Rapport de Projet Adventure Quest

- **Etudiant :** Derbazi Mokhtar
- **Module** : `PCOO`

1. Introduction

1.1 Le projet

Adventure Quest est un moteur de jeu 2D que j'ai développé seul en Java avec LibGDX. Le but était de créer un système où on peut ajouter des cartes et des personnages sans toucher au code, en utilisant Tiled.

1.2 Objectifs

- Moteur extensible avec Tiled
- Gestion des collisions
- Sauvegarde/chargement
- Interface utilisateur complète
- Architecture propre (MVC)

2. Technologies utilisées

Langages et frameworks

- **Java 17** : Langage principal
- **LibGDX 1.12.1** : Framework de jeu
- **Tiled** : Éditeur de cartes
- **Gradle** : Build system

Outils

- IntelliJ IDEA
- Git/GitHub
- PlantUML pour les diagrammes

3. Architecture du projet

Le projet adopte une architecture MVC (Modèle / Vue / Contrôleur) afin de garantir la lisibilité, la maintenabilité et l'extensibilité du code.

Modèle

Contient les données et la logique métier :

```
- Player : position, points de vie, déplacements
- NPC : entités statiques avec dialogues
- Obstacle :
```

```
    - solide (bloquant)
    - dangereux (inflige des dégâts)
- Portal : lien entre deux cartes
- GameMap : représentation logique d'une carte Tiled
```

Contrôleur

Contient la logique de jeu et les systèmes globaux :

```
- GameController : boucle principale et orchestration
- InputHandler : gestion des entrées clavier
- CollisionManager :
    - collisions joueur  mur
    - collisions joueur  obstacles
- PortalManager : téléportation entre cartes
- SaveLoadManager : sauvegarde / restauration de l'état du jeu
```

Vue

Gère l'affichage

```
- Renderer : rendu des entités
- MapRenderer : affichage de la carte Tiled
- Caméra centrée sur le joueur avec limites de carte
```

Structure des fichiers

```
AdventureQuest/
  core/
      src/com/adventurequest/
          AdventureQuestGame.java
          controller/
          model/
          view/
          loader/
          effects/
  desktop/
  assets/
      maps/
      textures/
      sounds/
      fonts/
  build.gradle
  README.md
### Structure des packages
- com.adventurequest/
  controller/ # Logique du jeu
```

```
model/ # Données et entités
view/ # Affichage
loader/ # Chargement des ressources
effects/ # Effets visuels
```

Classes principales

Modèle

- **Entity** : Classe de base pour tous les objets du jeu
- **Player** : Le personnage principal
- **NPC** : Personnages non-joueurs
- **Obstacle** : Murs et pièges
- **Portal** : Portails entre cartes
- **GameMap** : Représente une carte Tiled
- **GameState** : État actuel du jeu

Contrôleur

- **GameController** : Orchestre tout le jeu
- **InputHandler** : Gère le clavier
- **CollisionManager** : Détecte les collisions
- **PortalManager** : Gère les changements de carte
- **SaveLoadManager** : Sauvegarde le jeu

Vue

- **Renderer** : Affiche tout à l'écran
- **MapRenderer** : Affiche les cartes
- **MenuScreen** : Menu principal
- **SaveLoadScreen** : Gestion des sauvegardes

4. Comment ça marche

4.1 Chargement des cartes

Chargement des cartes

Les cartes sont créées dans Tiled et chargées au runtime via LibGDX.

Chaque carte contient : - une couche `background` (décor) - une couche `collision` (tiles bloquantes) - une couche `entities` (joueur, PNJ, obstacles) - une couche `portals` (passages entre cartes)

Le moteur interprète automatiquement les couches et leurs propriétés pour instancier les entités correspondantes.

Système d'entités (via Tiled)

Les entités sont définies uniquement par les propriétés Tiled.

