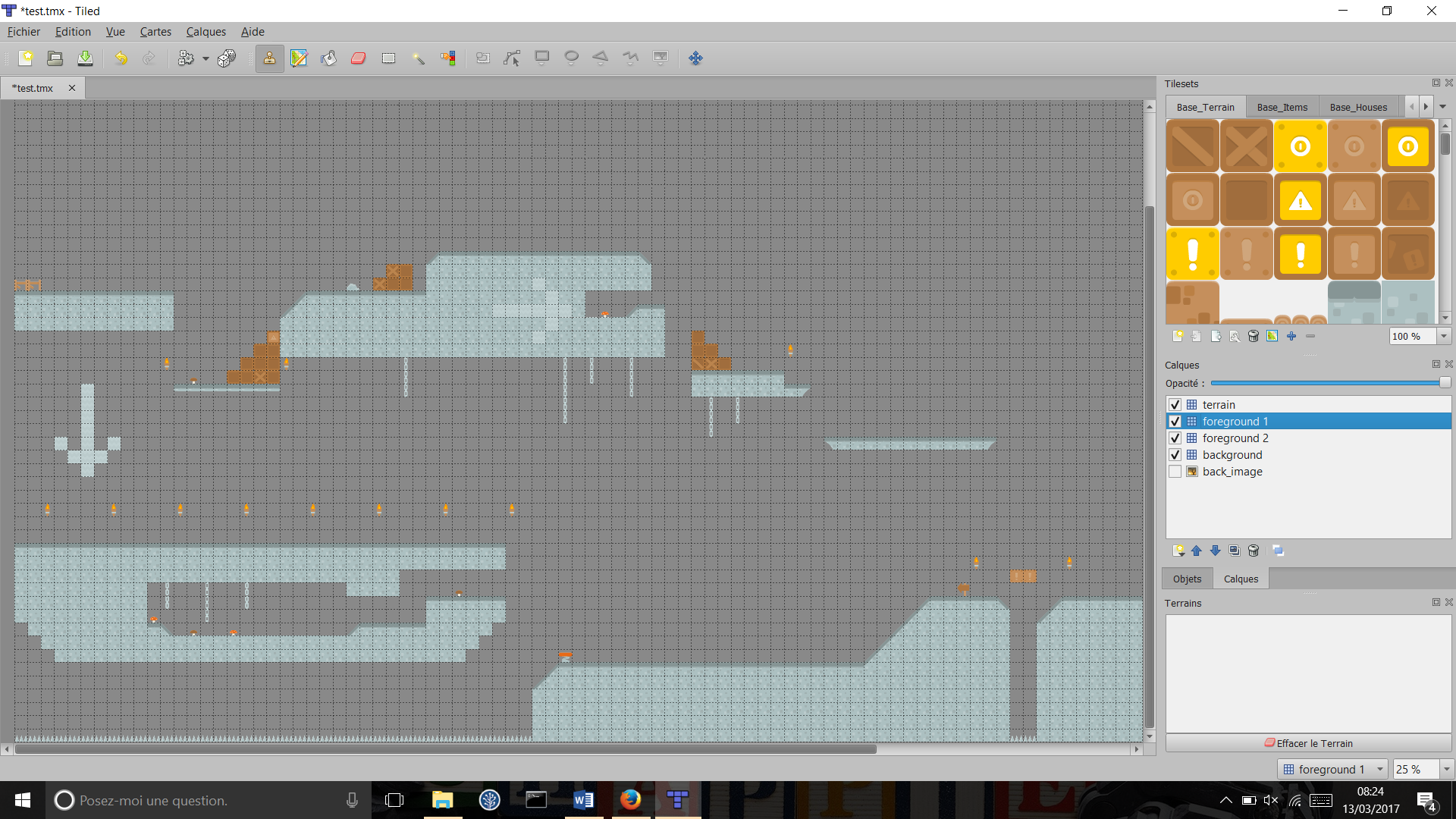
**LobsterTwo** : police sous licence SIL Open Font License.

