Zero to Hero

Rapport de projet

Louis Brunet
Damiao Costa Santos
Baptiste Lacroix
Tuteur : M. Luc Hernandez

Rapport

Zero to Hero

Sommaire

Rapport

Zero to Hero

Introduction

Contexte

Afin de tester nos connaissances et d'apprendre à s'organiser, nous avions un projet de 6 mois à faire en parallèle aux cours. Celui-ci pouvait être sur n'importe quel sujet, personnel ou professionnel, du moment qu'il y avait assez de travail pour 3 à 5 personnes et chaque groupe devait avoir un tuteur.

Idée de projet

Notre idée était de faire un jeu vidéo. Celui-ci serait un mix entre 2 styles de jeu :
Le Roguelike : Exploration de donjons et autres endroits générés aléatoirement et procéduralement.
Le Platformer : Jeu dans lequel le personnage doit sauter de plateforme en plateforme.
Notre jeu, Zero to Hero, est donc un Platformer Roguelike, vu de profil, dans lequel le joueur doit explorer un donjon rempli d'ennemies.

Organisation du travail

Afin de travailler efficacement et rapidement, nous avions répartit les tâches comme ceci :
Damiao C.S. :
Travail sur les ennemies, sur leurs intelligences et sur leurs types.
Bastien Barbieri :
Travail sur les menus et les interfaces du jeu.
Louis Brunet :
Travail sur le moteur physique, sur le joueur et sur les armes/objets.
Baptiste Lacroix :
Travail sur les donjons, la génération de ceux-ci, la création des salles.
Travail sur les assets du jeu.
Global :
Conception des objets/armes/ennemies.
Dû à la perte d'un membre de notre groupe, Bastien Barbieri, Louis s'est occupé des menus et interfaces et Baptiste s'est occupé des registres du jeu.

Rapport

Zero to Hero

Outils utilisés

Java

Nous avons décidé de coder le jeu en Java, car on connaît bien ce langage et on voulait l'approfondir.

LibGDX

LibGDX est une librairie Java, conçue pour le développement de petits jeux vidéo.

Gradle

Nous avons appris à faire des Makefile en Java, mais pour un projet Java, nous pensons que Gradle est un meilleur moteur de production, comme celui-ci est conçu, à la base, pour Java.

Eclipse/IntelliJ

Eclipse et IntelliJ sont les deux IDE les plus connus pour développer en Java. Chacun donne accès à pleins de fonctionnalités pour faciliter la programmation en Java.

