BAGRAMYAN Isabelle
BENAMER Narjes
MACHADO DA SILVA Carolina

PCOO

Rapport de Projet : Développement d'un Moteur de Jeu 2D avec LibGDX

Contexte et objectifs du projet :

Dans le cadre d'un projet en programmation et conception orientée objet, nous devons développer un moteur de jeu 2D utilisant LibGDX, une librairie de développement multiplateforme en Java. Notre objectif premier est de le rendre modulaire et extensible : il se veut être une solution souple permettant de gérer des objets et des collisions, mais permettant aussi à d'autres développeurs d'enrichir rapidement le jeu sans toucher à la structure du code source du moteur. Pour cela, nous allons apporter une attention particulière à la création et à l'importation de cartes via l'outil Tiled, un éditeur de carte pour jeu vidéo, afin que l'ajout de nouveaux éléments (personnages, objets, obstacles, etc.) puisse être assuré par héritage de classes du moteur.

Nous avons choisi d'implémenter un moteur type Shoot'em Up, un genre de jeu vidéo où le joueur doit détruire des ennemis tout en évitant leurs attaques.

Technologies et outils utilisés :

- LibGDX : Moteur de jeu principal utilisé pour le rendu graphique, la gestion des entrées, et la gestion du cycle de vie du jeu.
- Tiled : Logiciel utilisé pour la création et la gestion des cartes du jeu. Il nous permet de manipuler des couches de tuiles et d'objets pour définir les éléments du jeu sans toucher directement au code.
- IDE : Eclipse a été utilisé pour le développement du code source.
- Git : Gestion de version du code source à l'aide de GitHub pour un suivi collaboratif.
- Java 8 : Langage de programmation utilisé pour le développement du moteur.

Fonctionnalités implémentées :

- Joueur : Le joueur peut se déplacer à l'aide du clavier et tirer des projectiles (balles).
- Aliens : Des ennemis sont disposés dans une grille et se déplacent horizontalement à l'écran, se déplaçant vers le bas lorsqu'ils atteignent les bords.
- Gestion des collisions : Les balles tirées par le joueur détectent les collisions avec les aliens, et les aliens sont détruits lorsqu'ils sont touchés.
- Game Over : Le jeu se termine si un alien touche le joueur.
- Réseau de sprites : Les textures du joueur, des balles et des aliens sont gérées via des objets « Texture » de LibGDX.
- Gestion de carte : La carte du jeu est gérée via Tiled et permet d'afficher un arrière-plan statique (par exemple, un champ d'étoiles).

Configuration et ajout de contenu avec Tiled :

L'ajout de contenu dans notre moteur de jeu est simple grâce à l'intégration de Tiled. Voici comment un utilisateur peut ajouter de nouveaux niveaux :

1. Création de la carte avec Tiled : Dans Tiled, l'utilisateur crée une nouvelle carte avec des tuiles définissant l'environnement du jeu. Les personnages, objets et obstacles sont ajoutés sous forme d'objets ou de couches spécifiques.
2. Exportation et Importation dans LibGDX : Une fois la carte créée, elle est exportée en format « .tmx » et importée dans LibGDX via un fichier de configuration qui charge les objets et tuiles en fonction des coordonnées définies dans Tiled.
3. Chargement dans le moteur de jeu : Le moteur de jeu charge la carte via un MapLoader, qui interprète les tuiles et objets et crée les éléments correspondants dans le jeu. Ce processus est totalement automatisé, permettant à l'utilisateur de se concentrer uniquement sur la création de contenu.

Compilation et exécution :

Lire le fichier README disponible sur notre dépôt GitHub : https://github.com/ibramy/Projet_Pcoo_Java.git

---

Présentation technique du projet et contributions

Architecture générale du moteur de jeu :

Le moteur de jeu est structuré autour de quatre modules principaux, chacun ayant une responsabilité distincte pour assurer extensibilité et simplicité d'utilisation :

1. Classe principale : `MyGdxGame`
   - Gère le cycle de vie du jeu : création des ressources, mise à jour de l'état du jeu, rendu graphique, et libération des ressources.
   - Coordonne les interactions entre les entités (joueur, aliens, cartes Tiled) et gère les événements tels que les collisions et les conditions de victoire/défaite.
2. Entité abstraite : `GameEntity`
   - Sert de classe de base pour toutes les entités du jeu (par exemple, joueur, ennemis).
   - Fournit des méthodes génériques pour la mise à jour et le rendu, facilitant l'implémentation de nouvelles entités via l'héritage.
3. Joueur : `Player`
   - Hérite de GameEntity et implémente les comportements spécifiques au joueur, comme les déplacements horizontaux et les tirs.
   - Gère les entrées utilisateur via le clavier pour interagir avec le jeu.
4. Ennemis : `Alien` et gestionnaire des aliens
   - La classe Alien représente les ennemis, avec leur état (vivants ou morts).

Un diagramme UML représentant les principales classes du moteur et leur relation est également disponible sur notre dépôt GitHub : https://github.com/ibramy/Projet_Pcoo_Java.git

Utilisation et extensibilité du moteur de jeu :

Le moteur de jeu est conçu pour être extensible grâce à l'héritage. Voici comment ajouter de nouveaux éléments au jeu :

1. Ajouter un nouveau type de personnage :
    - Créez une classe qui hérite de GameEntity.
    - Implémentez la méthode draw() pour définir comment le personnage est affiché.
    - Surchargez update() pour gérer ses comportements spécifiques (mouvements, interactions).
2. Ajouter un nouveau type d'obstacle :
    - Créez une classe qui hérite de GameEntity ou une classe spécifique comme « Obstacle ».
    - Gérez les collisions avec le joueur ou d'autres entités en surchargeant update().
3. Ajouter un nouveau type de décor :
    - Utilisez la classe TiledMap pour ajouter des objets statiques dans la carte.
    - Si le décor doit être interactif, créez une classe qui hérite de GameEntity.

Ces principes d'héritage et de modularité permettent de développer de nouvelles fonctionnalités tout en réutilisant les mécanismes existants, rendant la librairie flexible et évolutive.

Répartition des tâches :

Isabelle BAGRAMYAN : Développement du moteur physique, centré sur la création de l'interface, gestion du joueur et des ennemis, écriture du fichier README.

Narjes BENAMER : Développement de la logique du jeu, gestion des collisions, illustration du moteur de jeu par les diagrammes UML.

Carolina MACHADO DA SILVA : Développement de l'intégration de Tiled, optimisation du système de carte, gestion du gameover, écriture du compte-rendu.

Bilan du projet :

Le projet a permis de répondre à l'objectif principal de créer un moteur de jeu 2D extensible. Nous avons réussi à développer un moteur fonctionnel qui permet de gérer des personnages et des objets créés via Tiled. Les défis rencontrés étaient principalement liés à l'apprentissage de LibGDX, à l'intégration de Tiled et à la gestion des collisions.

Perspectives d'amélioration :

Ajouter un système de sauvegarde et de chargement des parties.

Ajouter plus de types d'objets interactifs et d'effets spéciaux visuels.

Annexes

Tutoriels et documentation LibGDX : https://happycoding.io/tutorials/libgdx/
Documentation Tiled : https://mapeditor.org/