Réalisation d'application

Rapport itération 2

Jeu d'aventure

Auteurs
Loïc Loko
Ghassen Ghouibi
Abdelrahim Bouchiha
Groupe 5

Sommaire

I Présentation

Dans le cadre du cours "Réalisation d'application", nous avons eu la tâche de réaliser l'itération 2 du jeu d'aventure intitulé "Zuul". Le but de cette itération est de découvrir certains facteurs qui influencent la conception d'une classe. Nous répondons ici à la question : Qu'est ce qui rend la structure d'une classe bonne ou mauvaise ? Ainsi dans cette itération, en répondant aux différentes questions, nous avons découvert des principes importants dans la conception d'une classe. Ce rapport contient donc les différentes réponses aux questions.

II Zuul-with-image, zuul-better

I getCommandList (Voir code)

II Zuul-better)

Nous n'avons pas trouvé l'archive de zuul-better par conséquent cette question n'a pas pu être traitée.

III StringBuilder(Voir code)

IV Recherche d'images appropriées au jeu (Voir fichiers sources)

V Titre du jeu

Le jeu est intitulé "One Piece Treasure Cruise"

VI Hashmap

VII Game, GameEngine, UserInterface (Voir code)

La classe Game classe est la classe principale de l'application "One Piece Treasure Cruise", elle fait appelle à la classe GameEngine qui crée toutes les pièces, crée le parser et démarre le jeu, GameEngine évalue et exécute également les commandes que le parser retourne. Elle use aussi de la classe UserInterface qui implémente une interface utilisateur graphique simple avec une zone de saisie de texte, une zone de sortie de texte et une image optionnelle.

VIII addActionListener, actionPerformed (Voir code)

AddActionListener ajoute l'écouteur d'action spécifié pour recevoir les événements d'action de ce champ de texte. Elle prend en paramètre l'écouteur d'action à ajouter.

ActionPerformed prend en paramètre une action, et fait appelle à la fonction processCommand afin de créer une interface pour le champ de texte de saisie, lire la commande et faire ce qui est nécessaire pour la traiter. Ici obligatoire, car Userinterface implémente l'interface ActionListener.

IX Ajouter des boutons (Voir code)

Nous avons ajouté les boutons des commandes eat, look, help et go.

X MVC (Voir code)

MVC est un design pattern très puissant qui nous permettra de bien structurer de grosses applications graphiques. Le Modèle-Vue-Contrôleur (MVC) est un pattern architectural qui à la charge de séparer les données (le modèle), l'interface homme-machine (la vue) et la logique de contrôle (le contrôleur). Il permet ainsi de générer des événements lors d'une modification du modèle et d'indiquer à la vue qu'il faut se mettre à jour. Ainsi dans notre projet, en utilisant ce pattern, nous prévenons les risques potentiels de changement dans notre logique de contrôle.

Ce design implique une séparation en 3 couches :

- Le modèle, il contient les données de l'application, il les reçoit et notifie la vue.
- La vue, elle représente l'interface graphique de l'utilisateur, sa tâche est juste d'afficher les données reçues par le modèle.
- Le contrôleur, il capte les actions de l'utilisateur, vérifie la cohérence des données et les transforme afin qu'ils soient compris par le modèle. Il peut aussi demander à la vue de changer. Il effectue la synchronisation entre le modèle et les vues.

XI Image de mot (Voir code)

XII Item (Voir code)

XIII Item Description (Voir code)

La classe item contient la description des items, chaque Room possède un item(variable), et chaque item s'initialise en même temps qu'une Room dans GameEngine.

L'affichage doit se faire dans GameEngine.

XIV Items (Voir code)

XV Intégration des objets (Voir code)

XVI back (Voir code)

XVII back test, back back (Voir code)

XVIII stack (Voir code)

XIX Javadoc (Voir documentation)