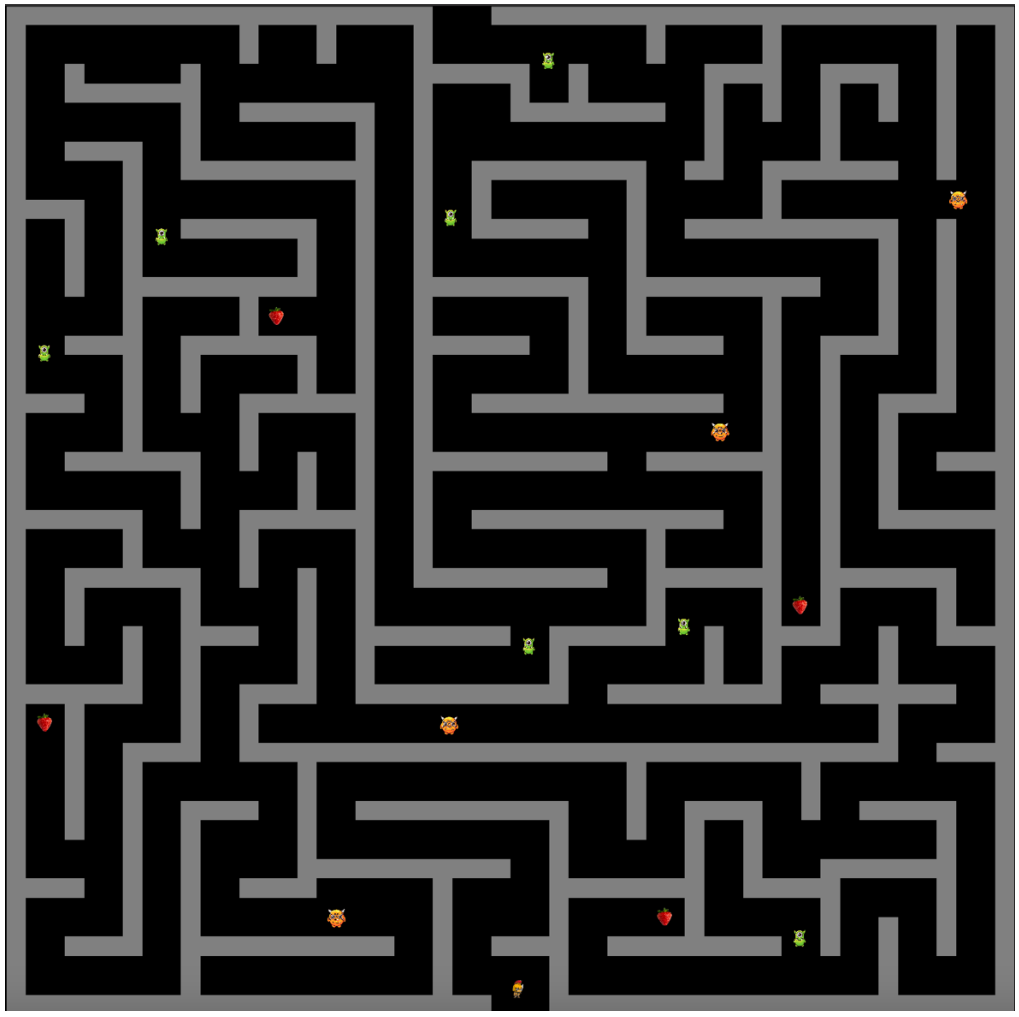
Rapport de Projet

Développement d'un Moteur de Jeu 2D avec LibGDX

MAZE GAME



Encadré par : Christopher Leturc et Stéphane Jeannin.

Table des matières

Équipe et Contributions

- **Hiba Bacha** : développeur du moteur physique
- **Rafia Ben Slama** : développeur du gameplay
- **Olesea Popa** : développeur des interfaces menu

1 – Introduction

Notre projet a pour but de créer un moteur de jeu 2D extensible utilisant LibGDX et Tiled. Il consiste en un jeu de labyrinthe interactif, dans lequel le joueur doit se déplacer pour atteindre la sortie tout en évitant des ennemis afin de ne pas perdre de vies. Le joueur a aussi la possibilité d'augmenter ses points de vies en collectant des fruits dispersés dans le labyrinthe. Un niveau se termine soit par une victoire, lorsque le joueur atteint la sortie, soit par une défaite, si ses points de vie tombent à zéro.

Objectifs

- Développer un moteur de jeu 2D permettant de charger des cartes créées dans Tiled.
- Implémenter une navigation fluide pour le joueur, incluant des collisions et des interactions avec les objets.
- Permettre la configuration des niveaux via Tiled sans modifier le code.
- Fournir une interface utilisateur intuitive pour naviguer dans les niveaux et afficher des états de jeu (victoire, défaite).