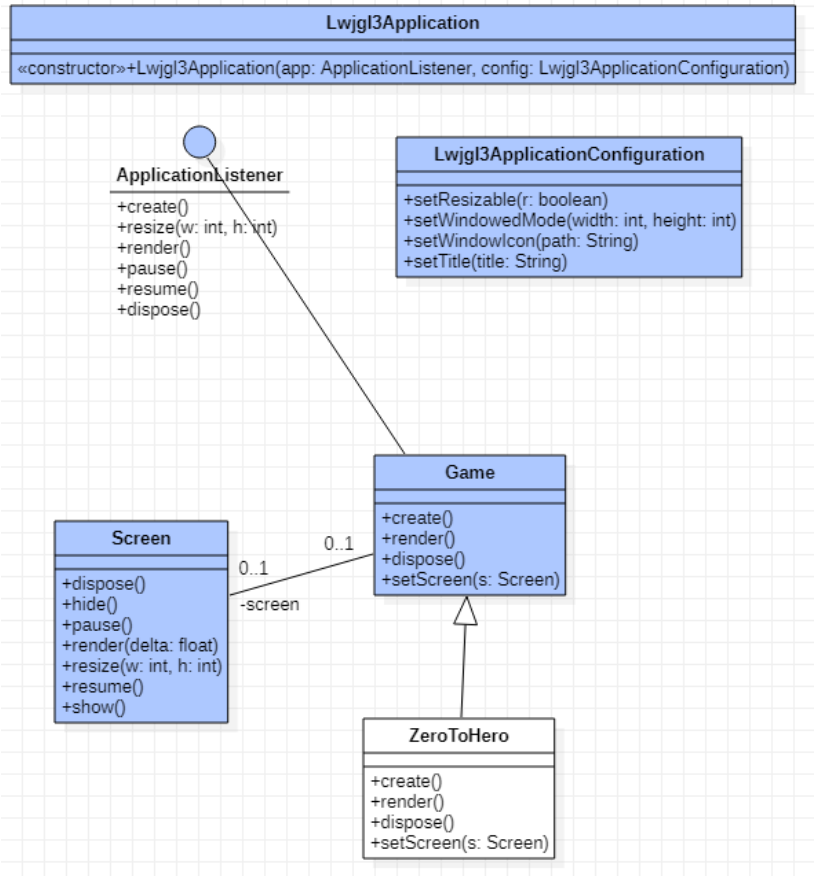
Rapport

Zero to Hero

Présentation API libGDX

Lancer une application

LibGDX utilise la Lightweight Java Game Library (LWJGL). La classe Lwjgl3Application permet de lancer le jeu sur PC.
La classe Game de libGDX représente tout le jeu. Le jeu peut afficher un écran (Screen) à la fois. L'écran à afficher peut changer en cours d'exécution.
Dans le diagramme suivant, des méthodes et attributs ont été omis pour ne montrer que ce qui est pertinent au lancement de l'application.

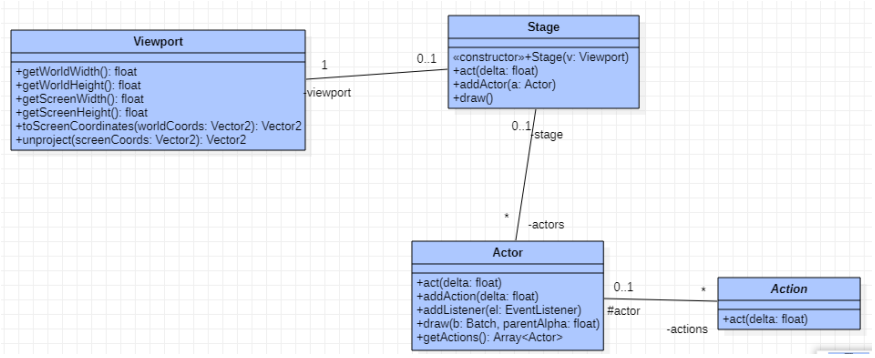


Affichage

LibGDX utilise la métaphore d'une pièce de théâtre pour sa logique d'affichage. Une scène de théâtre (Stage) contient plusieurs acteurs qui font des actions. Chaque acteur sait jouer son rôle (act) et s'afficher (draw). La scène peut ainsi être mise à jour et affichée. Chaque écran du jeu est responsable de gérer ses Stage à afficher.
Le diagramme de classes simplifié suivant illustre ceci.

Rapport

Zero to Hero



Rapport

Zero to Hero

Présentation des systèmes

Ecrans

Le jeu est divisé en deux écrans principaux : le menu principal et l'écran de jeu.

Menus et interfaces

Le menu principal est l'écran sur lequel arrive le joueur quand il lance le jeu.
Sur cet écran, il a la possibilité de lancer une partie, de changer ses paramètres, ou de quitter le jeu.
L'écran de jeu est l'écran sur lequel le joueur arrive lorsqu'il lance une partie, cet écran est divisé en deux parties : Le jeu (Arrière plan), l'interface utilisateur (Premier plan).
Le plan de jeu est composé de plusieurs couches : les tuiles (cases), le filtre de couleur d'arrière plan, les objets et les acteurs.
Quant à l'interface utilisateur, elle est divisée en plusieurs compartiments : les informations joueurs, l'inventaire, et la mini-carte. Les informations joueur montrent les statistiques du joueur (Sa vie, son porte-monnaie, etc.) et l'inventaire montre les objets que le joueur a sur lui.
L'écran de jeu peut aussi afficher une autre interface s'affichant que si le joueur appuie sur pause ou Echap : le menu pause. Ce menu contient 3 boutons : Résumé, Settings et Quit, et 2 compartiments : l'arme et l'inventaire. Le compartiment de l'arme montre l'arme en plus gros, et donne le nom et la description. L'inventaire, quant à lui, montre les objets du joueur, avec le nom et la description.

