|  |  |
| --- | --- |
|  | Plateformer 2D |
|  | C:\Users\malo.zollikof\Documents\HE-ARC\ProjetP2\Java\Game\Plateformer\ressources\map\tuiles\platformer-pack-redux-360-assets\Sample1.png |
| 06/06/2016 | Projet P2 – Rapport |
| Plateformer 2D  Projet P2 – Rapport | Candidats : Malo Zollikofer, Luc Wunderlin, Loïc Schwab  Classe: INF2 DLM\_B  Enseignant: Aïcha Rizzotti  Institution : HE-Arc  Laboratoire : NE 144 |

Abstract

Ce rapport présente la création d’un jeu en Java qui utilise Slick2D. Il s’agit d’un jeu de plateforme ou l’objectif est d’atteindre la clef dans un niveau pour passer au niveau suivant. Le jeu étant un jeu de plateforme, le joueur ainsi que tous les autres éléments du jeu sont soumis à la gravité. Le joueur est également confronté à des ennemies qu’il peut mettre hors d’état de nuire à l’aide d’une arme.

Table des matières

[1. Introduction 3](#_Toc452990969)

[2. Planification 4](#_Toc452990970)

[3. Slick 5](#_Toc452990971)

[4. Mise en place de l’IDE 6](#_Toc452990972)

[5. Analyse 7](#_Toc452990973)

[5.1 Structure 7](#_Toc452990974)

[5.1.1 State 7](#_Toc452990975)

[5.1.2 LevelObject 7](#_Toc452990976)

[5.1.3 Level 7](#_Toc452990977)

[5.1.4 Controller 7](#_Toc452990978)

[5.2 Graphisme et bande son 7](#_Toc452990979)

[5.2.1 Tiled 7](#_Toc452990980)

[5.3 Contrôle des personnages 7](#_Toc452990981)

[5.3.1 Clavier / Souris 7](#_Toc452990982)

[5.3.2 Manette 7](#_Toc452990983)

[5.4 Physique 7](#_Toc452990984)

[5.4.1 Applications des forces et vitesses 7](#_Toc452990985)

[5.4.2 Gestion des collisions 7](#_Toc452990986)

[5.5 Menu 7](#_Toc452990987)

[5.6 HUD 7](#_Toc452990988)

[5.7 Résolution 8](#_Toc452990989)

[6. Protocole de test 9](#_Toc452990990)

[7. Bug connu 10](#_Toc452990991)

[8. Améliorations possibles 11](#_Toc452990992)

[9. Problèmes et difficultés rencontrées 12](#_Toc452990993)

[10. Conclusion 13](#_Toc452990994)

[11. Webographie 14](#_Toc452990995)

# Introduction

Le travail présenté dans ce rapport est la réalisation d’un jeu en 2D. Le but est de se déplacer dans différent niveaux en ayant des objectifs à atteindre, tels que ramasser des pièces et également une clé. Le joueur est également confronté à des ennemis.