### Outils



**TILED**

Il s’agit d’un éditeur de cartes (ou maps) libre de droit qui permet de réaliser les différents niveaux en y positionnant les différents éléments (plateformes, décors, etc. …). Il repose sur un système de couches, ou « layers ». C’est un principe simple qui s’explique naturellement : si le level design était abordé comme une illustration, il serait alors impossible de distinguer les différents éléments : comment reconnaître et appliquer des comportements particuliers aux divers éléments ? Reconnaître ce qui passe devant, derrière, au-travers etc. de notre personnage ?



***Interface de Tiled****. On remarque : le quadrillage de la map, les tilesets sur la droite ainsi que les layouts (calques)*

La carte est découpée en carrés de 70 par 70, appelés Tiles. On peut définir la taille de la carte en précisant le nombre de Tiles qu’elle comprend, en longueur et en largeur. Puis, on travaille avec les différentes couches afin de créer différents plans et profondeurs. En général, on créé une base de trois couches : le background, le foreground et le terrain (où le personnage se déplace) auxquelles on peut ajouter des variations.

Il suffit ensuite de remplir la carte à l’aide du quadrillage et de Tileset (sets d’éléments graphiques). La prise en main du logiciel est relativement simple, la difficulté réside dans la conception du niveau elle-même afin de lui donner un vrai sens sans se contenter d’aligner les blocs.

**KRITA**

Un logiciel de dessin libre (GNU General Public License version 2 et plus).

Plus intuitif que The Gimp et infiniment moins complexe que Photoshop, Krita dispose de tous les outils idéaux à la réalisation de dessins à niveau amateur ou pro. Nous l’avons peu utilisé pour le moment.

