Rapport Zuul

I)

A. *Auteurs*

Antoine AUBERT

B. *Thème*

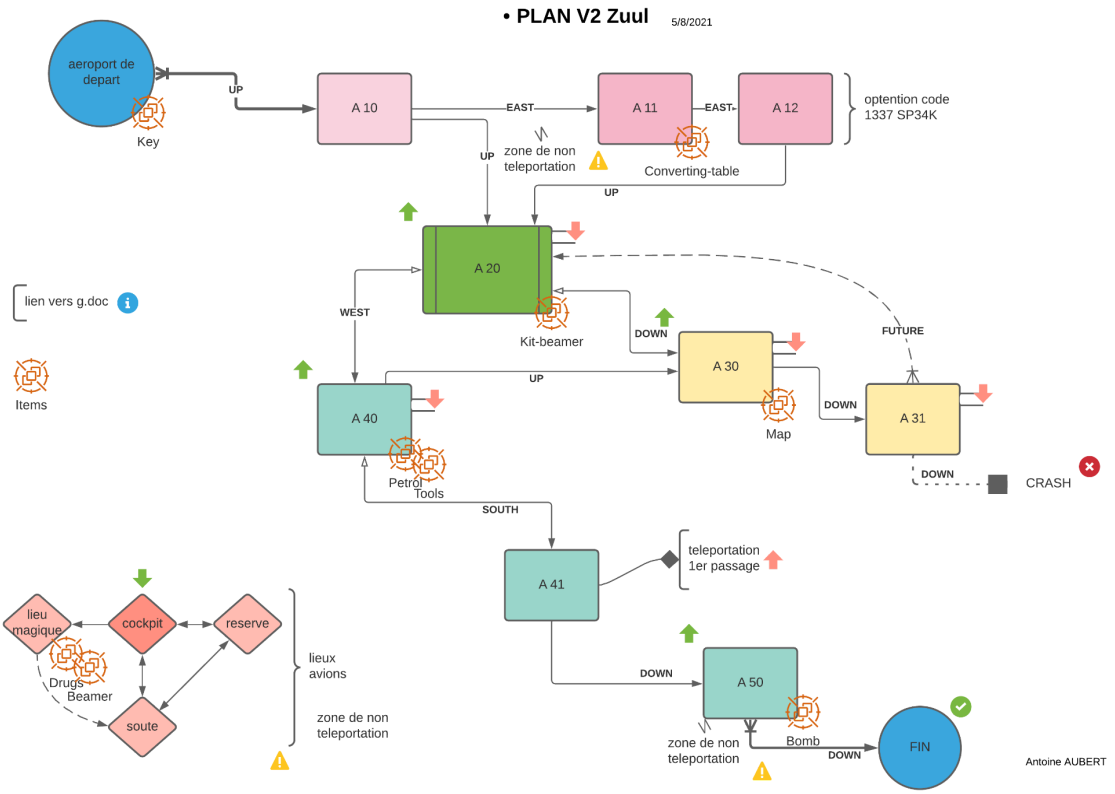
Simulateur d’avion pendant la ww2.

C. Résumé du scénario

Pilote d’avion militaire pendant la guerre, vous devez réaliser votre mission, sans périr. Il va falloir décoller de votre aéroport et réaliser votre mission, tout en passant par des checkpoints, et divers contretemps. Mais attention, nous sommes dans un contexte réaliste, et les paramètres réels s'appliquent, tel que le pétrole restant, d'éventuelles réparations, et transmission de message secret.

D. Plan

[plan v2]



E. Scénario détaillé

but:

Finir le jeu en validant les étapes ci-dessous...

organisation:

→ annonce du but et modalités du jeu via un message, donne les 1ers commandes à disposition.

1. but: codage décodage en A 10-12 (utilisation d'item requis: "traducteur"):
   → obtention du code 1337 SP34K→ A 11 prendre item livre et en A 12 traduire, donne traduction en A 20 pour continuer plus loin.
2. obtention de la carte du ciel en A30-31, puis teleportation (EX-LIST):
   →Allez en A30 prendre item carte puis en A31→ téléportation en a 20 puis allez en A 40
3. en A 40+Aavion etape pétrole (transport item "essence" (EX-LIST))
   →PRB essence, allez dans avion pour régler prb.

4. →arriver en A 41 et téléportation dans (A1-,A2-,A3-,A40)
5. → allez en A40 puis avions, régler prb calibrage puis re A 41

6. réalisation mission partie II en A50
   → En A50 faire dernière mission, aller dans avions, soute faire objectif, puis revenir en A 50
7. → FIN, message fin !!!