2 - Présentation du projet

Technologies et Outils Utilisés

- **LibGDX** : Développement principal du moteur de jeu.
- **Tiled** : Outil de création de cartes et de positionnement des objets tels que le joueur, les ennemis, et les fruits.
- **IntelliJ IDEA** : IDE utilisé pour écrire, tester et compiler le code.
- **GitHub** : Gestion de versions et collaboration.
- **PlantUML** : Génération de diagrammes UML.

Fonctionnalités implémentées

1. Gestion des cartes

- Chargement des cartes en format JSON générées par Tiled.

- Prise en charge des différentes couches :
    1. **Tile Layer** : couche de murs
    2. **Object Layer** : couche d'objets permettant de positionner le player, les fruits, les ennemis
    3. **EndZones** : couche de zone de fin

2. Gameplay

- Déplacement fluide du personnage : Le joueur peut se déplacer librement dans le labyrinthe tout en interagissant avec l'environnement.

- Gestion des collisions :
    - Collision avec les murs : Empêche le joueur ou les ennemis de traverser des obstacles infranchissables.
    - Collision avec les fruits : Permet au joueur de collecter des fruits, augmentant ses points de vie.
    - Collision avec les ennemis : Réduit les points de vie du joueur. Le type d'ennemi détermine la quantité de vie perdue (1 ou 2 points de vie).
    - Collision avec les zones de fin (End Zones) : Détecte si le joueur atteint la sortie, marquant la fin du niveau.

- Ennemis dynamiques : Les ennemis se déplacent horizontalement ou verticalement en fonction des propriétés définies dans Tiled.

- Collecte d'objets : Le joueur peut collecter des fruits pour augmenter ses points de vie.

- Fin de partie : lorsque le joueur fini un niveau (en gagnant ou perdant) il peut décider de rejouer le niveau, de quitter le jeu ou bien de sélectionner un autre niveau sur le menu de sélection.

3. Progression et niveaux

Le jeu comprend quatre niveaux, chacun ayant une carte unique aux tailles et dispositions spécifiques. La complexité des labyrinthes et la disposition des objets augmentent à mesure que le joueur progresse.

Les niveaux sont déblocables de manière progressive : chaque victoire débloque le niveau suivant, tandis que les niveaux non encore accessibles restent verrouillés.

Une fois débloqués, les niveaux peuvent être rejoués à tout moment via l'écran de sélection.

4. Interface utilisateur

L'interface utilisateur a été conçue pour être intuitive et esthétique. Elle inclut des écrans interactifs (chargement, sélection des niveaux, victoire, défaite), un HUD affichant les points de vie et des sons.

Configuration et Ajout de Contenu avec Tiled

Le moteur de jeu a été conçu pour être facilement extensible, permettant d'ajouter de nouveaux niveaux ou personnaliser les ennemis sans modifier le code source. Grâce à Tiled, il est possible de créer et configurer des cartes supplémentaires en quelques étapes :

1. Dessiner le labyrinthe dans la couche « Tile Layer » avec le tileset « tileset.png » disponible dans assets.
2. Positionner les objets interactifs (joueur, ennemis, fruits) sur la couche « Object Layer ».
3. Configurer le déplacement des ennemis via une propriété personnalisée que l'on nomme « movable ». Si on souhaite que l'ennemi se déplace verticalement, lui donner la valeur « vertical » sinon, par défaut, l'ennemi se déplacera horizontalement.
4. Enregistrer la carte en format .json.
5. Dans le fichier généré, ajouter la ligne `"image":"tileset.png"` après la ligne `"source":"tileset.tsx".`

Une fois les nouvelles cartes ajoutées dans le dossier des assets et référencées dans le gestionnaire de niveaux, elles sont automatiquement prises en charge par le moteur.

Compilation et exécution

Prérequis

- Java 11+.
- LibGDX SDK version 1.11.
- IntelliJ IDEA avec Gradle

Étapes de compilation

- Ouvrir le projet dans **IntelliJ IDEA**.
- Compiler le projet en exécutant la commande `./gradlew build` dans le terminal intégré.

Lancement du jeu

Pour lancer le jeu, utilisez la commande `./gradlew :lwjgl3:run`.

3 - Présentation technique du projet et contributions

Architecture Générale du Projet

Structure du Projet

Le projet est organisé selon le modèle MVC, répartissant les responsabilités entre trois principaux packages :

1. `Model` : Contient la logique métier et les données de l'application.
2. `View` : Gère l'affichage et l'interface utilisateur.

3. `Controller` : Traite les interactions de l'utilisateur et met à jour le modèle et la vue en conséquence.

Diagramme UML de l'Architecture

Le diagramme UML de notre projet est disponible à la racine du projet sous le nom « UML.png ».