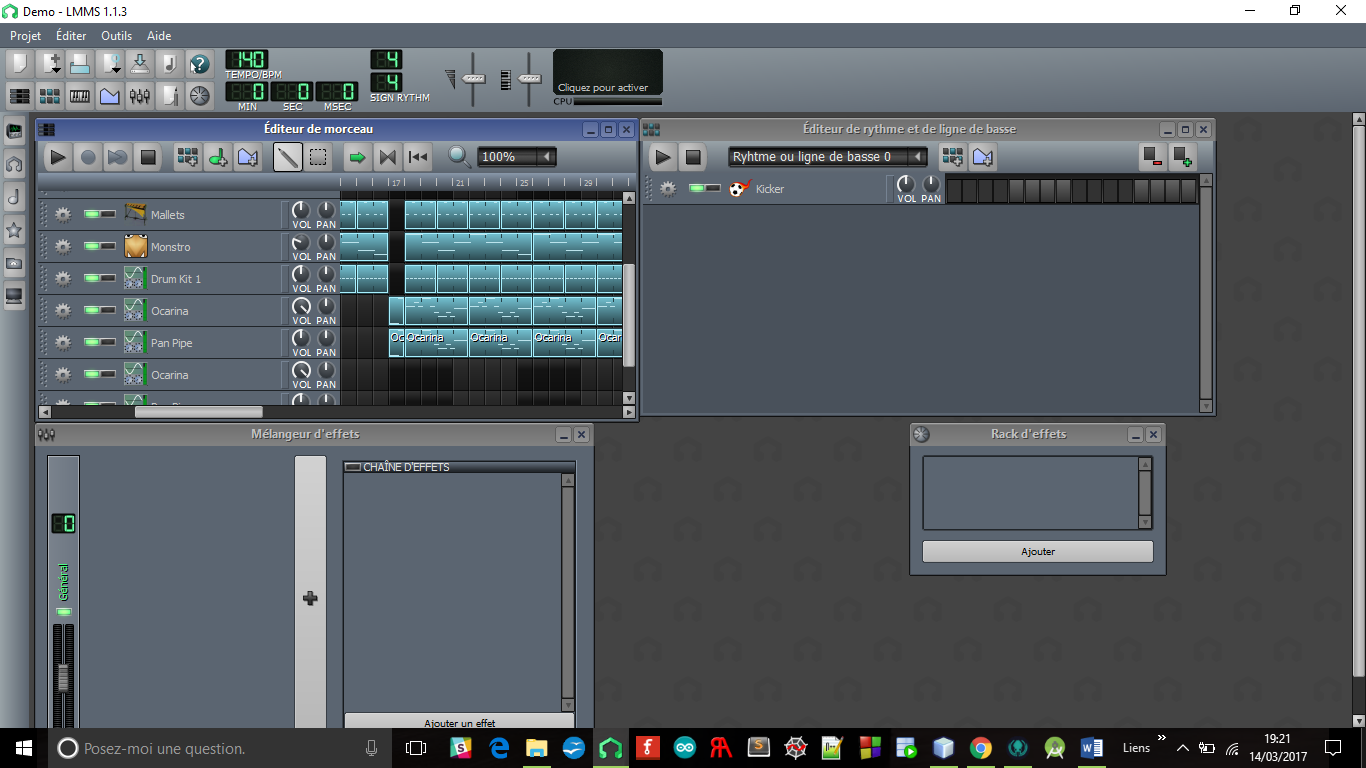
## Musiques & Sons

### Musiques

Les musiques sont –à l’instar des graphismes- un point clef d’un jeu vidéo, comme l’explique Paul-Victor Vettes, alias PV Nova dans ses vidéos (cliquer [ici](https://www.youtube.com/watch?v=7xU3B4uIyaY) et [là](https://www.youtube.com/watch?v=gTC95xzsZmU)). Ceux-ci vont influencer directement sur l’ambiance du jeu et l’état d’esprit du joueur. Comme disait Richard Wagner « La musique commence là où s’arrête le pouvoir des mots ». Ce n’est donc pas une partie négligeable. La musique peut nous permettre de plonger le joueur dans le jeu et donc de le rentre plus réceptif et ouvert à l’apprentissage.

Nous avons dans un premier temps penser à enregistrer nous même les musiques, ayant de nombreux instruments au sein de notre groupe, de types variés. Faute de temps et ayant considéré le code du jeu prioritaire, nous avons donc décider d’abandonner en partie l’idée.

En effet, chaque musique du jeu a été créer à manuellement, à l’aide du logiciel de son LMMS. Ce logiciel comporte un nombre important de sons préenregistrés, une interface simple d’utilisation et intuitive. De plus, il a l’avantage d’être gratuit et open source.



Ces musiques sont gérées de manière simple dans le code, à l’aide de LibGDX(cf. 4.1).

### SFX

Les sons et bruitages sont primordiaux dans un jeu vidéo. Ils sont au moins aussi importants que les musiques.

Les sons de notre jeu ont été réalisés à la voix pour des raisons d’efficacité.

Ils ont été enregistrés à l’aide d’un micro et du logiciel Audacity en raison de sa facilité d’utilisation et des connaissances préalables de certains de nos membres.

METTRE UN SCREEN

La gestion des sons dans le programme se fait à l’instar des musiques et des images ; à l’aide des classes « MainGame » et « Content » (cf. 4.1).

# Les obstacles rencontrés

## Organisation

Durant la phase d’attribution des tâches, nous nous étions fixés des objectifs par personne avec comme ambition que chacun participe tout de même à toutes les différentes parties du projet. Nous avions raisonné en terme de travail à accomplir et non en créneaux propices à la mise en commun des réalisations. L’absence de temps réservé dans notre planning hebdomadaire nous a handicapé pour l’organisation de réunion informative ou bien récapitulative.

Nous avons donc souffert d’un manque de relation directe entre les participants. La distance nous séparant les uns des autres n’a également pas aidé à coder simultanément durant nombres de nos temps libres comme le week-end.