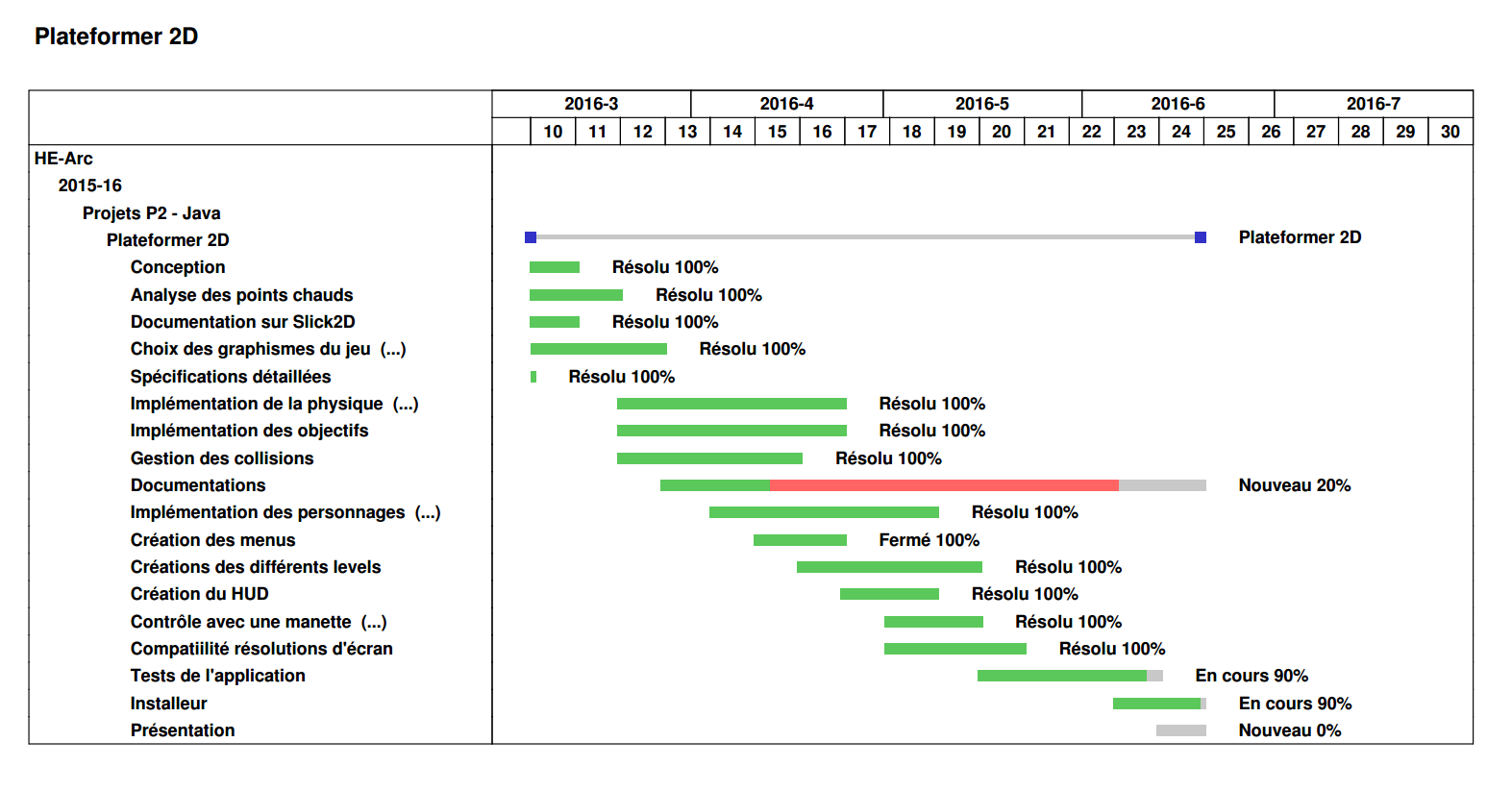
Nous avons librement choisi ce projet et l’avons réalisé dans le cadre du P2 de printemps. Il a été réalisé en Java ainsi qu’avec la bibliothèque Slick2D. L’objectif de ce projet était d’utiliser le langage Java, qui est multiplateforme, pour réaliser une application graphique.

Java a été un outil imposé mais étant donné que nous avons également un cours sur ce langage, l’expérience apportée par la création de ce projet nous a été très bénéfique. Ce langage étant très utilisé dans l’informatique, nous sommes satisfaits d’avoir eu l’occasion de l’utiliser et d’en apprendre plus sur lui.

N’ayant jamais développé de jeu vidéo, nos connaissances dans ce domaine étaient assez limitées au début du projet. De ce fait, nous avons passé passablement de temps à nous renseigner pour créer la meilleure architecture possible. Malgré ça, nous avons pu implémenter des fonctionnalités que nous avions classées comme objectifs secondaires dans notre cahier des charges.

# Planification

Notre expérience en développement de jeu vidéo étant faible, notre planning de départ n’a pas été simple à réaliser. Dès lors, nous avons fait au mieux pour le respecter mais notre planning final est quand même bien différent de celui initialement créé.



# Slick

Slick2D est une bibliothèque de programmation de jeux vidéo développé en java s’appuyant sur OpenGL via LWJGL. Bien que son auteur l’ai abandonnée car il ne code plus en Java, cette libraire est parfaitement adapté au développement de jeu vidéo.

L’avantage de faire un jeu en utilisant cet outil est la performance. Le rendu est effectué en OpenGL, ce qui est plus efficace que le rendu basique de Java.

Slick2D est un outil open source et disponible sous la licence gratuite BSD. Malgré que l’auteur l’ait abandonnée, la documentation, les tutoriels ainsi que les questions posées sur différemment forums permettent de s’y retrouver et de créer un jeu en bonne et due forme.

Les apports de Slick permettent de gagner du temps et de simplifier le développement d’un jeu en 2D. Sa principale aide est de pouvoir gérer des états. Un état peut être utilisé juste comme un menu ou comme un niveau. Chaque état implémente des fonctions de bases qui sont l’initialisation, la fonction de calcul et la fonction de rendu. Ces 2 dernières fonctions sont contenues dans une boucle de jeux.

Slick gère le chargement d’un état, son initialisation puis active la boucle de jeux jusqu’à ce que l’on décide de changer d’état (à la fin d’un niveau par exemple). Par contre, Slick n’intègre aucune mécanique de jeu (i.e. un moteur physique).