Description des Packages et des Classes

Une documentation complète en Javadoc accompagne le projet. Elle détaille les packages, les classes, ainsi que leurs méthodes, et est disponible dans le dossier dédié à la documentation : **docs/javadoc/index.html**.

Model	View	Controller
▪ EndZone ▪ Enemy ▪ Entity ▪ Fruit ▪ GameMap ▪ Level ▪ LevelInterface ▪ Movable ▪ Player ▪ Tile	▪ EndScreen ▪ GameOverScreen ▪ GameScreen ▪ HUD ▪ LevelRenderer ▪ LevelScreen ▪ LevelSelectScreen ▪ LoadingScreen ▪ VictoryScreen	▪ LevelManager ▪ ScreenManager ▪ AudioManager ▪ Main

Utiliser et étendre la librairie du moteur de jeu

Notre moteur de jeu est conçu pour être facilement extensible grâce à une architecture orientée objet et à l'utilisation de LibGDX. Voici les étapes pour un utilisateur souhaitant ajouter ou modifier des fonctionnalités :

- Ajouter un niveau :
    1. Créer la carte de ce niveau sur Tiled comme expliqué précédemment.

2. Ajouter le niveau au gestionnaire des niveaux : Ouvrez la classe `LevelManager` et ajoutez une instance de votre nouveau niveau à la liste des niveaux.
3. Ajoutez l'image correspondante pour le nouveau bouton dans le dossier des assets
4. Modifier `LevelSelectScreen` de sorte qu'il crée le bouton pour ce niveau.

- Ajouter un type d'ennemis :
    1. Créer une nouvelle classe qui étends de `Enemy` dans laquelle vous spécifierez un comportement particulier pour ce type d'ennemi.
    2. Dans les cartes Tiled ajouter un objet dans la couche objets et définir un type spécifique pour cet ennemi.
    3. Modifier le chargement des ennemis dans `GameMap`, charger la classe appropriée en fonction du type défini dans Tiled.

- Modifier un élément du jeu (maps, images, sons) :
    1. Aller dans la classe AssetPaths du package assets.
    2. Changer l'attribut correspondant au chemin de l'élément.

Répartition des Tâches

Hiba Bacha : Développeur du moteur physique

- Chargement et gestion des cartes du jeu, incluant les obstacles, objets interactifs, et ennemis.
- Implémentation des collisions entre les différents éléments du jeu (joueur, ennemis, fruits, murs, zones de fin).
- Gestion du comportement des ennemis, incluant leur déplacement dynamique et leurs interactions avec le joueur.
- Gestion sonore du jeu.

Rafia Ben Slama : Développeur du gameplay

- Développement des mécaniques du joueur, incluant le déplacement fluide, les interactions avec l'environnement, et la gestion des vies.
- Implémentation de la logique de progression entre les niveaux : déblocage, transitions, et rechargement des niveaux.
- Intégration et gestion des interactions entre les composants du jeu (ennemis, objets, zones de fin).

Olesea Popa : Développeur des interfaces utilisateur

- Création des écrans du jeu : écrans de victoire, défaite, sélection de niveaux, chargement, et écran principal.
- Conception et intégration des menus et du HUD.
- Design visuel.
- Gestion des transitions entre les écrans et adaptation des interfaces pour différents formats d'écran.
- Création des cartes pour les différents niveaux avec Tiled.

La documentation Javadoc a été prise en charge par chaque membre pour les classes qu'elle a développées.

Bien que chaque membre ait eu son propre rôle dans le développement de ce moteur de jeu, nous tenions a souligné le fait que nous avons travaillé de manière collaborative en s'entraidant lorsque cela était nécessaire.

4 - Conclusion et Perspectives

Bilan

Ce projet nous a permis de concevoir un moteur de jeu de labyrinthe interactif en utilisant LibGDX, avec une gestion dynamique des niveaux grâce à Tiled. Nous avons développé quatre niveaux, avec une difficulté progressive.

Le moteur que nous avons conçu est conçu pour être extensible, permettant d'ajouter facilement de nouveaux niveaux, ennemis, et éléments interactifs. Grâce à cette architecture flexible, le jeu peut évoluer rapidement en intégrant de nouveaux contenus, tout en conservant une structure claire et maintenable.

Perspectives d'amélioration

Pour enrichir le moteur, plusieurs pistes peuvent être envisagées :

1. Animations : Ajouter des animations pour le joueur et les ennemis.
2. Fonctionnalité de mise en pause d'une partie.
3. Chronomètre : Introduire un chronomètre pour ajouter des contraintes de temps dans les niveaux.
4. Extension du moteur : Évoluer vers un moteur capable de gérer des jeux 2D plus complexes, comme des jeux de plateformes ou des RPG, grâce à des fonctionnalités avancées.