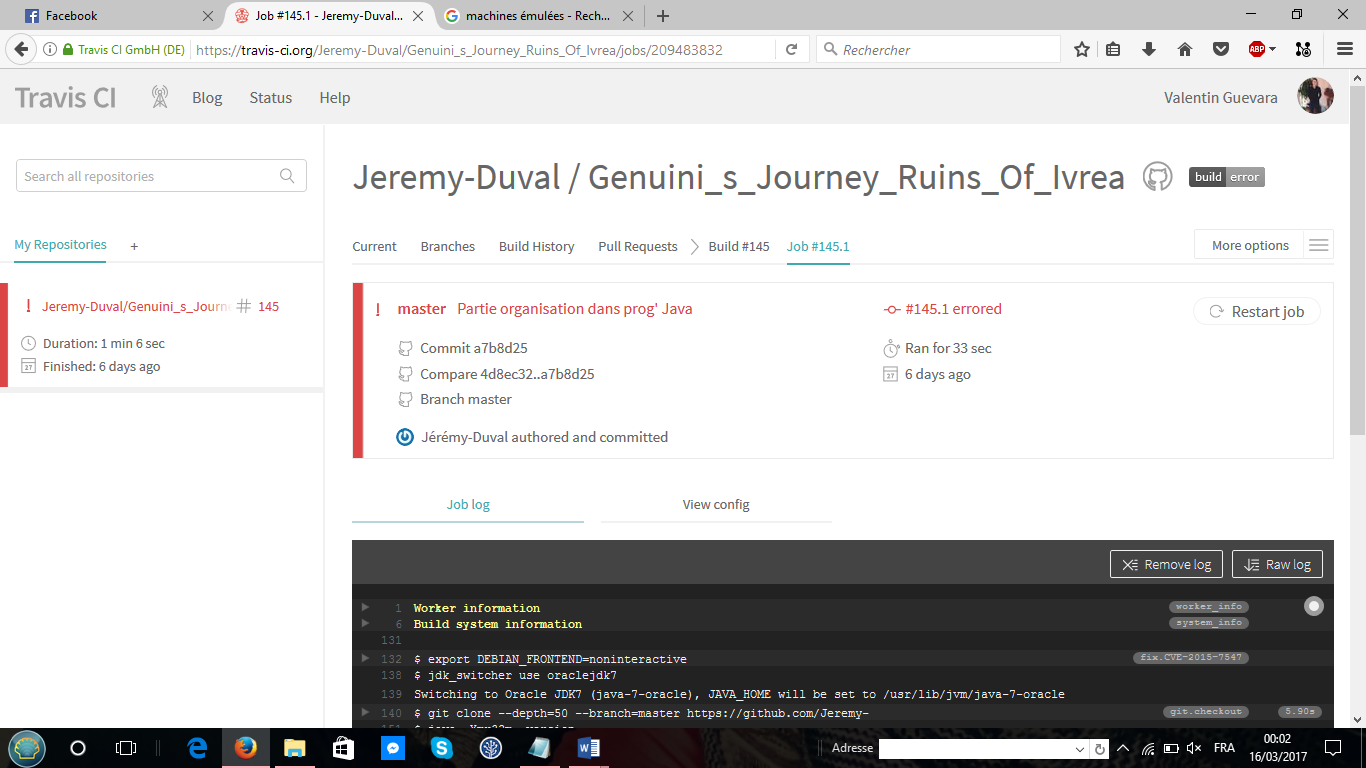
## Intégration Continu

Sur les conseils de madame Cordier, nous voulions développer notre jeu dans une démarche d’intégration continue. Notre objectif était simple, créer des parties de programme testant une unité de code et par la suite implémenter les différentes fonctions. Le logiciel devaient pouvoir être essayés sur le système développant l’application mais également les autres systèmes d’exploitation.

Pour parvenir à avoir un jeu universel, nous avons utilisé le site internet Travis-CI qui permet de créer de nombreuses machines virtuelles représentants les différents SE mais également de lancer dees tests unitaires et vérifier s’ils réussissent ou échouent.

<https://travis-ci.org/>

Nous nous attendions à ce que tout se passe convenablement lorsque Travis nous informa de nombreuses erreurs. Au premier abord, le problème nous semblait non compréhensible puis dans un second temps, nous avons compris que la problématique majeure résidait dans le fait que les machines virtuelles émulées par Travis ne parvenaient pas à installer la librairie « RXTX »(cf…).

De fait, toutes les lignes de code contenant un appel à une fonction de cette bibliothèque logicielle occasionnaient des erreurs.