F. Détails des lieux, items, personnages

lieux:

lieu	n	S	action par lieu
Aéroport	1	up	début
AIR 10	2	up / east	
AIR 11	3	east	
AIR 12	4	up	
AIR 20	5	west down/ cockpit	
AIR 30	6	down/ cockpit	
AIR 31	7	futur / down	
AIR 40	8	south / cockpit / up	
AIR 41	9	up / down	téléportation
AIR 50	10	cockpit / down	
cockpit	11	R / S / LM	
réserve	12	C / S	
soute	13	C / R	
magic-place	14	S	
end	15	§ ou aéroport ...	fin

items:

nom item	ou	utilité
Converting-table	A 11	traduction du code en A 11-12
Map	A 30	
Key	Airport	
Petrol	A40	permet de gagner de l'autonomie
Tools	A40	
Bomb	A50	
Drugs	Magic-place	double la capacité de charge
Kit-beamer	A20	est nécessaire pour charger l'item (use Kit-beamer)
beamer	Magic-place	permet de se téléporter, une fois chargé dans la même pièce

personnages:

	pilote (protagoniste)	copilote	operateur radio	mécanicien	tour contrôle
rôle	Player	néant	néant	néant	néant

G. Situations gagnantes et perdantes

-réaliser la mission au complet
-perdre à cause de crash {si plus de down que de up... ou manque de pétrole}
- sinon si plus de pétrole

H. *Éventuellement des énigmes, mini-jeux, combats, etc.*

-passer en A12 si on fournit le bon code

I. *Commentaires*

-fait:

quoi ?		description
ordre des directions		création d'un tableau dans room avec tt les directions, cela permet d'afficher les sortie dans un ordre précis: NSEWUDCRSLmF. Pour cela création de 2 boucles for each imbriqués, on teste si pour chaque direction si la hashmap à la direction, si oui on rajoute a vExitStr la direction.
unknown dir & no door		Pour garder la distinction entre there is no door et unknown direction, je teste grâce à une boucle for each, si la direction tapée est présente dans le tableau des directions possibles. si ce n'est pas le cas,return this
création de use		GameEngine: création de use(Command) → permet d'interagir avec un Item. test si il y a un second mot test si la player possède l'item puis test si le second mot de la commande est dans la liste des items pouvant interagir. avec ajour du nécessaire dans GE et CW.
création d'un input test		Le but est de créer un dialogue pour aller en A12 depuis A11. L'exit de A11 "east" est est dans le corps d'un if() qui est faux au début lors de l'appel de creatRoom(). En A11 si on prend l'item "Converting-table" puis si on l'utilise un pop-up nous demande un code. String vOutPut = javax.swing.JOptionPane.showInputDialog("Enter the code"); import nécessaire comme pour aPlayerName. si l'input est valide ctd ";" actuellement on print le "code bon" puis on autorise le player à aller dans cette pièce via l'attribut aAllowToGo que l'on est à true afin de rendre de traitement conditionnelle valide. Pour définir un exil sans passer par creatRoom j'ai créé une nouvelle méthode openRoomExit(Room); elle test si this.aPlayer.getAllowToGo() puis sur le param Room on set l'exit. l'astuce est d'utiliser la tableau contenant toutes les Room et de lui passer le bon indice. il n'y a plus qu'à définir la nouvelle position d Pour rendre le test plus fun il faut avoir des personnage qui nous guide dans ce qu'il faut faire.
création d'un dossier		Dans .>test>résultats il y a les différents résultats du test complet à différents moment. Cela permet de comparer et avoir une trace du bon

résultats		fonctionnement de toutes les commandes.

II)

Réponses aux exercices

A. Exercice 7.5

création de print LocationInfo() qui permet d'éviter une duplication de code entre printWelcom et goRom , dans la class Game.
Direction présente: NSEW.
utilisé Current Room.a GetDescription → à remplir

B. Exercice 7.6

Modification de l'encapsulation pour réduire le couplage.
(modif non effectué sur les up et down → 7.8
modif class Room:
-public Room getExit(String direction)
-passage des attribut en prv
-return this; → permet dans Game de différencier pas de sortie et mauvaise direction. (cas de Unknown direction)
modif class Game:
-vNextRoom = aCurrentRoom.getExit(vDirection), teste 1 eul cas et pas 1 par direction

C. Exercice 7.7

Room:
-public String getExitString() test si il y a la direction et si oui, la

rajoute a la string vExitStr.
Game:
-dans private void printLocationInfo() apl de vExitStr (pour afficher les lieux dispo comme next room) via
this.aCurrentRoom.getExitString();

D. Exercice 7.8

Le but ici est de simplifier la classe Room et Game et de réduire le couplage, dus au partage de certains attributs publics. Pour cela nous utiliserons des hashmap.