Les fonctionnalités que Slick gère sont les suivantes :

* l’affichage
* le son (format ogg)
* les contrôles (clavier, souris, joystick)
* Linux / Osx / Windows
* le chargement des images (jpg, png, etc…)
* les polices de caractère (TTF)
* l’animation des sprites et leur découpage
* l’affichage d’une carte crée sous Tiled
* transformation (rotation, échelle, translation)
* les boucles de jeux

# Mise en place de l’IDE

Pour la création de ce projet, nous avons utilisé l’environnement de développement Eclipse. La procédure d’installation est relativement simple. Voici les points à suivre :

1. Télécharger la dernière version de la bibliothèque sur le site officiel : [slick.ninjacave.com](http://slick.ninjacave.com/)
2. Décompresser l’archive
3. Créer un projet sous Eclipse
4. Dans le projet, ajouter un dossier lib, y copier les fichiers jinput.jar, lwjgl.jar et slick.jar que vous trouverez dans l’archive.
5. Ajouter ces fichiers à la bibliothèque du projet
6. Créer un dossier lib/natives, y copier tous les fichiers avec les extensions .jnilib, .dll, .so et .dylib que vous aurez également trouvez dans l’archive.

# Analyse

## Structure

Afin d’avoir un code propre et facile à maintenir, nous avons développées une structure assez complète. Grâce à cette architecture basée sur les états, il est simple d’ajouter de nouveaux niveaux. Nous avons également bien développé l’architecture autour des niveaux afin qu’il soit aisé d’ajouter des nouveaux ennemis ou autre éléments à notre jeu.

### State

Nous avons 2 sortes d’état. Les états de types menus et les états de types levelstate. Slick ne fait pas de différence entre eux mais nous oui, car nous n’allons pas y présenter la même chose.

Voici une des points importants à comprendre avant de se lancer dans l’architecture :

* La classe WindowGame instancie tous les états de jeu (menus et niveaux). Cette classe hérite de StateBasedGame
* Les états contiennent une méthode « init » appelée lors de l’instanciation
* Lorsqu’on entre dans un état, la méthode « enter » de l’état est appelée
* Lorsqu’on est dans un état, c’est les méthodes « update » puis « render » qui sont appelées en boucle
* Lorsqu’on quitte un état, la méthode « leave » de l’état est appelée
* Tous les états héritent de BasicGameState

Ensuite, chaque état à un ID unique. Grâce à ça, on peut utiliser les méthodes EnterState(ID) afin d’entrer dans un nouvel état.

#### Menu

Les menus contiennent tous un ou plusieurs boutons qui permettent d’accéder à différents états. Par exemple, le menu principal contient des boutons qui nous amèneront soit dans un état de jeu soit dans un autre état menu. Ils contiennent également pour la plupart un attribut background de type image qui correspond à l’image de fond ainsi qu’un autre attribut de type image pour le curseur. Enfin, ils peuvent contenir différents sons ou musique qui peuvent être joués lorsque l’on se trouve dans un état précis. Tous nos menus se trouvent dans le package menu.

Les boutons sont faits à l’aide de notre classe SlickBouton. Cette dernière dérive de MouseOverArea. Cette façon de faire éviter qu’on dessine des images directement sur le menu et qu’on définisse en dur des zones « cliquables ». La classe SlickBoutton que nous avons créé permet d’instancier un bouton avec une image, une taille et une position. Ensuite, c’est ce bouton qui gère les méthodes tel que MousClick ou MouseOver.

#### LevelState

Les états LevelState sont plus complets que les états menus. Premièrement, comme chaque était niveau est similaire (il y a toujours un monde, un personnage et des ennemis dans chaque niveau), nous avons fait une classe parent LevelState. Cette classe contient un nombre important d’attributs. Les principaux sont les suivants :

* Un niveau (type Level)\*
* Un joueur (type Player)\*
* Une liste d’ennemies (type Ennemie)\*
* Une liste d’objectifs (type Objective)\*
* Une arme (type Weapon)\*
* Une musique (type Music)
* Un objet physics (type Physics)\*

Tous les éléments avec une \* seront expliquées en détail plus loin dans le rapport.

Comme chaque niveau est différent (autre carte, autres ennemies, etc), mais que tous les niveaux utilises la même structure et fonctionne de la même manière (ordre d’instanciation, même physique, etc), nous avons mis en place un chemin d’initialisation de niveau :

### LevelObject

#### Character

#### Weapon

#### Objectifs

#### Projectiles

### Level

### Controller

## Graphisme et bande son

### Tiled

## Contrôle des personnages

### Clavier / Souris

### Manette

## Physique

### Applications des forces et vitesses

### Gestion des collisions

## Menu

## HUD

## Résolution

# Protocole de test

# Bug connu

# Améliorations possibles

# Problèmes et difficultés rencontrées

# Conclusion

# Webographie