## Débogage

Une fois la programmation avancée à un stade un peu plus conséquent, il était désormais temps de passer d’un écran à un autre comme par exemple jouer sur la plateforme principale puis par la suite afficher l’écran de mort. Durant un long moment, aucun problème n’avait surgi puis lors d’une victoire notre jeu s’est arrêté brutalement. Dans les informations de l’erreur était inscrit ALC\_CLEANUP.

Apres de nombreuses recherches sur internet auprés de personnes ayant été confronté au même souci mais également de tentative de débogage, nous en avons conclu qu’il s’agissait d’un problème de rendu graphique lié à la synchronisation.

A la suite d’une factorisation intense du code, nous avons pu bien comprendre les multiples origines de ce message d’erreur.

En effet, le monde virtuel lors de sa mise à jour, bloque ses composants. Or de nombreuses tentatives d’accès à des objets bloqués étaient effetuées par des composants du jeu. Il fallut donc bien séparer la mise à jour du monde et celle des autres composants du jeu. De plus, pour régler ces problèmes, une gestion directe de la destruction des corps devait être implémenté.

Bien que ce bug soit à présent très rare, une des sources de celui-ci est inhérent à l’intégration de Box 2D à LibGDX et nous pensons donc pas pouvoir le résoudre.

# Futur Développement

## Interactions avec Arduino & Défis

Lors de multiples réflexions autour du jeu, nous avons pensé à de nombreuses interactions avec Arduino et à de nombreux défis.

Les possibilités sont infinies, Arduino -étant open source- se construit sans cesse. Pour chaque nouveau composant, nous avons la possibilité de développer un nouveau niveau, toujours plus élaboré et par incidence, un nouveau défi.

Voici quelques idées d’interaction :

* Un potentiomètre pourrait nous permettre de régler l’intensité d’un sort et de l’utilisation de point de magie.
* Un potentiomètre pourrait nous permettre de régler la densité de certains objets, de certaines plateformes afin de pouvoir passer à travers ou monter dessus.
* Un potentiomètre nous permettrait de régler l’intensité lumineuse afin de faire apparaitre ou disparaitre des objets ou des ennemis.
* Un capteur photoélectrique pourrait nous permettre de faire la même chose suivant la luminosité qu’il détecte.
* Un capteur d’humidité pourrait faire pleuvoir sur une zone et, par exemple, y faire pousser un arbre afin d’accéder à une plateforme trop haute ; ou de la même manière, assécher une zone afin de créer un passage là où un arbre nous bloquait.
* Un moteur (continu, servomoteur, pas à pas) pourrait nous permettre d’actionner certains mécanismes bloqués dans le jeu comme une manivelle.
* Un gyroscope nous donnerait la possibilité de faire déplacer certaines choses ou même le personnages en inclinant la carte Arduino.
* Un télémètre nous permettrait de faire soulever le personnage ou des objets (comme des plateformes) en soulevant la carte Arduino.
* Un écran tactile ou plus simplement un écran LCD avec quelques boutons nous permettrait de faire jouer une deuxième personne qui pourrait aider la première dans le jeu, ou de déclencher des mini défis sur ces écrans.
* Un module wifi, Bluetooth, XBee pourrait permettre la communication entre deux cartes et ainsi ouvrir de multitude de possibilité. Le personnage pourrait pas exemple passer entre deux écrans.

Pour chaque type d’interaction il nous faudra des défis. Comme expliqué dans la partie correspondante (cf. 3.6.2) il y a deux familles de défis.

Les défis avec réponses proposées seraient plus présents au début de chaque cours afin de tester des connaissances théoriques ou n’étant pas directement du code (par exemple une question sur les signaux analogiques et numériques). Ils présentent de nombreuses possibilités que nous pourrons évaluer à chaque nouveau cours créé.

Les défis dans lesquels il faut programmer offrent autant de possibilités voir plus. Nous avions prévu que ce type de défi devienne de plus en plus complexe et élaborer afin de vraiment amener l’utilisateur vers l’apprentissage.

