MCR Projet - Décorateur

Bijelic Alen, Bogale Tegest, Gillioz Dorian

18.06.2023

Introduction

Dans le cadre du cours de modèles de conception réutilisable (MCR), nous avons choisi d'étudier et de mettre en œuvre le modèle de conception appelé "Décorateur". Ce modèle offre une approche flexible pour ajouter dynamiquement des fonctionnalités supplémentaires à des objets existants sans les modifier directement.

Notre objectif était de comprendre en profondeur le fonctionnement du modèle et d'explorer ses avantages et ses applications pratiques. Pour illustrer son utilisation, nous avons développé un jeu appelé 'Equipper's Journey'.

'Equipper's Journey' est un jeu classique où le joueur doit collecter des équipements tels que des armures, des armes et des potions, tout en affrontant des antagonistes sur le chemin. L'objectif du jeu est de trouver des équipements pour améliorer les capacités du joueur afin de progresser et de trouver la clé, cachée dans un endroit difficile d'accès. Une fois la clé récupérée, le joueur doit atteindre la porte pour gagner le jeu et passer au niveau supérieur.

Implémentation

LibGDX

Pour ce projet, nous utilisons LibGDX pour la physique du jeu. La structure proposée par LibGDX consiste en un Launcher qui se trouve dans
desktop\src\com\decorator\game\DesktopLauncher.java
et le coeur de l'application dans
core\src\com\decorator\game\DecoratorGame.java.
Le dossier assets contient toutes les animations, images et cartes utilisées par l'application.

Joueur

La classe Player correspond au personnage que nous contrôlons. A cet égard, il possède différentes caractéristiques et comportements, permettant de gérer les animations, les actions, son contrôle et les équipements.

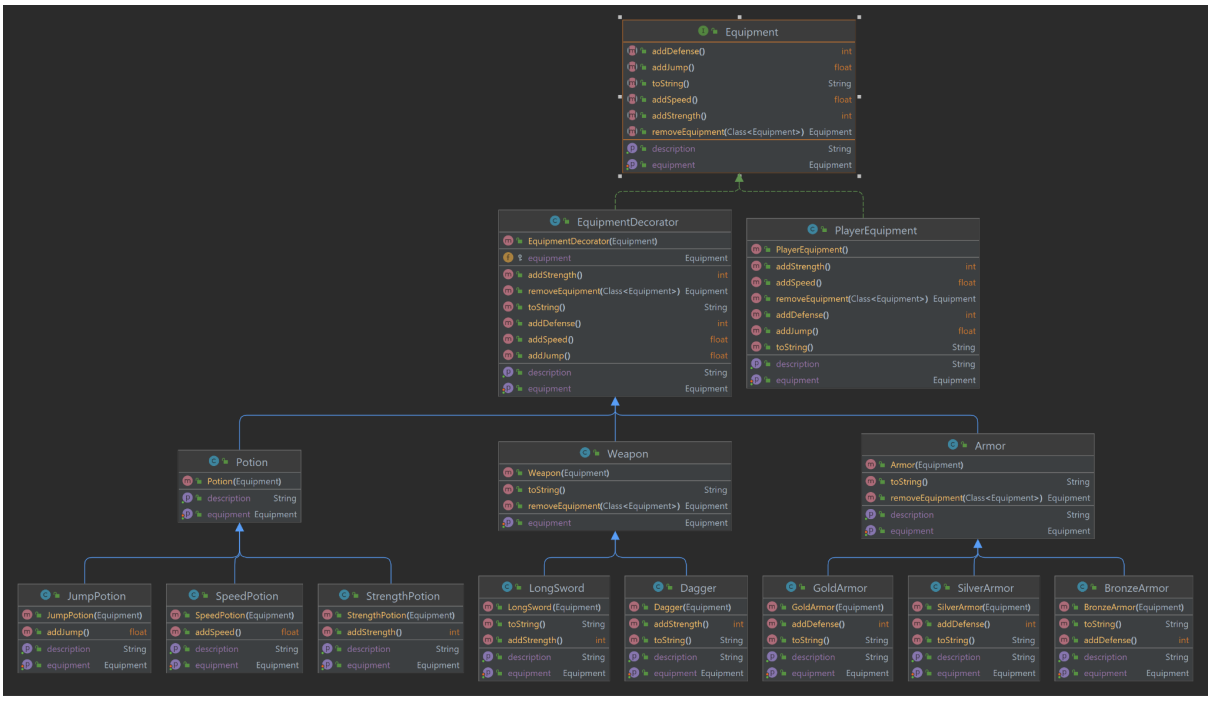
Tout d'abord, les animations sont gérées à l'aide d'un tableau multidimensionnel. Les dimensions du tableau sont les suivantes: armes, armures et actions. Toutes les images composant les animations du joueur se trouvent dans le dossier assets/player. Pour créer une animation, on crée un objet Array<TextureRegion> correspondant au frames de l'animation, puis on ajoute une à une, chaque image de l'animation. Cette phase est réalisée pour chaque action, selon l'arme et l'armure que porte le joueur, dans la méthode initAnimations.
Nous avons défini des états dans lequel se trouve le joueur. Il s'agit d'un enum et contient les états suivants: JUMPING, IDLE, RUNNING, ATTACKING ou DEAD. Ainsi nous avons une méthode getState, permettant de vérifier l'état du joueur et de le retourner. Par exemple, si le joueur se trouve en l'air, alors il est en train de sauter. Cela nous est utile notamment pour ne pas autoriser les sauts multiples. Si à un temps t, notre CurrentState était JUMPING, alors on sait qu'au temps t + 1, on n'autorisera pas à l'utilisateur de sauter. Cette méthode contient d'autres vérifications, notamment le fait de pouvoir attaquer, tout en sautant.
Pour afficher l'animation correspondant à l'état dans lequel se trouve le joueur, nous avons la méthode getFrame qui prend en paramètre le différence entre le temps actuel et le temps initial de l'animation et qui retourne un objet TextureRegion correspondant à une frame de l'animation en cours au temps t. Elle permet également d'orienter le joueur dans la bonne direction, selon son sens de marche.