Donjon

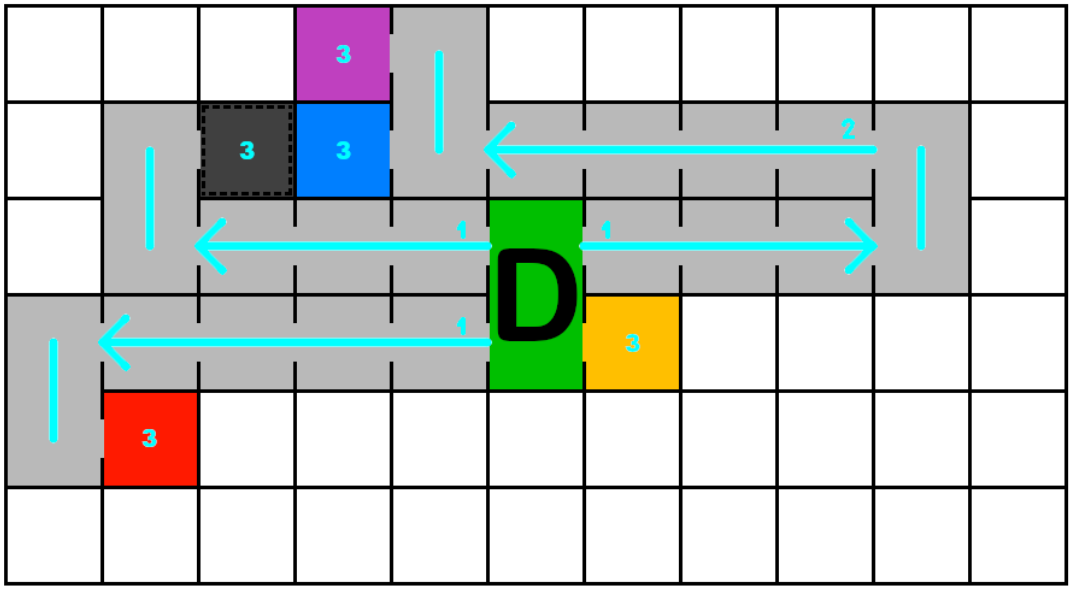
Le donjon est représentable en une hiérarchie :
- Le niveau;
- Les salles du niveau et les passages inter-salle;
- Le contenu des salles.

Tout d'abord, le niveau du donjon est créé, le générateur est exécuté et va générer des salles sur une grille de 11x6, en partant toujours du milieu (les cases 6;3 et 6;4). Le joueur apparaîtra toujours dans la salle centrale, qui, elle, sera toujours une grande salle dans laquelle le joueur est en sécurité. Le contenu de chaque est aussi généré aléatoirement selon la salle.