Cela commencerait donc par une question demandant simplement une ligne de code, comme déclarer une variable, initialiser une broche, allumer une led… Pour finir sur des défis où le joueur devra saisir un programme entier ; en entrant lui-même le squelette du programme et des fonctions.

## Objectifs de Jeu

## Univers

# Bilans Personnels

## Jérémy

Je pense que ce projet m’a beaucoup apporté et j’ai particulièrement aimé l’idée de pouvoir aider des personnes pour l’apprentissage d’Arduino.

Sur le plan technique, cela m’a permis de continuer à cotôyer l’univers d’Arduino que je connaissais depuis le lycée et d’être dans la continuité de mon projet du premier semestre (un poster sur Arduino).

Ensuite, j’ai vraiment pu me familiariser avec Java et apprendre à construire un projet dans ce langage : organiser les packages et les classes, constater l’importance des commentaires et de la javadoc dans un travail de groupe… Nous avons aussi dû apprendre à maitriser un nouvelle bibliothèque (libGDX) afin de mener à bien ce projet. Cela m’a permis de constater à quel point la recherche et l’apprentissage étaient des points important dans un projet.

Enfin, j’ai apprécié de pouvoir me renseigner sur les licences libres, d’en appliquer une à notre projet et de travailler sous forme d’open-source. Cela permet de contribuer aux valeurs de partage, de liberté et de donner une dimension plus humaine à notre projet.

Sur le plan humain, cela m’a permis de constater que tout n’était pas toujours facile. Il faut apprendre à persévérer avec la fatigue et les difficultés.

J’ai tout de même passé de bons moments. Construire un projet pouvant perdurer, et ce, dans la bonne entente est vraiment agréable. Avec une tutrice à l’écoute et des amis en coéquipiers, cela ne pouvait que fonctionner.

## Valentin

Lors de la première réunion, nous cherchions une idée commune, une ambition qui pourrait nous apporter l’envie de programmer en continue durant une année. Nous nous étions tous retrouvés sur la possibilité de jouer et d’apprendre dans un même temps. N’y a-t-il pas de meilleur moyen d’assimiler de nouvelles notions ?

Depuis mes débuts dans le milieu de l’informatique, la programmation de composants électroniques s’est imposée à moi, et plus particulièrement les systèmes Arduino ainsi que leurs dérivés. Notre but était de faire ressentir aux utilisateurs ces sensations que nous-mêmes avons éprouvées et éprouvons encore.

Je retire de cette réalisation de projet un sentiment positif. Dès la phase de conception, nous étions tous les quatres motivés pour parvenir à arriver au bout de cet objectif que nous nous étions fixé. Dans toute la continuité du projet, nous nous sommes heurtés à de nombreux problèmes (Travis, Bug de librairie…etc). J’ai pû tirer des enseignements de tous ces soucis et je saurai désormais réagir efficacement à ce type de situation dans un contexte professionnel.

Le semestre 3 a été, pour nous, le début de la phase de développement. J’avais hâte de programmer dans le langage JAVA. J’ai pour objectif dans un futur proche d’en faire mon langage de programmation quotidien, j’apprécie son côté haut niveau et le développement orienté objet. Son utilisation dans ce projet a donc été une excellente nouvelle d’un point de vue personnel.

En ce qui me concerne, j’éprouve certains regrets car j’ai plutôt codé dans la précipitation. Si je devais revenir en arrière, je programmerais dans la durée en étalant le travail. Mais, nous ne pouvons pas être persuadé que cette méthodologie aurait porté ses fruits. Toutefois, je suis satisfait de ce que j’ai réussi à accomplir et de ce que nous avons réussi à réaliser.

Pour conclure, cette aventure a été une réussite collective et personnelle. J’ai adoré travailler, coder, jouer aux côtés de mes trois compagnons et pouvoir communiquer avec notre tutrice.

Ce projet ne pouvait pas mieux correspondre à ce que j’espérais d’un travail en commun et j’ai énormément appris.

Je remercie donc mes collègues ainsi que Mme Cordier mais aussi toutes les personnes qui nous ont permis de mener à bien ce projet.