Player - défini par : `type=PLAYER` - la position initiale sert de point de respawn - points de vie : 100 PV

PNJ - `type=NPC` - propriétés : `name`, `dialogue`

Obstacles - `type=OBSTACLE` - propriété : `obstacleType` - solide : bloque le déplacement - dangereux (spike, lava, fire, danger) : inflige des dégâts

Portails - couche `portals` - propriétés : `targetMap`, `targetX`, `targetY`

Aucune modification du code n'est nécessaire pour ajouter de nouvelles entités tant que leur type est reconnu par le moteur.

Contrôles

Déplacement

- vue top-down
- déplacement continu sur 4 directions
- collisions gérées par :
  - la couche `collision`
  - les obstacles solides

Clavier

- déplacement : ZQSD ou flèches
- interaction PNJ : E ou F
- validation / menu : Entrée ou Espace

Gestion de la santé

- points de vie maximum : 100 PV
- collision avec un obstacle dangereux :
  - −15 PV par contact
  - cooldown pour éviter la perte continue
- lorsque les PV atteignent 0 :
  - le joueur est téléporté à son point de spawn
  - les PV sont réinitialisés

Technologies

Technologies utilisées

- Java (JDK 11+)
- LibGDX (framework principal)
- LWJGL3 (backend bureau)

- Tiled (éditeur de cartes)

Dépendances principales (Gradle)

- `com.badlogicgames.gdx:gdx`
- `gdx-backend-lwjgl3`
- `gdx-platform`
- `gdx-box2d` (si utilisé)

Gestion automatique via Gradle.

5. Structure détaillée du code

5.1 Fichiers Java du projet

Le projet contient **25 classes Java principales** organisées en 7 packages :

Package `model.entities` (8 classes)

- `Entity.java` - Classe abstraite de base pour toutes les entités
- `Player.java` - Gestion du joueur (déplacement, santé, spawn)
- `NPC.java` - Personnages non-joueurs avec dialogues
- `Obstacle.java` - Obstacles solides ou dangereux
- `Chest.java` - Coffres interactifs contenant des objets
- `Spike.java` - Pièges infligeant des dégâts
- `PowerUp.java` - Objets collectables (santé, vitesse, saut)
- `Portal.java` - Portails de téléportation entre cartes

Package `model.interfaces` (3 interfaces)

- `Collidable.java` - Interface pour les objets ayant des collisions
- `Interactable.java` - Interface pour les objets interactifs
- `Updatable.java` - Interface pour les objets mis à jour chaque frame

Package `model.world` (2 classes)

- `GameMap.java` - Représentation d'une carte Tiled avec couches de collision
- `Portal.java` - Gestion des portails et téléportations

Package `model` (2 classes)

- `GameState.java` - État global du jeu (joueur, carte, entités, temps)
- `MenuState.java` - État du menu principal

Package `controller` (5 classes)

- `GameController.java` - Contrôleur principal orchestrant tous les systèmes

- `InputHandler.java` - Gestion des entrées clavier avec détection de pression
- `CollisionManager.java` - Détection et résolution des collisions
- `PortalManager.java` - Gestion des transitions entre cartes
- `SaveLoadManager.java` - Sauvegarde/chargement en JSON

Package `view` (4 classes)

- `Renderer.java` - Rendu principal (entités, UI, effets)
- `MapRenderer.java` - Rendu des cartes Tiled avec caméra orthographique
- `MenuScreen.java` - Affichage du menu principal
- `SaveLoadScreen.java` - Interface de sauvegarde/chargement

Package `loader` (2 classes)

- `TiledMapLoader.java` - Chargement et parsing des fichiers `.tmx`
- `EntityFactory.java` - Factory pattern pour créer les entités depuis Tiled

Package `effects` (2 classes)

- `ParticleSystem.java` - Système de particules pour effets visuels
- `DamageNumber.java` - Affichage des dégâts flottants

Classes principales

- `AdventureQuestGame.java` - Point d'entrée LibGDX (boucle de jeu)
- `DesktopLauncher.java` - Lancement de l'application desktop

5.2 Diagrammes UML

Diagramme de classes Le diagramme de classes illustre l'architecture complète du projet :

