Rapport de Projet

Développement d'un moteur de jeu 2D avec LibGDX

Skeleton 5 [Tower Defense]

Équipe et contributions

- **Louis Ciebiera** – [Développeur du moteur de jeu et de l'architecture générale]
- **Adrien Guieu** – [Développeur du système de cartes et intégration avec Tiled]
- **Elvin Narimanov** – [Développeur du gameplay et des entités]
- **Michel Oyoundjian** – [Développeur de la gestion des collisions et des interactions]

Section 1. Introduction

Ce projet s'inscrit dans le cadre du développement d'un moteur de jeu 2D en Java, basé sur la librairie LibGDX, avec pour objectif principal de concevoir une architecture modulaire, extensible et orientée données.

L'idée centrale du projet est de séparer au maximum la logique du moteur du contenu du jeu, afin de permettre l'ajout ou la modification de cartes, d'entités et d'éléments de gameplay sans modifier le code Java.
Pour répondre à cet objectif, le moteur repose sur l'outil Tiled, utilisé pour la création et la configuration des cartes du jeu.

Les objectifs principaux du projet sont les suivants :

- Concevoir un moteur de jeu 2D réutilisable et extensible.
- Implémenter un système de cartes basé sur Tiled.
- Gérer différents types d'entités (joueur, obstacles, éléments interactifs).
- Permettre l'ajout de contenu via des fichiers externes.
- Proposer une architecture claire facilitant la maintenance et l'évolution du moteur.

Section 2. Présentation du projet

Technologies et outils utilisés :

- **LibGDX** : framework principal pour le développement du moteur de jeu 2D.

- **Tiled** : outil de création et de configuration des cartes.
- **Java (JDK 17)** : langage de programmation principal.
- **Gradle** : gestion de la compilation et des dépendances.
- **IDE** : IntelliJ IDEA.
- **Git / GitHub** : gestion de versions et travail collaboratif.

Fonctionnalités implémentées

Le moteur de jeu implémente les fonctionnalités suivantes :

- Chargement et affichage de cartes 2D depuis des fichiers Tiled.
- Gestion de couches de tuiles (tiles layers).
- Gestion de couches d'objets (object layers).
- Système d'entités (joueur, obstacles, éléments statiques).
- Détection de collisions simple basée sur les objets Tiled.
- Gestion du déplacement des entitées.
- Architecture modulaire permettant l'ajout de nouveaux types d'entités.
- Utilisation de musique et de sons depuis des fichiers externe.

Configuration et ajout de contenu avec Tiled

L'ajout de contenu dans le jeu se fait principalement via **Tiled**, sans modification du code Java.

Le processus est le suivant :

1. Création d'une nouvelle carte dans Tiled.
2. Définition des couches de tuiles (sol, décor, etc.).
3. Ajout de couches d'objets pour :
    - Les obstacles
    - Les zones de collision
    - Les points de spawn
4. Attribution de propriétés personnalisées aux objets (type, comportement, identifiant).
5. Export de la carte au format .tmx.

6. Chargement automatique de la carte par le moteur au lancement.

Le moteur interprète les fichiers Tiled en distinguant :

- Les couches de tuiles pour le rendu visuel.
- Les couches d'objets pour la logique du jeu.
- Les propriétés personnalisées pour configurer le comportement des entités.

Compilation et exécution

Prérequis :

- Java JDK 17 ou supérieur
- Gradle
- Environnement LibGDX correctement configuré

Étapes de compilation :

1. Cloner le dépôt GitHub du projet.
2. Ouvrir le projet dans l'IDE.
3. Lancer la tâche Gradle build.

Lancement du jeu :

- Exécuter la classe principale du module desktop. [./gradlew lwjgl3:run]
- Le jeu se lance dans une fenêtre LibGDX avec la carte chargée automatiquement.

Lien vers le dépôt GitHub : https://github.com/Louiscieb/Java-games

---

Section 3. Présentation technique du projet et contributions

Architecture générale du moteur de jeu

Le moteur repose sur une architecture modulaire composée des éléments suivants :

- **GameEngine** : point d'entrée principal du moteur.
- **ScreenManager** : gestion des écrans du jeu.
- **MapManager** : chargement et gestion des cartes Tiled.
- **Entity** : classe abstraite représentant une entité du jeu.
- **Player / Obstacle** : implémentations concrètes d'entités.
- **CollisionManager** : gestion des collisions.

- **Music/Sound** : gestion de la musique et des bruits d'ambiance.

Cette architecture permet de séparer clairement :

- Le rendu
- La logique
- Les données

Des diagrammes UML sont fournis pour illustrer les relations entre les classes principales.

Utiliser et étendre la librairie du moteur de jeu

Un développeur souhaitant utiliser ou étendre le moteur peut :

- Hériter de la classe abstraite Entity pour créer un nouveau type d'objet.
- Ajouter de nouvelles propriétés dans Tiled pour configurer le comportement.
- Implémenter de nouvelles mécaniques sans modifier le cœur du moteur.

Par exemple :

- Création d'un nouveau type de personnage.
- Ajout d'objets interactifs.
- Implémentation de bâtiments ou de zones spéciales.

Répartition des tâches

- **Louis Ciebiera : conception de l'architecture générale, gestion du moteur et du cycle de vie du jeu.**
- **Adrien Guieu : intégration de Tiled, gestion des cartes et chargement des données.**
- **Elvin Narimanov : implémentation du gameplay, gestion des entités et des collisions.**
- **Michel Oyoundjian : gestion du rendu, des entrées utilisateur et des interactions avec l'environnement.**

Section 4. Conclusion et perspectives

Ce projet a permis de concevoir un moteur de jeu 2D fonctionnel, modulaire et extensible, répondant aux objectifs initiaux.

L’intégration de Tiled a permis une séparation efficace entre le code et le contenu du jeu.

Parmi les défis rencontrés :

- La gestion correcte des données issues de Tiled.
- La conception d’une architecture suffisamment générique.
- L’organisation du travail collaboratif.
- Le respect des régles de MVC.

Perspectives d’amélioration

- Implémentation d’un moteur physique plus avancé.
- Support de scripts pour le gameplay.
- Sauvegarde et chargement de parties.
- Différents menus.

UML :