La détection d'une entrée par l'utilisateur se fait dans la méthode checkUserInput. Selon les touches que le joueur appui, certains booléen vont devenir positifs ce qui va changer le comportement de la méthode getState et influencer la méthode getFrame. On utilise principalement les touches W, A, D et ESPACE, respectivement pour sauter, courir à gauche, courir à droite et attaquer. Il est également possible d'utiliser les touches HAUT, GAUCHE et DROITE. La touche X permet de mourir et n'était pas censée apparaître dans la version finale, mais elle permet de lancer l'animation pour mourir.

Toutes ces méthode sont déclenchées par les méthode update et render, utilisé par LibGDX pour mettre à jour l'affichage.

Mise oeuvre du pattern Décorateur

Nous avons utilisé le ce pattern afin d'implémenter les différents équipements que le joueur peut avoir.



Interface Equipement (Composant):
Cette interface définit les méthodes de base que tous les équipements doivent implémenter.

Classe PlayerEquipment (Composant concret):
Cette classe sert de composant de base sur lequel les décorateurs peuvent être ajoutés. Elle fournit les valeurs de base pour les différentes caractéristiques de l'équipement comme la force, la vitesse, le saut et la défense.

Classe EquipmentDecorator (Décorateur abstrait)
Cette classe est la classe de base pour tous les décorateurs d'équipement. Elle contient une référence à un objet 'Equipment' , ce qui permet d'ajouter des fonctionnalités supplémentaires aux équipements existants en les décorant avec de nouveaux équipements.

Décorateurs concrets : Armor, Potion, Weapon

La classe 'Armor' est un décorateur qui représente les trois types d'armures disponibles pour le joueur : le 'BronzeArmor', le 'GoldArmor' et le 'SilverArmor'. Ces armures

augementent la capacité de défense du joueur, offrant ainsi une meilleure protection contre les attaques ennemies.Chaque type d'armure a une valeur de défense différente:

- 'BronzeArmor' offre une défense de 30
- 'SilverArmor' offre une défense de 50
- 'GoldArmor' offre une défense de 100

La classe 'potion' est un décorateur qui représente les différentes options disponibles dans le jeu : la 'SpeedPotion', la 'JumpPotion' et la 'StrenghtPotion'. Ces potions donnent des capacités spéciales au joueur lorsqu'il les consomme.