Room:
Game:

- Changement de vNextRoom qui réaliser une disjonctions des cas, par
  vNextRoom = this.aCurrentRoom.getExit(vDirection); Cela permet de préciser uniquement les directions voulu et de ne plus se limiter avec des attributs pour chacune des directions.

4) La méthode changé est setExits, remplacée par setExit, car elle définit une seule sortie par appel. Cela est cependant plus pratique car nous n'avons pas à définir les sorties inexistantes.

5) Pour garder la distinction entre unknown direction et there is no door, j'ai du créer un tableau contenant les direction et je teste dans une nouvelle for each si la direction est connu, si oui on return this.aExits.get(pDirection);
sinon return this.

E. Exercice 7.9

room:

- Changement de getExitsString, avec une Hashmap qui stock les différentes sorties disponible pour la pièce en question. Création d'une boucle for adapter aux hashmap
  →**Comparaison HashMap / tableau**

F. Exercice 7.10

La méthode getExitString renvoie la description de chaques sorties de la pièce courante.
LA boucle for each est faite telle que pour chaque chaîne de caractères de la collection, la chaîne de caractères déclarant les sorties l'ajoute à sa propre chaîne.
Elle est retournée à la fin de la boucle.

Si la classe game a moins de commentaire java que la classe room, cela est dû au fait qu' il y a moins de méthodes / constructeurs en public.

G. Exercice 7.11

Création d'une méthode getLongDescription(), qui revoit une description détaillée par lieux. Permet de réduire le future couplage qui serait nécessaire pour + d'info.

Room:
- public String getLongDescription()

Game:
- Modif de private void printLocationInfo() pour afficher la longue description.

H. Exercice 7.12

d **option**

I. Exercice 7.13

d **option**

J. Exercice 7.14

Ajout de commandWords, permet de faire plus d'action de base.
CommandWords:
- ajout du command word.
- tableau avec une signature type:

private final String[]

Game:

- création d'une méthode look, qui retourne la description du lieu courant.

ProcessCommand:

- ajout de la méthode

K. Exercice 7.15

De même que 7.14

L. Exercice 7.16

Changement dans Command Word pour faire apparaître les commandword dispo dans la liste de printHelp() → Game. Afin de ne pas augmenter le couplage, nous créons un lien entre game et Parser dans printHelp() puis de Parser a Command. Cela évite encore une fois le couplage.

- Juste changements des attributs par rapport au pdf.

M. Exercice 7.17

d **option**

Version intermédiaire !

N. Exercice 7.18

Pour anticiper un éventuel couplage, nous modifions la méthode showall().

CommandWord:

- changement de nom de la méthode: showall → getCommandList
- création puis return d'une string qui prend les valeurs des différentes commandes possibles.

Parser:

- changement de nom: X → getCommands
- changement de la signature: perte du void car return this.aValidCommands.getCommandList();

game:

- creation d'un system.out.print(this.aParser.getCommands());

Cela permet d'afficher et de créer la string a 2 endroit différents, donc favorise une bonne encapsulation.

Exercice 7.18.1

Pas de modifications apportées.

Exercice 7.18.2

d

Exercice 7.18.3

→ images en bas de pages

Exercice 7.18.4

Rajout de System.out.println pour afficher le titre actuel:
-"WWII airplane simulator
modif de printWelcome

Exercice 7.18.5

création d'un attribut tableau dans GameEngine, il est initialisé et déclaré dans creatroom après avoir configuré les rooms.

Exercice 7.18.6

Room:

- ajout de l'Image et modif du constructeur → modif de GAME: creatrooms; précise les nom d' images associées aux lieux.

Parser:

- suppr de scanner car plus d'input clavier direct (interface graphique)
- remplacement de scanner par StringTokenizer qui découpe en morceaux la string.
- copier coller de zuul img → perso

Game:

- transfert presque total vers `GameEngine (création préalable)`
- `reste 2 attributs et un constructeur par défaut`

GameEngine:

- emigration du code de game
- avec changement:
- tous system.out.print → this.aGui.println pour interface graphique
- création de endGame
- creation interpretCommand (analyse les inputs)
- printWelcome rajout de "if" pour afficher une image si besoin
- creation public GameEngine()
- création public void setGUI
- perte de ma méthode play car interface graphique
- `processCommand a une pString et plus pCommand`
- `modif de printLocationInfo()`
- `attribution des noms des images et dans creat room via oracle !`

JFrame :: ajoute la prise en charge de l'architecture des composants

JTextField : permet l'édition d'une seule ligne de texte), *JTextArea* (affiche du texte brut

JTextArea : one d'affichage pour une courte chaîne de texte ou une image, ou les deux

JLabel :fournit une vue déroulante d'un composant léger

JPanel :conteneur léger générique

Exercice 7.18.7

d **option**

Exercice 7.18.8

UserInterface:

1/ import de tous les javax.swing nécessaire.

