# 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

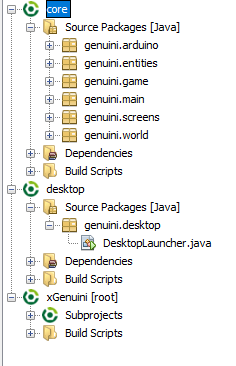
## Organisation du code et logique

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.

### 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.

#### Package Arduino

Regroupe toutes les classes

#### Package Entities

#### Package Game

#### Package Main

#### Package Screens

## 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

# Contenu Multimédia

## Gestion dans le Jeu

## Graphismes

### Menus et Typographie

### Textures

## Musiques & Sons

### Musiques

### SFX

# Les obstacles rencontrés

## Organisation & Maintenabilité

## Intégration Continu

## Débogage

Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence. — Edsger W. Dijkstra

# Futur Développement

## Interactions avec Arduino & Défis

## Objectifs de Jeu

## Univers

# Bilans Personnels

## Jérémy

## Valentin

## Fanny

## Adrien

# Conclusion

## Bilan Général

## Remerciements

# Annexes

## Diagrammes UML