# Introduction

## Contexte & Tuteur

## Présentation du projet

## Organisation

### GitHub

### Responsabilités des membres

## Droits

### Contenus multimédias et code non crées

### Licence

## Préconception

# Technologies utilisées

## Matériel Electronique

### Arduino

### Grove

## La librairie LibGDX

## La librairie RXTX

# Mécaniques du jeu

## Organisation du code et logique

### Séparation Monde/Ecran

### Séparation Tiled/Java

## Menus et changement d’écran

## Génération du monde

### Génération du Terrain

### Génération des objets

#### Types d’objets, interactions et utilité

#### Processus de création et méthode

## Les entités vivantes

### Créatures mobiles

#### Génération

#### Différents types et caractéristiques

### Joueur

#### Génération

#### Caractéristiques

## Arduino

### Interactions

## Système d’apprentissage

### Cours

### Défis

Les défis sont le point clef du jeu.

Ils nous permettent de tester nos connaissances sur les cours, vu précédemment par le joueur ; ce sont eux qui vont permettre l’apprentissage d’Arduino.

Le principe est simple : le joueur se retrouve confronté à un problème dans le jeu. Il est alors poussé à ouvrir le grimoire dans lequel l’attend un défi. Celui-ci, une fois résolu va permettre l’accessibilité à une nouvelle fonctionnalité qui, par incidence, relancera la progression du joueur au sein de niveau.

Il y a deux types de défi.

Le premier consiste en une question avec plusieurs propositions de réponse (comme dans le premier niveau). C’est le plus simple pour le joueur mais aussi pour la programmation.

Le second consiste en une question avec un champs texte servant à entrer une ou plusieurs lignes de codes (comme dans le deuxième niveau). L’utilisateur va donc devoir entrer de vraie instruction, comme s’il programmait.

Ces deux solutions ont été construite au fur et à mesure de notre compréhension de LibGDX, de la communication entre Java et Arduino et de notre capacité à structuré et factoriser le code.

En effet, la question avec proposition de réponse est simple à mettre en œuvre. Elle consiste à créer plusieurs boutons et à tester la valeur de celui sur lequel cliquera l’utilisateur.

Le second type de question est plus complexe du fait qui nécessite l’analyse du code saisi par le joueur. Nous comparons donc une chaine de caractère correspondant à la solution à celle contenant la réponse de l’utilisateur. Nous pourrions aussi aller plus loin et créer un parseur Arduino, mais ce serait peu rentable au niveau du temps pour l’instant.

# Contenu Multimédia

## Gestion dans le Jeu

La gestion du jeu se fait de manière simple et efficace.

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

### Menus et Typographie

### Textures

## Musiques & Sons

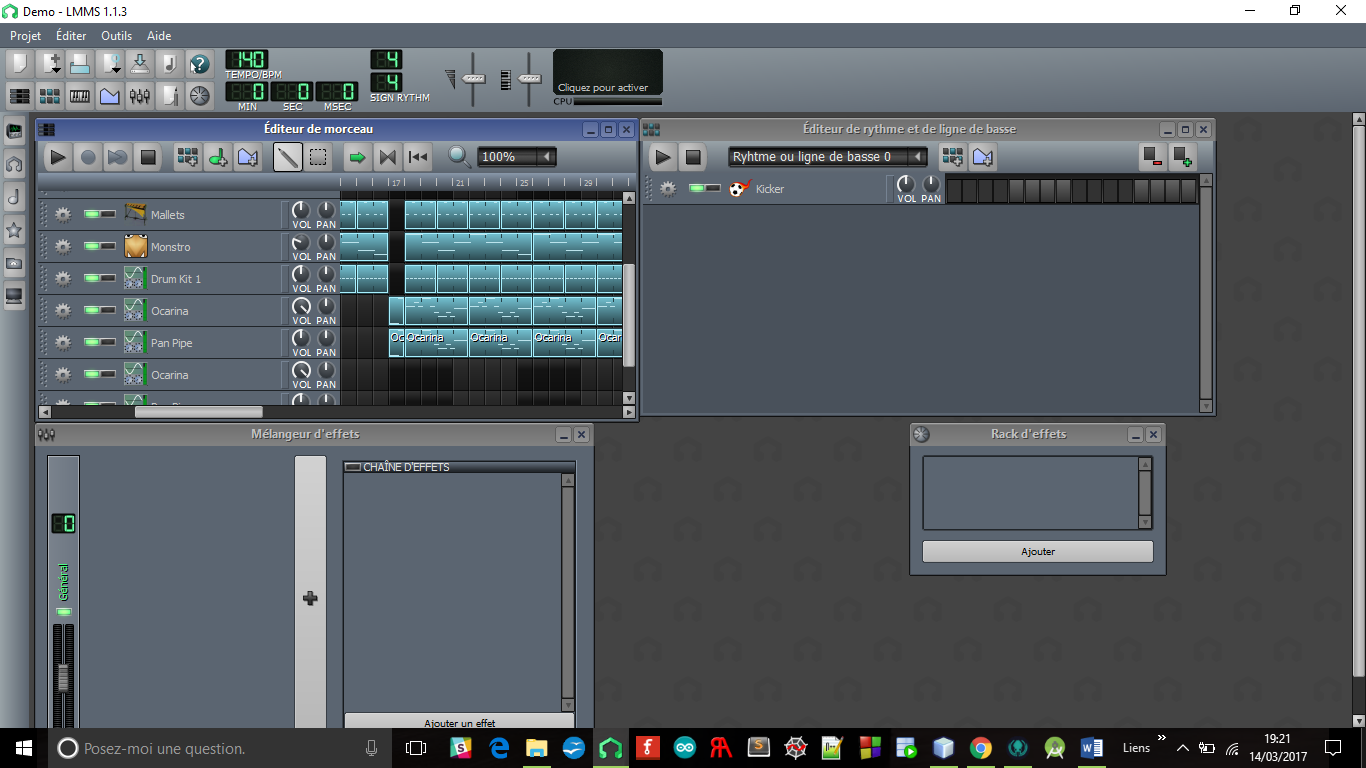
### Musiques

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 connaissance préalable de certain 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).

# Futur Développement

## Interactions avec Arduino & Défis

## Objectifs de Jeu

## Univers

# Bilans Personnels

## Jérémy

## Valentin

## Fanny

## Adrien

# Conclusion

## Bilan Général

## Remerciements

# Annexes

## Diagrammes UML