Rapport de Projet - Développement d’un Moteur de Jeu 2D avec LibGDX

Star Fighter

Jules Roy

GitHub : https://github.com/Jujuuce/StarFighter

Section 1. Introduction

Le projet vise à développer un moteur de jeu 2D « *Shoot'em Up* » extensible avec libGDX, permettant d'ajouter facilement du contenu sans modifier le code Java. Le moteur gère des entités (player, allies, ennemies) principalement via un modèle MVC. Il intègre Tiled pour créer des cartes 2D, où des éléments comme les ennemis peuvent être placé sans codage supplémentaire.

L’objectif principal est de générer un code flexible, modulable et orienté objet. Celui-ci doit être clair, robuste et avec le minimum de dépendances grâce aux nombreux patrons de conceptions inclus. Il devra générer des Exceptions en cas de problème, contenir des tests unitaires et une documentation Javadoc.

Section 2. Présentation du projet

Technologies et Outils Utilisés

- **LibGDX :** pour le développement du moteur de jeu.
- **Tiled :** pour la création et la configuration des cartes.
- **IntelliJ** : IDE couramment utilisé.
- **Javadoc libGDX** : pour la documentations.
- **OpenGameArt.org** : pour les ressources.
- **ChatGPT** : pour générer du code et la JavaDoc.

Fonctionnalités implémentées.

Navigation entre les différents écrans : Chaque écran héritant de *ScreenType* est capable de passer à un autre à tout moment en changeant son attribut *nextScreen*. L’écran de destination devra être instancié dans l’attribut *screens* de la Classe *StarFighter*, ceci est fait dans le *Main*.

Création de boutons à commande : Les écrans qui héritent de *MenuScreens* peuvent facilement créer un bouton avec une commande spécifique. De nouvelles commandes peuvent être créées en implémentant la Classe *Command*.

Lecture et écriture de JSON : Le premier écran propose le chargement d’un fichier compilé contenant les scores pour chaque niveau. Celui-ci est un JSON sauvegardé dans le répertoire

assets/Saves/. Si le fichier n'existe pas, il sera créé. Le fichier est sauvegardé à chaque fin de niveau.

Gestions des options : Possibilité de changer l'attribution des touches, si l'application est dans une fenêtre ou en plein écran et le volume global depuis l'écran des options. Cet écran est accessible depuis le menu principal et depuis le menu pause.

Plusieurs modes de jeu : Il y a deux modes de jeu possibles. Un mode classique avec l'emplacement des ennemis prédéterminé sur la carte (Level 1, Level 2, Level 3), qui se termine si le joueur n'a plus de points de vie ou si il a vaincu tous les ennemis. Un mode infini avec des ennemis qui apparaissent à des endroits aléatoires et qui ne s'arrête que lorsque le joueur n'a plus de points de vie.

Capacités des entités : Chaque entité est un Model dont il existe une Vue et un Controlleur. Elles peuvent générer d'autres entités, se déplacer et sont détruites si leurs points de vies sont à 0. Il y a des collisions entre ces entités qui leur fait perdre des points de vie.

Tests unitaires : Des tests unitaires sont disponibles surtout pour les Classes et méthodes n'utilisant pas de Batch et de Texture.

Javadoc complète.

Fonctionnalités que j'aurais souhaité implémenter

- Possibilité de choisir et modifier la taille de la fenêtre
- Transitions entre les écrans moins abrupte
- Des succès qui sont compilés dans les JSON et qui peuvent être débloqués pendant le jeu.
- Des animations pour rendre le jeu plus vivant.
- Plus de tests unitaires.

Configuration et Ajout de Contenu avec Tiled

Pour ajouter une nouvelle carte Tiled il suffit de la placer dans le répertoire assets/Levels/ et de mettre son nom dans l'écran *LevelSelectionScreen*. On peut modifier un des boutons ou juste en ajouter un nouveau.

Lorsque le niveau sera complété, le score sera ajouté à la sauvegarde actuelle.

La carte avance à l'infini verticalement tant que le jeu continu, il faudra faire une carte dans la longueur comme un couloir. La taille de la carte a peu d'importance car le jeu prend toute la largeur et fait défiler toute la longueur.

Pour placer les ennemis, il faut posséder un calque d'objet « *Objects* » et placer des points dont l'attribut '*name*' est l'ennemi souhaité. Il faut aussi ajouter ces nouveaux '*name*' dans la HashMap *enemyFactories* pour qu'ils soient correctement instanciés.