Caractéristique des potions:
- 'SpeedPotion' augment la vitesse du joueur d'un facteur de 1.2, lui permettant de se déplacer plus rapidement
- 'JumpPotion' augmente la capacité de saut du joueur, lui permettant de sauter plus haut.
- 'StrenghtPotion' augmente la force du joueur, améliorant ainsi sa capacité de défense.

La classe 'Weapon' est également un décorateur qui représente les deux armes disponibles : la 'Dager' et la 'LongSword'. Chaque type d'arme possède un niveau de dégâts différent.
- 'Dagger' inflige 50 points de dégâts
- 'LongSword' inflige 100 points de dégâts

Comportement du pattern mis en oeuvre

Les comportement du pattern mis en oeuvre sont les suivants:

Modification du comportement existant :
- augmenter la vitesse du joueur
- augmenter la force du joueur
- augmenter la capacité de saut du joueur

Empiler des décorateurs:

- Accumulation des trois types de potion
- accumulation des potions de vitesse et de force afin de les utiliser en même temps

Substitution des décorateurs:
- remplacer une armure si la nouvelle est meilleure.

Enlever un décorateur
La méthode 'removeEquipement' dans la classe 'EquipementDecorator' est utilisée afin de supprimer un équipement spécifique de la hiérarchie des décorateurs. Elle prend en

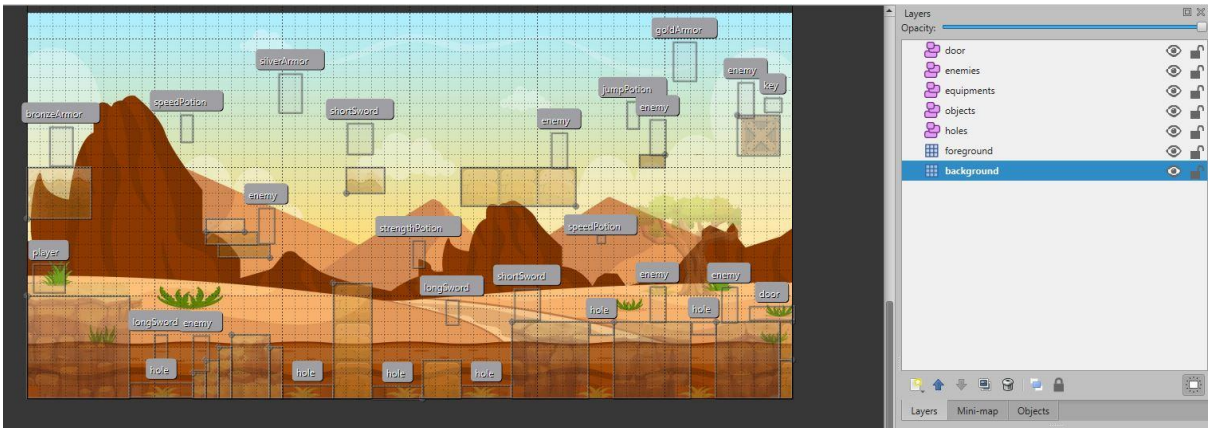
paramètre la classe de l'équipement à enlever et elle est utilisée de manière récursive pour chercher et supprimer l'équipement correspondant.
Ainsi elle permet de retirer l'équipement souhaité tout en maintenant les autres décorateurs intacts.

Map

Nous avons utilisé 'Tiled', un éditeur de map libre et open source afin de créer notre map. Nous avons tout d'abord défini la couche 'background' et inséré une image. Ensuite dans la couche 'foreground', nous avons placé les différents objets interactifs, tels que les équipements, les ennemies, la porte, les trous et les plateformes.

Ensuite, nous avons utilisé les classes 'TiledMap' et 'TiledMapRenderer' de la bibliothèque libGDX pour charger et afficher les différentes couches de la map, en utilisant les tuiles et les objets créés.

