Rapport projet Zuul

HOUEE Adrien - E3T

Scénario

Le squelette d'un skateur légendaire répondant au nom de **Dem BONES** reprend vie et souhaite de nouveau "vivre" sa passion, **le skateboard**.
Pour cela, il devra dans un premier temps réunir toutes les pièces nécessaires pour remonter une planche de skateboard complète, c'est à dire :
- 1 plateau
- 1 grip
- 2 trucks
- 8 roulements à billes
- 4 roues

Réunir toutes les pièces du skateboard

Le plateau et le grip
Les 2 trucks
Les roulements à billes
Les roues

Rapport

Auteur

HOUEE Adrien - E3T

Thème

Dans une boutique de skateboard, Dem Bones doit monter sa nouvelle planche de skateboard.

Réponses aux exercices

Exercice 7.1

L’application ouvre une nouvelle fenêtre qui permet d’entrer les commandes go quit et help et de voir les retours du jeu.
La commande ‘go’ permet de se déplacer.
La commande ‘quit’ permet de quitter le jeu.
La commande ‘help’ affiche l’endroit où le joueur se trouve et ce qu’il peut faire.
On observe qu’il y a 5 pièces pour le moment.

Exercice 7.2

La classe CommandWord liste les mots de commande autorisés.
La classe Command permet d’agir sur les commandes entrées par l’utilisateur.
La classe Room permet de définir les pièces du jeu et la manière dont elle s’agence entre elles.
La classe Parser permet la gestion des entrées utilisateurs et en fait des commandes (Command).
La classe Game est le moteur de jeu. Elle permet de lancer une partie et de jouer.

Exercice 7.4

Dans cet exercice, on modifie la fonction createRooms de la classe Game afin d’y ajouter les pièces du jeu.

Exercice 7.5

1. L'exercice 7.4 est bien terminé.

4. Nous avons apporté cette modification afin d'optimiser le code et de ne pas le répéter. De plus, si nous devons effectuer un changement dans l'affichage des informations de la pièce (Room) courante, en changeant cette fonction seulement, la modification sera répercutée à tous les endroits nécessaires.

Exercice 7.6

1. Quoi qu'il arrive une valeur sera affectée à chaque attribut, donc que la valeur soit null en paramètre ou null par défaut, cela ne change pas grand chose.

Exercice 7.7

1. Les sorties d'une pièce (Room), sont relatives à la pièce, elles n'ont donc aucun intérêt à ne pas être dans la classe Room.

Exercice 7.8

4. La procédure setExits est devenue inutile, car elle n'est pas assez généraliste et limite les évolutions que l'on peut apporter au projet. Elle a été remplacée par setExit.

Exercice 7.8.1

Ajout des pièces manquantes, on peut désormais se déplacer à l'étage et au sous-sol.

Exercice 7.9

3. La fonction keySet nous retourne l'ensemble des clés d'une HashMap

Exercice 7.10

La méthodes getExitString renvoie une chaîne de caractères (String) contenant les sorties disponible d'une pièce (Room).

Exercice 7.11

Nous avons ajouté la méthode getLongDescription dans la classe Room et nous avons modifié la méthode printLocationInfo dans la classe Game.

Exercice 7.14

Nous avons ajouté la méthode look dans la classe Game, ainsi que le mot de commande look dans la classe CommandWord.

Exercice 7.15

Nous avons ajouté la méthode eat dans la classe Game, ainsi que le mot de commande eat dans la classe CommandWord.

Exercice 7.16

Nous avons implémenté les fonctions showAll et showCommands

Exercice 7.18

Nous avons modifié la méthode showAll de la classe CommandWord pour qu'elle devienne getCommandList et qu'elle renvoi la liste de toutes les commandes disponibles.
Dans la classe parser, nous appelons getCommandList dans la méthode showCommands.
Enfin, nous avons modifié la méthode printHelp afin qu'elle appelle la méthode showCommands.

Exercice 7.18.6

2) c. Nous n’avons plus besoin du scanner car nous allons passer par une interface graphique et donc récupérer la valeur des inputs.

Exercice 18.8

Ajout d’un bouton en position EAST qui permet l'exécution de la commande help grâce à un ActionListenner et un ActionPerformed.

Exercice 7.20

Création de la classe Item et extension de la classe Room afin qu’elle puisse contenir une classe Item.
En effet, les getters sont utiles pour récupérer les informations et les manipuler à l’extérieur de la classe Item.

Exercice 7.21

Les informations relatives à un Item dans la classe Room seront produites grâce à la classe Item.
Les différentes String permettant les descriptions d’un Item seront donc produites grâce à la classe Item.
La classe qui affichera ces informations sera la classe GameEngine car c’est dans cette classe qu’on met à jour l’affichage de l’interface graphique.

Exercice 7.22

Implémentation des HashMap dans la classe Room pour qu’elle puisse contenir plusieurs Item. La HashMap sera sous forme de tableau associatif Clé -> Valeur.

Exercice 7.23

Ajout d’un attribut PreviousRoom dans la classe Game.
Implémentation de la méthode back pour retourner dans la pièce précedente dans la classe Game et ajout du mot de commande back dans la classe CommandWord.

Exercice 7.26

Mise en place d’une Stack (Pile) pour stocker plusieurs Room.
La commande back fonctionne correctement car lorsque l’on exécute la commande, on procède à la vérification si la pile est vide ou non, donc si on peut retourner en arrière ou non.

Exercice 7.28.1

Implémentation de la commande test (classe GameEngine et CommandWord) ainsi que mise en place de la lecture de fichier.

Exercice 7.28.2

Création des différents fichiers de tests.

Exercice 7.29

Implémentation d'une nouvelle classe Player. Nous avons déplacé toutes les fonctions et attributs relatif à un joueur dans la classe Player.
Un Player devra avoir pour information unique sa pile de mouvements, son poids maximal, son inventaire et sa pièce courante.

Exercice 7.30

Implémentation des nouvelles commandes take et drop.
Chacune de ces méthodes prend un objet de la Room / Player et le dépose dans le Player / Room.

Exercice 7.31

Mise en place d'une HashMap afin que le player puisse avoir plusieurs Item.

Exercice 7.31.1

Création de la classe ItemList afin qu'elle s'occupe de toutes les interactions avec les listes d'Item. Modification des classes Player et Room afin d'utiliser ItemList.

Exercice 7.32

Mise en place de la gestion du poids des Item et du poids que le Player peut porter.
Vérification qu'un Player peut prendre un Item.

Exercice 7.33

Implémentation de la nouvelle commande inventory. Elle permet de consulter les Item que le Player a sur lui.

Exercice 7.34

Implémentation d'un Item consommable. Pour cela nous avons ajouté un attribut dans la classe Item précisant si l'Item est consommable ou non puis nous avons doublé la capacité de l'inventaire du joueur lorsque celui-ci le consomme.

Exercice 7.42

Ajout d'une condition dans laquelle le joueur perd la partie; Pour ma part, un nombre de mouvements (commande go et back).

Exercice 7.43

Mise en place d’un trapdoor dans le jeu. Le joueur pourra passer ce trapdoor dans un seul sens et ne pourra donc pas revenir en arrière. Pour cela nous avons créé une méthode isExit et une méthode clearRoomHistory.