Compilation et exécution

1. **Prérequis** : Java 23, Gradle 4.4.1, Gdx 1.13.0.1

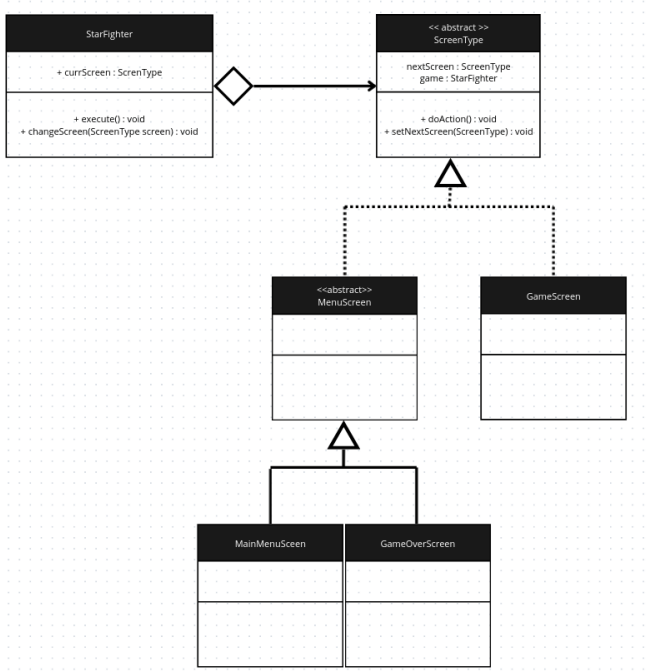
2. **Étapes de Compilation** : La compilation est gérée par le gradle inclus.
3. **Lancement du Jeu** :

- Depuis un IDE : exécuter la Classe Lwjgl3Launcher.java
- Depuis le terminal : exécuter la commande « ./gradlew lwjgl3:run » dans le répertoire StarFighter ou lancer l'exécutable « ./runGame.sh ».

Section 3. Présentation technique du projet

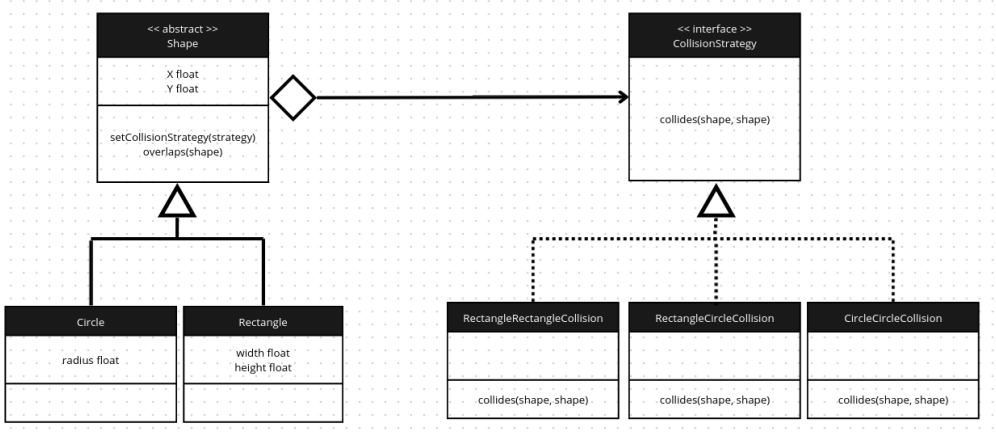
Architecture Générale du moteur de jeu

Les principaux acteurs sont les écrans qui fonctionnent ensemble comme un automate :



Ils utilisent les ressources présentent dans la Classe StarFighter.

Stratégie pour les collisions :



Utiliser et étendre la librairie du moteur de jeu

Il est possible de créer de nouveaux écrans interactifs en héritant de *ScreenType* ou *MenuScreens*.

Il est possible de générer d'autres types d'ennemis et de projectiles avec des comportements spécifiques grâce aux *Factory* utilisés.

Il est possible de rajouter des stratégies de collisions pour de nouvelles Formes qui héritent de *Shape*.

Il est possible de rajouter des commandes qui implémentent l'interface Command.

Section 4. Conclusion et Perspectives

Le jeu en lui-même fonctionne correctement cependant il reste encore de nombreuses choses à améliorer. Les plus importantes sont la remise à l'échelle lorsque l'on passe en mode plein écran et les animations des entités.

Toutefois ces améliorations ne changeront pas le principal problème qui est que ce projet n'est pas assez orienté objet. En effet, même si il inclus de nombreux patron de conception et que j'ai essayé de ne pas créer de dépendances, le résultat final n'est pas complètement modulable.

Mon problème principal est la façon dont j'ai abordé le sujet. J'ai cherché des solutions pratiques et immédiate aux défis rencontrés en pensant seulement après à extensibilité de mes packages.

Il aurait fallu créer des packages très modulables et à partir de ceux-ci créer un jeu simple. J'aurais dû faire l'intégralité de mes classes flexibles pour ensuite faire un Main qui puisse les utiliser facilement et efficacement. Malheureusement, j'ai construits certaines classes en les rendant procédurales, ce qui est loin de l'orienté objet.

Certains aspects de mon code sont tout de même orientés objet : les collisions selon les formes, les écrans et leur boutons, les nombreuses factory et le modèle MVC se rapproche de ce que j'aurais voulu accomplir. Mais ce n'est pas suffisant pour dire que mon projet est « orienté objet ».

Pour conclure je dirais que même si il y a quelques éléments correctes, il faudrait revoir le projet dans son ensemble et trouver une organisation et une mise en place moins procédurale.

J'ai tout de même grandement appris lors de la réalisation de ce travail que j'ai trouvé fort intéressant et j'ai pour objectif d'utiliser ces nouvelles connaissances dans le développement de nombreux futurs projets.

Annexes

Cours de Dr Cristopher Leturc

https://opengameart.org/