HAUTE ÉCOLE
D'INGÉNIERIE ET DE GESTION
DU CANTON DE VAUD
www.heig-vd.ch

MCR Modèle monteur

Groupe 7

Oussama Lagha
Le Guillou Benjamin
Guillaume Vetter
Luca Reis De Carvalho
Claude-André Alves

Cours	MCR
Nom du professeur	Pier Donini
Nom de l'assistant	Sébastien Rosat

Introduction

Le modèle monteur est un patron de conception qui permet de construire des objets complexes en utilisant une approche étape par étape. Ce modèle fait partie des patrons de création. Il permet de séparer la construction d'un objet de sa représentation. Il est très utile pour créer des représentations différentes à partir d'un même processus.

Implémentation

Pour représenter ce patron de conception nous avons créé un petit jeu. Dans ledit jeu nous pouvons créer un robot-combattant, mise en place du patron par la construction du robot. Ce robot sera ensuite opposé à un autre robot-combattant généré aléatoirement.
L'assemblage du robot doit respecter plusieurs choses :

- Il devra au minimum avoir un châssis, une paire de jambes, deux bras et une tête comme membres.
- Le châssis définit l'énergie que peut utiliser les robots si cette valeur est dépassée par les éléments du robot on ne peut le monter.

Un petit affichage graphique des combattant a été mise en place pour lequel nous avons produit de sprite.
L'animation des combats s'effectue en ligne de texte car nous avons considéré que ce n'était pas la partie importante de ce projet.
Nous avons décidé de faire le jeu avec l'outil libgdx compatible avec java. Le choix de libgdx est fait car il permet de gérer tous les aspects d'un jeu.

Monteur (Builder)

- [Objet – Créateur]
- Intention
    - Dissocier la construction d'un objet de sa représentation, afin de pouvoir créer des représentations différentes à l'aide d'un seul et même processus de construction.
- Motivation

    - Permet à un client de construire un objet complexe en spécifiant uniquement son type sans se soucier des détails de son implémentation.
    - Permet à un client de construire un objet complexe par étape. Très pratique pour créer par étape des objets qui devront par la suite être immuables (ex : StringBuilder).
- Cas d'utilisation
    - L'algorithme de création d'un objet complexe est indépendant des parties qui composent l'objet
    - Le système doit autoriser différentes représentations pour les objets en cours de construction
- Structure

- *Directeur* :
    - Un directeur peut être réutilisé.
    - Et il contient le ou les algorithmes permettant de créer un produit, étape par étape.
    - Utilisez l'interface fournie par Builder.
- *Constructeur* :
    - Un constructeur doit être suffisamment général (pour permettre à pratiquement tous les différents *ConcreteBuilder* de construire leur produit correspondant.)
- *ConcreteBuilder* :
    - Un constructeur construit toutes les pièces nécessaires et sait comment les assembler.
    - Et garde une trace de son produit. (Il contient une référence de son produit.)
    - Les clients obtiennent leur produit final d'un constructeur.
- *Produit* :
    - C'est l'objet complexe à construire. Pour chaque *ConcreteBuilder*, il existe un *produit* correspondant.
    - Et il fournit l'interface nécessaire au constructeur correspondant pour construire les pièces logiques et les assembler.
- Le client :
    - Choisissez un constructeur dont il a besoin.
    - Et choisissez un réalisateur dont il a besoin.
    - Ensuite, injectez le constructeur de béton dans le directeur.
    - Appelez la méthode build() du directeur.

- Conséquences

- Avantages

  Ce pattern permet une séparation claire entre l'ordonnancement de la création, et la construction des différentes parties de l'objet.
  Possibilité de créer plusieurs variantes d'un même produit.

- Désavantages

  L'inconvénient de ce design est qu'il devient rapidement très volumineux, et ajoute de la complexité pas toujours justifiée

Diagramme de classe

Voir annexe