Rapport

Zero to Hero

Algorithme de génération

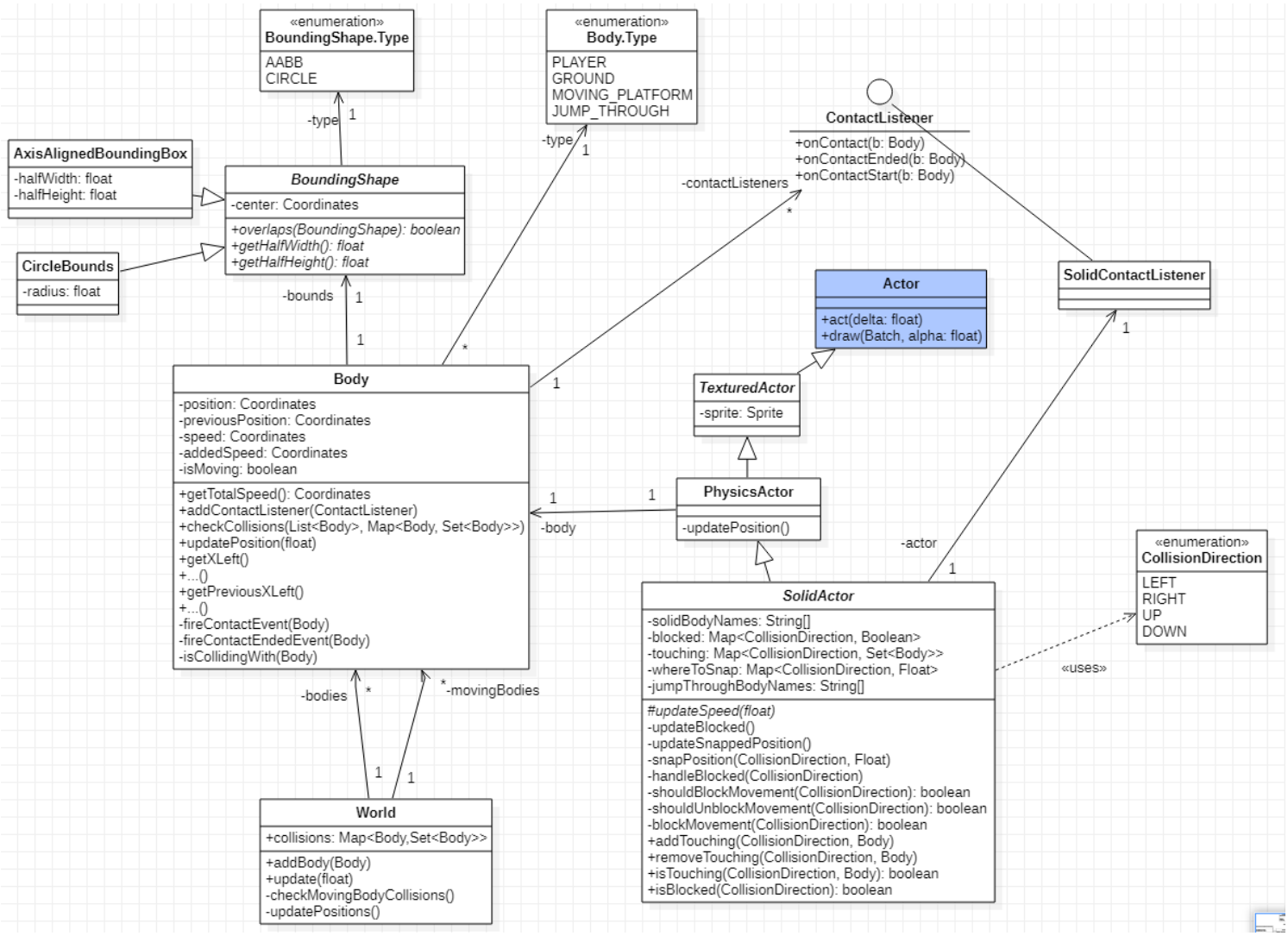


Le générateur va tout d'abord créer la salle de départ, ici noté **D**, ensuite il va décider des directions à partir desquels il va générer des salles. Il va décider du nombre de salles à générer, avec au minimum 2 salles, la dernière salle étant toujours une grande salle (ici, ce sont les flèches notées **1**).
Si le nombre de salles est en dessous de 15, alors il va choisir une grande salle au hasard, et va encore générer des salles, et ce jusqu'à ce qu'il y ai entre 15 et 30 salles (ici, ce sont les flèches notées **2**).
Une fois toutes les salles "normales" générées, il va sélectionner des emplacements où il n'y a qu'une seule salle adjacente, afin de générer les salles "spéciales" en tant qu'impasses (ici, elles sont notées **3**).
Il y aura toujours une salle spéciale de chaque type : Magasin d'armes, Magasin d'objets, salle de challenge, salle de boss, et salle secrète.

Rapport

Zero to Hero

Moteur physique



Un World est un environnement dans lequel interagissent des corps (Body). Un corps a une forme (BoundingShape), une position (son centre) et une vitesse. Quand un World est rafraîchit, les positions de ses corps sont mises à jour selon leur vitesse, puis les collisions sont calculées. Si une collision est détectée, alors tous les ContactListener ajoutés aux corps en collision sont avertis.

Un objet de la classe PhysicsActor a une référence à un corps physique. La position de l'acteur dans son Stage est mise à jour selon la position du corps.

Un SolidActor représente un PhysicsActor qui peut être bloqué par certains types de corps qu'il considère "solides". Il a aussi une liste de types de corps qui ne le bloquent que par le bas.

Rapport

Zero to Hero

Personnages

Les personnages sont les acteurs “vivants” de la scène, ils ont une intelligence artificielle et des statistiques. La statistique commune à tous les personnages est le nombre de points de vie.