Afin de gérer les interactions appropriées avec les objets de la map, nous avons utilisé les fonctionnalités de détection de collision de libGDX. Par exemple, lorsque le joueur entre en contact avec un équipement, il le collecte en respectant les différentes règles du jeu.



HUD (Heads-up display)

Le HUD se situe au sommet de l'écran et permet d'afficher continuellement des informations à propos du joueur. Il affiche la vie du joueur, son attaque, sa défense, son armure équipée, son épée équipée, le nombre de potions de chaque consommée ainsi qu'un indicateur pour montrer s'il a récupéré la clé.

Informations au début du jeu :

1. Barre de vie du joueur => 100
2. Attaque de base du joueur => 10
3. Défense de base du joueur => 0
4. Armure de base du joueur => Aucune
5. Arme de base du joueur => Poing
6. Nombre de potions de vitesse bues => 0
7. Nombre de potions de saut bues => 0
8. Nombre de potions de force bues => 0
9. Clé ramassée ? => non

Informations après avoir ramassé des objets pendant le jeu :

1. Barre de vie du joueur => 100
2. Attaque du joueur améliorée avec l'équipement ramassé => 60
3. Défense du joueur améliorée avec l'équipement ramassé => 100
4. Armure de base du joueur => Armure en or
5. Arme de base du joueur => Dague
6. Nombre de potions de vitesse bues => 1
7. Nombre de potions de saut bues => 1
8. Nombre de potions de force bues => 0
9. Clé ramassée ? => oui

Conclusion

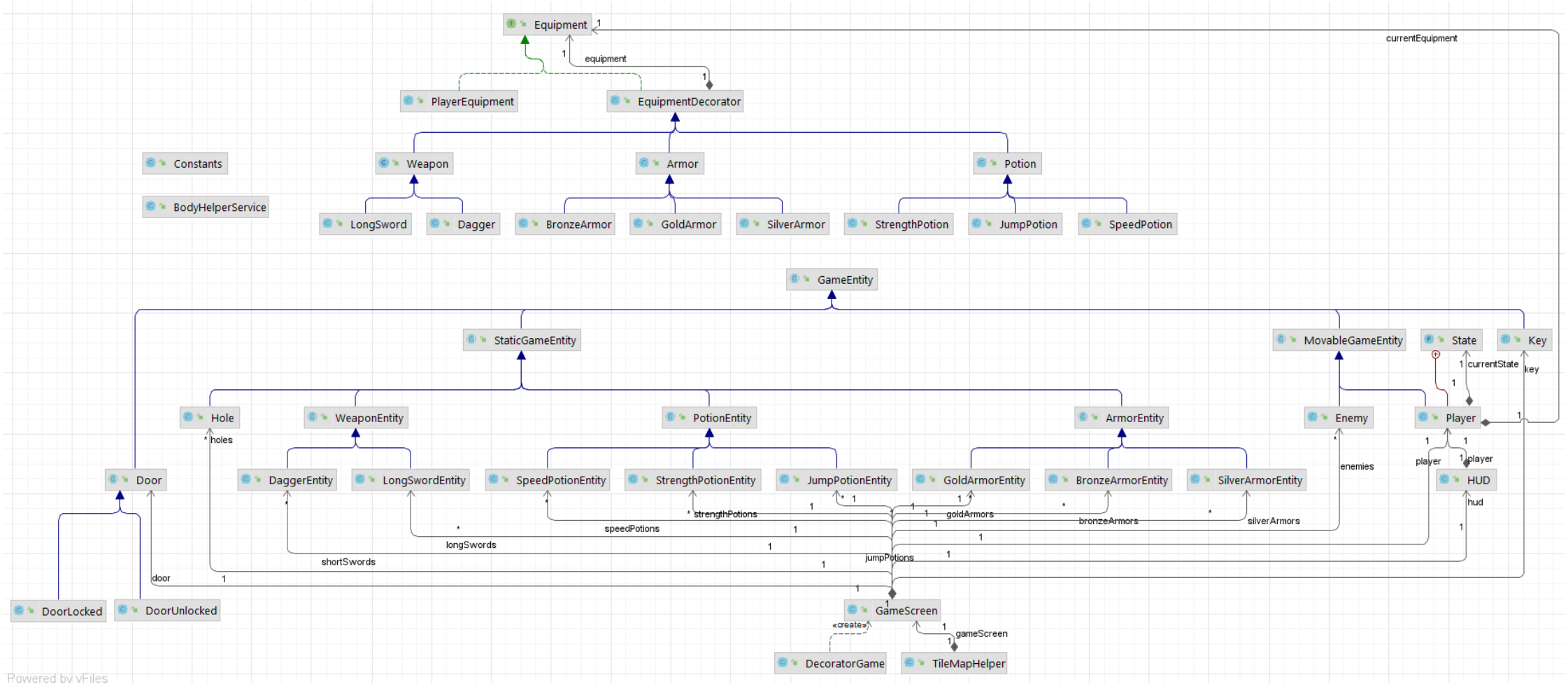
En conclusion, ce projet nous a permis de mettre en pratique le pattern décorateur, notamment pour la gestion des équipements du joueur. Chaque armure, arme et potion a été conçue comme un décorateur qui peut être ajouté ou retiré à tout moment, offrant ainsi une grande flexibilité et des possibilités de personnalisation.

