

Projet Approche Objet

---

# UN JEU DE PLATEAU LABYRINTHE

---

Guillaume CHARLET  
Kenji FONTAINE  
Romain ORDONEZ  
Adrien HALNAUT

21 décembre 2017

# Table des matières

|          |                               |          |
|----------|-------------------------------|----------|
| <b>1</b> | <b>Description du projet</b>  | <b>3</b> |
| <b>2</b> | <b>Architecture du code</b>   | <b>3</b> |
| 2.1      | Main . . . . .                | 3        |
| 2.1.1    | controllers . . . . .         | 3        |
| 2.1.2    | models . . . . .              | 3        |
| 2.1.3    | views . . . . .               | 3        |
| <b>3</b> | <b>Description du travail</b> | <b>4</b> |
| 3.1      | Première séance . . . . .     | 4        |
| 3.2      | Premier pas . . . . .         | 4        |

# 1 Description du projet

Ce travail a été réalisé dans un cadre universitaire, par des étudiants en Master 1 informatique à l'université de Bordeaux. Le but de ce projet est de mettre en pratique les connaissances acquises lors du cours d'Approche Objet.

Labyrinth est un jeu de plateau dans lequel le joueur peut se déplacer et devra trouver la sortie dans un labyrinthe en franchissant divers obstacles.

## 2 Architecture du code

Le dossier src contient notre code source pour le projet, réparti dans différents sous-dossiers. On y trouve également Main, qui est exécuté lorsque le programme est lancé.

### 2.1 Main

Description de Main

#### 2.1.1 controllers

**Master** : Description de Master.

**ingame** :

- EntitySpawner : Instancie les entités du dossier drawable avec une position aléatoire.
- EventManager : S'occupe de détecter et de gérer les collisions entre entités (entre le joueur et un ennemi par exemple).
- GameManager : Gère le déroulement du jeu. Initialisation, gestion des déplacements des entités, ainsi que leur destruction (si un bonus est ramassé par exemple).
- Inputs : Reçoit les entrées utilisateurs afin de déplacer le joueur dans le labyrinthe.
- SpriteManager : Permet de récupérer les images utilisées dans l'interface graphique.

#### 2.1.2 models

**drawable** :

- Bonus : Classe des bonbons ramassables par le joueur.
- Character : Classe utilisée par le joueur ainsi que les ennemis. On distingue le joueur des ennemis à l'aide d'un booléen.
- Door : Classe de la porte de sortie utilisée dans le labyrinthe.
- Entity : Classe abstraite héritée par toutes les autres entités. Elle définit le fonctionnement générale d'une entité.
- EntityType : Énumération des différentes entités pouvant être utilisées dans le jeu.
- OnOff : Classe des boutons permettant d'activer/désactiver un mur dans le labyrinthe.
- SpriteType : Énumérations des différentes images pouvant être associées à une entité dans le jeu.

**game** :

- maze/Direction : Énumération des directions différentes qu'un mouvement peut prendre.
- maze/Menu : Classe du Menu, non utilisée.
- maze/WallType : Énumération des différents états qu'un mur peut prendre.

#### 2.1.3 views

**ViewFrame** : Permet d'initialiser l'interface graphique ainsi que de mettre l'affichage des entités à jour à chaque déplacement.

## 3 Description du travail

### 3.1 Première séance

Lors de la première séance, nous avons réfléchi sur la structure qu'allait prendre notre code. Nous nous sommes mis d'accord pour partir sur une architecture MVC (modèle, vue et contrôleur) afin de disposer d'une certaine flexibilité. Nous avons ensuite déterminé sur papier, la hiérarchie de nos classes, packages, etc... Puis nous nous sommes réparti le travail en groupe.

Voici le premier modèle retenu :

- Model
  - Drawable
    - Gentil
    - Mechant
  - Static
    - Switches
    - Doors
    - Bonus
  - Game core
    - Laby/Graph
    - Menu
- View
  - Rien pour l'instant
- Controller
  - Main/Master
  - In-Game
    - Inputs
    - Events
  - GameManager

### 3.2 Premier pas

Nous avons commencé notre implémentation par la génération de graphe (modèle) ainsi que l'affichage de la grille (vue).