Joueur

Le joueur est un personnage spécial, lié aux touches définies par l’utilisateur dans les options. Le joueur a bien une intelligence artificielle, mais celle-ci ne contrôle pas le joueur, elle est là pour comprendre les touches et indiquer au personnage “joueur” que faire.
Le joueur a quelques statistiques en plus des autres personnages, en plus de sa vie. Il a des boucliers, des pièces, des clés, des bombes, un inventaire, etc.
L’inventaire permet de stocker des objets, trouvables en jouant. Ces objets (à ne pas confondre avec les “Objets interactifs”) sont là dans le but de modifier les statistiques du joueur. Par exemple, un objet peut augmenter la vie maximale du joueur, lui changer la hauteur de ses sauts, créer plus de ramassables à la mort d’un ennemi, et plus encore.
Le joueur peut aussi avoir une arme, qui lui permet d’attaquer les ennemis de différentes manières.

Ennemis

Les ennemis sont des personnages non-joueurs pour qui le but est de tuer le joueur. Ils ont chacun une intelligence artificielle différente selon leur type. Ils n’ont que de la vie, et peuvent avoir une arme, mais rien d’autre. Leur hitbox est aussi plus large que le joueur pour permettre plus de facilité pour leur tirer dessus. Leur intelligence est plutôt limitée à simplement voir si le joueur est assez proche pour l’attaquer.

Objets interactifs

Ramassables

Les objets ramassables sont de petits objets qui peuvent apparaître après l’ouverture d’un coffre, la destruction d’une caisse ou la mort d’un ennemi. Ceux-ci sont au nombre de 4 : les cœurs, les pièces, les clés, et les bombes.

Magasins

Les magasins sont des salles spéciales qui apparaissent dans le donjon, il y en a obligatoirement 2 : un magasin d’armes et un magasin d’objets.
Le magasin d’armes vendra toujours 2 armes et 2 ramassables
Le magasin d’objets vendra toujours 2 objets et 2 ramassables

Entitées

Ce que nous appelons entitées sont des acteurs autres que des personnages, par exemple : portes, caisses, coffres, etc. Les entitées sont toutes différemment interactives, le coffre et la porte vous demanderons d'interagir avec, alors que la caisse et certains murs de pierre devront être exposés à la bombe.

Réglages utilisateur

La classe Preferences de libGDX permet de stocker des données sur la machine de l'utilisateur sous forme de fichiers XML. La classe GameSettings gère l'accès à ces fichiers depuis le code. Elle permet aussi de formater les données lues et écrites. Les préférences utilisateur stockées sont les réglages de volume et de touches clavier.

L'utilisateur peut accéder à une interface pour modifier ces réglages depuis le menu principal ou depuis le menu de pause. Cette interface scrollable contient des sliders pour modifier les volumes sonores, ainsi que des boutons pour configurer les contrôles du jeu. L'utilisateur peut enregistrer ses modifications en cliquant sur le bouton "Apply", ou les annuler en quittant les réglages. Ces boutons restent en haut de l'écran, même quand l'interface est déroulée.

A cause de limitations de libGDX, les touches sont affichées comme si le clavier de l'utilisateur était configuré en QWERTY, indépendamment de sa vraie configuration.

Registres Json

Les registres Json sont des fichiers qui contiennent les données relatives à chaque ennemi, objet, ou arme du jeu. Prenons, par exemple, le simple_bow du registre weapons :

```
{
  "simple_bow": {
    "Name" : "Simple Bow", // Nom de l'arme
    "Desc" : "This is just a test description for the simple bow.", // Description de l'arme
    "DamagesMin" : 1, // Dégâts avec puissance minimale
    "DamagesMax" : 15, // Dégâts avec puissance maximale
    "ProjectileType" : "bullet", // Type de projectile
    "ProjectileImagePath" : "weapons/simple_projectile.png", // Texture du projectile
    "ProjectileImageWidth" : 1, // Nombre de colonnes sur la spritesheet
    "ProjectileImageHeight" : 1, // Nombre de lignes sur la spritesheet
    "ProjectileSizeMin" : 0.2, // Taille avec puissance minimale
    "ProjectileSizeMax" : 0.4, // Taille avec puissance maximale
    "ProjectileSpeedMin" : 200, // Vitesse avec puissance minimale
    "ProjectileSpeedMax" : 600, // Vitesse avec puissance maximale
    "Cooldown" : 1, // Temps de recharge
    "ChargeDuration" : 1, // Durée de charge
    "Rarity" : 1, // Rareté
    "Width" : 70, // Largeur
    "Height" : 90, // Hauteur
    "Size" : 0.5, // Taille
    "IsChargeable" : true, // L'arme est-elle une arme à charger (exemple : Arc)
    "Image" : "weapons/bow_idle.png", // Texture (arme non chargée)
    "ImageWidth" : 1, // Nombre de colonnes sur la spritesheet
    "ImageHeight" : 1, // Nombre de lignes sur la spritesheet
    "ChargingImage": "weapons/bow_charging_spritesheet.png", // Texture (arme en charge)
    "ChargingImageWidth" : 6, // Nombre de colonnes sur la spritesheet
    "ChargingImageHeight" : 2, // Nombre de lignes sur la spritesheet
    "CustomBehavior": null, // Fichier .java à utiliser pour changer le comportement de l'arme
    "Currency" : "GOLD", // Monnaie à utiliser
    "Price" : 100 // Prix avec cette monnaie
  },
  ...
}
```