De plus, l'utilisation du framework de création de jeux LibGDX s'est avérée extrêmement bénéfique. La documentation complète et les ressources disponibles en ligne nous ont grandement aidé à surmonter les défis techniques et à exploiter pleinement les fonctionnalités du framework. En explorant les aspects clés du jeu tels que la physique, les animations et l'interface utilisateur (HUD), nous avons pu obtenir une compréhension solide du fonctionnement d'un jeu de plateforme.

Cependant, en raison de contraintes de temps, nous n'avons pas pu intégrer les ennemis et les mécanismes de dégâts. Notre priorité était de nous concentrer sur le bon fonctionnement du pattern décorateur, qui constituait l'élément central de notre projet. Malgré cette limitation, nous sommes satisfaits des résultats obtenus et des connaissances acquises tout au long du processus de développement.

Annexes

UML


Equipment
currentEquipment
equipment
PlayerEquipment
EquipmentDecorator
Constants
BodyHelperService
Weapon
Armor
Potion
LongSword
Dagger
BronzeArmor
GoldArmor
SilverArmor
StrengthPotion
JumpPotion
SpeedPotion
GameEntity
StaticGameEntity
MovableGameEntity
State
Key
currentState
key
Hole
WeaponEntity
PotionEntity
ArmorEntity
Enemy
Player
holes
Door
DaggerEntity
LongSwordEntity
SpeedPotionEntity
StrengthPotionEntity
JumpPotionEntity
GoldArmorEntity
BronzeArmorEntity
SilverArmorEntity
enemies
player
HUD
hud
strengthPotions
goldArmors
bronzeArmors
silverArmors
speedPotions
longSwords
shortSwords
jumpPotions
door
DoorLocked
DoorUnlocked
GameScreen
«create»
gameScreen
DecoratorGame
TileMapHelper
Powered by yFiles

Manuel d'installation

Notre projet se trouve sur Github et est accessible publiquement avec le lien suivant: https://github.com/AlenBijelic99/MCR_Decorator_Project

Notre projet utilise Gradle, il faut donc s'assurer préalablement que cela est installé. L'installation peut se faire depuis le site officiel de Gradle: https://gradle.org/.

Pour LibGDX, il faut utiliser une version Java entre 11-18: https://libgdx.com/wiki/start/setup

La manière la plus simple pour ouvrir notre projet, consiste à l'ouvrir avec Intellij en ouvrant le projet avec le fichier build.gradle situé à la racine du projet.

Après l'installation des dépendances, il sera possible de build l'application et de démarrer le jeu depuis la méthode main de la classe DesktopLauncher dans desktop\src\com\decorator\game\DesktopLauncher.java.

Si l'application doit être lancée sur MacOS, il faut ajouter l'argument JVM **-XstartOnFirstThread.**

Nous avons rencontré certains problème lors du build à cause de la version Java. Depuis Intellij, il faut modifier la version dans les paramètre de Gradle, ici en jaune.