- **Hiérarchie d'héritage** : `Entity` comme classe mère de toutes les entités (Player, NPC, Obstacle, Chest, Spike, PowerUp, Portal)
- **Implémentation d'interfaces** : Les entités implémentent `Collidable`, `Interactable` et `Updatable` selon leurs besoins
- **Pattern MVC** : Séparation claire entre Model (entités, état), View (rendu, écrans) et Controller (logique, gestion)
- **Pattern Factory** : `EntityFactory` crée les entités à partir des objets Tiled
- **Pattern Facade** : `GameController` centralise l'accès aux sous-systèmes

Les relations montrent : - Composition (`*--`) : GameController contient GameState, InputHandler, CollisionManager, etc. - Agrégation (`o--`) : GameMap contient plusieurs Entity et Portal - Dépendance (`..>`) : Les managers utilisent GameState pour leurs opérations

Diagramme de séquence Le diagramme de séquence décrit une itération typique de la boucle de jeu :

1. **Initialisation** : DesktopLauncher → AdventureQuestGame → GameController → création des managers et du GameState
2. **Boucle de jeu** (chaque frame ~60 FPS) :
   - InputHandler capture les touches et déplace le joueur
   - GameState met à jour toutes les entités (gravité, animations)
   - CollisionManager détecte les collisions (murs, obstacles, ennemis)
   - PortalManager vérifie les téléportations et change de carte si nécessaire
   - Renderer affiche la carte, les entités et l'interface (barre de vie)
3. **Sauvegarde** : Sérialisation de GameState en JSON via SaveLoadManager

Le flux montre clairement la **séparation des responsabilités** : chaque système a un rôle précis et communique via le GameState central.

Comment étendre le moteur

Pour ajouter un nouveau type d'entité :

- Créer une nouvelle classe qui hérite de Entity

public class Ennemi extends Entity implements Updatable, Collidable { // attributs et méthodes }

- Ajouter la création dans EntityFactory

private Entity createEnnemi(MapObject object) { // lire les propriétés de l'objet Tiled // créer et retourner l'ennemi }

- Ajouter un cas dans la méthode createEntity

```
case "ENNEMI":
    return createEnnemi(object);
```

Dans Tiled, créer des objets avec la propriété type=ENNEMI

- Pour ajouter une nouvelle carte :

Créer la carte avec Tiled

L'ajouter dans assets/maps/

Le jeu la chargera automatiquement quand un portail y mènera !!!!!!!

Contributions

J'ai développé seul l'intégralité du projet :

- **Architecture** Conception de toute la structure du code

- **Moteur de jeu** Implémentation de la boucle de jeu, des entités et des collisions
- **Intégration Tiled** Liaison entre l'éditeur Tiled et le code Java
- **Interface utilisateur** Création des menus et de l'affichage
- **Système de sauvegarde** Implémentation de la sauvegarde en format JSON
- **Documentation** Rédaction du rapport et des commentaires dans le code

---

Conclusion intermédiaire

Le projet a atteint ses objectifs principaux :

- Un moteur de jeu 2D fonctionnel
- L'intégration avec Tiled pour ajouter du contenu sans coder
- Un système d'entités extensible
- Une architecture propre basée sur le modèle **MVC**

Difficultés rencontrées

- **Apprentissage de LibGDX** La bibliothèque était nouvelle pour moi
- **Intégration de Tiled** Comprendre la lecture et l'exploitation des fichiers `.tmx`
- **Gestion des collisions** Assurer un fonctionnement fiable et cohérent
- **Organisation du code** Maintenir une architecture claire et évolutive

---

Perspectives d'amélioration

- Ajouter de nouveaux types d'entités *(ennemis mobiles, objets interactifs, collectibles)*
- Implémenter un système de quêtes
- Ajouter des effets sonores et des musiques
- Créer un éditeur intégré directement dans le jeu
- Porter le jeu sur **Android**

Conclusion finale

Ce projet m'a permis d'apprendre énormément sur le développement de jeux vidéo. J'ai pu mettre en pratique des concepts de **programmation orientée objet** et d'**architecture logicielle**.

Le moteur est fonctionnel et constitue une base solide pour le développement de futurs jeux 2D.