Ici, pleins de données sont disponibles, et sont toutes obligatoires. Il y a plusieurs types de données : les informations, les paramètres de l'arme et les projectiles.
Les informations sont les textures (spritesheets : tableaux de textures), le nom, la description, etc.
Les paramètres de l'arme sont sa taille, son temps de recharge, etc;
Les projectiles sont composés de toutes les données qui commencent par Projectile, celle-ci sont la texture, la taille, la vitesse, etc.
Si vous voulez ajouter une arme au jeu, il faudra donc juste ajouter un élément dans le json weapons.json, de même avec les objets et les ennemis.

Gestion des ressources

Afin de séparer les ressources du jeu, nous les avons mis à part, dans un dossier “assets”, séparé du dossier “src” (source). Nous avons aussi décidé de séparer les différents types de ressources dans différents dossiers. Toute ressource utilisée par le jeu est chargée au lancement du jeu, sauf les salles, qui elles, sont chargées au démarrage d’une partie.

Audio

LibGDX fait la différence entre musique et effets sonores. Les musiques peuvent être longues et sont lues sur le disque. Les effets sonores sont généralement plus courts, et tout le fichier est chargé en mémoire.

La classe AudioPlayer permet de jouer des musiques et effets sonores facilement depuis n’importe où dans le jeu.

Conclusion

Ce projet nous a beaucoup appris. Nous nous sommes rendu compte de la complexité et du taux d'efforts assez conséquent qu'il fallait y mettre. L'organisation du travail était aussi très compliquée. Il nous aurait fallu préparer un emploi du temps et décider des choses à travailler chaque semaine. Nous étions bien partis au début, et avions déjà fini une bonne partie du moteur principal du jeu en quelques semaines, mais avons vite été découragés par la suite. Nous nous sommes repris en main il y a quelques semaines, et avons réussi à finir la plupart de ce qu'on avait prévu, nous étions peut-être un peu trop ambitieux.
Pour un projet de ce style, nous pensions que M. Hernandez serait un très bon tuteur, et nous le remercions pour son aide précieuse.

Conclusions personnelles

Louis B.

J'ai aimé travaillé sur ce projet, surtout développer le moteur physique du jeu. J'ai été un peu découragé pendant un moment par l'avancement du projet, dû en partie à un manque de ressources graphiques, et par la perte d'un membre du groupe, mais je suis content des systèmes que nous avons eu le temps de mettre en place. Grâce à ce projet, je me suis rendu compte de l'importance de la planification et de l'organisation des tâches entre les membres du groupe.

Damiao C.S.

Pour moi ce projet était très agréable, Louis et Baptiste communiquent beaucoup sur ce qu'ils font, ce qui me permet de rester toujours à jour avec toute modification, dés que je ne comprend pas une fonction souvent un des deux autres est en ligne et pourrait m'expliquer comment mieux l'exploiter. Je pense que j'aurais du mettre plus de temps personnel à avancer le projet de mon côté, après le départ d'un des membres, j'ai pu apercevoir à quel point on avait encore du travail à faire, mais dans tous les cas, je pense qu'avec notre peu d'expérience dans la création de jeu, et le manque de ressources on a quand même fait un bon travail, même s'il n'est pas terminé.

Baptiste L.

Je suis très fier de ce projet, je trouve qu'on s'est bien débrouillés malgré les soucis rencontrés. Même si le jeu n'a pas énormément de contenu, nous avons tout fait pour que l'ajout de contenu soit simple, par exemple : ajouter une arme au jeu en 30 secondes.