## Fanny

Technique

Personnellement, je ne suis pas une grande adepte de la programmation en Java et je ne peux pas dire que cette situation ait beaucoup évolué. J’ai eu quelques soucis avec Git (conflits avec le dépôt, problèmes de merge) que j’ai parfois pu gérer, mais que j’ai aussi souvent ellipsé par nécessité en lui préférant d’autres outils. Néanmoins, j’ai souvent subi l’absence de CMS et j’ai commencé à l’utiliser en parallèle personnellement afin de mieux comprendre sans mettre en péril notre dépôt et toutes ses branches…

Nous avons mis en place différents outils de gestion de projets intéressants dans un premier temps (au S2, début de S3) que nous avons délaissés : la façon de programmer un jeu était très différente de tout ce qu’on a pu voir et expérimenter et l’adaptation a été corsée. Malgré tout, je pense pouvoir affirmer que nous avons su surmonter les difficultés que nous avons rencontrées.

Humain :

Les échanges autour du projet ont toujours été très constructifs surtout durant la phase de conception. En pratique, j’ai subi une importante baisse de motivation durant certaines étapes du projet. J’ai eu beaucoup de hauts et de bas avec les différents outils (sans aucun doute liés à une mauvaise utilisation) ce qui m’a rendu la tâche fastidieuse. Les projets tuteurés sont vraiment un élément important du DUT et je trouve cela dommage qu’on ne nous réserve pas de créneaux sur l’emploi du temps pour le travailler en autonomie (excepté le jeudi après-midi, plutôt réservé aux activités sportives…).

De fait, j’ai constaté une certaine difficulté pour se coordonner, d’arriver à se voir en dehors des cours afin de travailler en équipe. C’est une problématique récurrente des travaux de groupe, et bien que je n’y ai pas été étrangère, cela m’a permis de constater son importance.

En dépit de ces difficultés, j’ai adoré présenter le projet aux visiteurs durant les Journées de l’Enseignement Supérieur. Malheureusement, je n’ai pu y prendre part qu’une journée et j’aurais aimé étendre l’expérience. Il était intéressant de voir la réaction et le retour des gens sur le jeu et de s’essayer à expliquer le plus simplement possible le concept qui n’est pas toujours évident à appréhender.

Pour finir, je dirais que le point le plus enrichissant du projet sur le plan humain est le fait que l’on se retrouve à travailler avec des gens ayant des profils très différents en termes de compétences et d’attitude, d’implication, de gestion du stress et du travail.

## Adrien

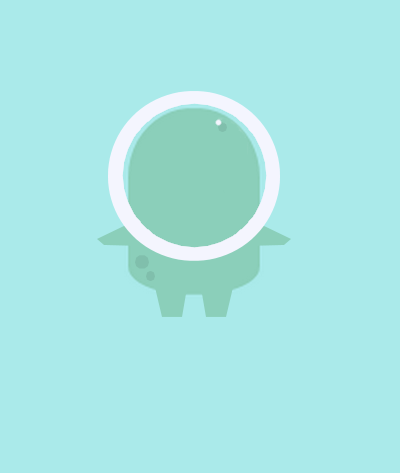
# Conclusion

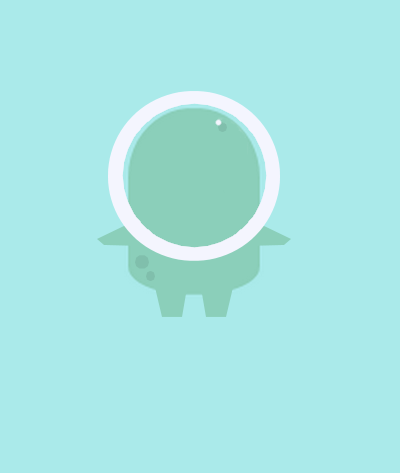
## Bilan Général

## Remerciements

# Annexes

## Diagrammes UML

+



1. CVS : Système de gestion de versions [↑](#footnote-ref-1)
2. Genuini : Personnage principal du jeu [↑](#footnote-ref-2)