2/ déclaration d'un attribut par bouton de type JButton
forme aButtACTION

3/ creatGUI()

- création et initialisation de JPanel c'est-à- dire de plan graphique.
- vPanel.setLayout(new BorderLayout()); // ==> only five places
  cette ligne permet d'organiser les espaces du plan
- vPanel.add(vPanel1, BorderLayout.EAST);
  Cette ligne attribut a vPanel un vPanel dans le côté east.
  Un panel dispose de 5 positions disponibles, mais on peut mettre un panel dans un panel afin de le rediviser et de disposer de plus d'espace / sections / emplacement differents.

etapes creation buttons

1 déclaration avec txt

faire les panel pour les placer

leurs mettres des actions lisner pour real les actions dans actionPerformed

EXPLICATION FONCTIONNEMENT DE PANEL...

Panel est une sorte de plan, découpable en 5

on peut mettre des panel dans des panel pour rediviser et faire plusieurs emplacements

le borderlayout --> cole le bouton au côtes

CONSTRUCTION TYPE

```
//permet la création du bouton avec le txt
this.aButtTEST = new JButton("TEST");
//permet de placer des plan ou insérer les boutons
vPanel.add( this.aButtTEST,BorderLayout.EAST );
// add some event listeners to some components
this.aButtTEST.addActionListener( this );
```

4/ actionPerformed:
Relie un bouton a une action:
if(pE.getSource()==this.aButtBACK){this.aEngine.interpretCommand("back");}
Cela veut dire que si le bouton back est cliqué, on va interpréter cela comme une input clavier “back”

O. Exercice 7.19

d **option**

P. Exercice 7.19.1

d **option**

Q. Exercice 7.19.2

User Interface:
Pour déplacer les images, il faut déjà créer un nouveau répertoire avec toutes les images. Pour indiquer ensuite ou sont les images il faut donner le chemin d'accès et le stocker dans un estring. String vImagePath = "Images/edit/" + pImageName+".png"; le .pgn évite de taper l'extension ou de l’inscrire dans le nom de l'image.
URL vImageURL = this.getClass().getClassLoader().getResource(vImagePath);
cette ligne renseigne sur l’URL (chemin d'accès au images)

R. Exercice 7.20

Item:
Création d’une class Item et d'attribut privés tel que le nom, le poid ou le prix et une description. Création d’un constructeur naturel pour initialiser ces attributs. Créations d'accesseur.
Room:
création d’un attribut Item aItem et d’une hashmap
HashMap<String, Item> aItems; qui associe string et item

et un tableau déclaré static avec la liste des items → permet un affichage dans un ordre précis.
cretations

- d'une méthode setItem(.) qui def pour chaques pièces les items presets.
- getItemString() contient hashmap pour afficher les différents item dispo dans chaques room dans un ordre précis cf tab.
  modification de getLongDescription → retourne aussi les items présents.

S. Exercice 7.21

Pas de modifications nécessaires.
Création de la classe item pour faire un model des items, avec toutes leurs informations.
GameEngine affiche les information des Item en appelant la méthode get Longue Description de Room qui appelle est même la description de l'item.

T. Exercice 7.21.1

option

ajout d'un attribut aNom et de son getter. Création de getItemLongueDescription() dans Item. Elle renvoie le nom, la description et le poids.

U. Exercice 7.22

Room:

- initialisation d'une hashmap aItem dans le constructeur.
- Création d'une procédure addItem permettant d'ajouter un item.

V. Exercice 7.22.1

L'utilisation d'une hashmap est ici évidente car elle permet d'attribuer directement à une string un item, sans devoir comme dans un tableau, connaitre son indice.

W. Exercice 7.22.2

Room:
En utilisant addItem je rajoute un item dans la hashmap via la commande .put.
La position des items sont renseignée dans createRooms.
Ajout de l'item "key" en salle initiale et "petrol" / "tools" dans A-40

X. Exercice 7.23

GameEngine:

- création d'une procédure sans paramètre.
- un premier test pour savoir si il y a un second mot → retourne via un .aGui.println "il n'y a pas un 2eme mot"
- dans le cas où l'input est correcte on attribut a la current room la room précédente; c'est a dire: this.aCurrentRoom = this.aPreviousRoom;

Y. Exercice 7.24 **option**

La méthode fonctionne bien pour une seule et unique itération, et un message d'erreur apparaît bien si un second mot est saisi dans la commande d'entre.

Z. Exercice 7.25 **option**

Lorsque l'on teste deux fois ou plus la méthode back, nous ne changeons plus de pièce, cela est dû au fait que la previousRoom est la Current room.

AA. Exercice 7.26

Afin de pallier ce problème il faut créer une stack qui est une sorte de liste, qui s'agrandit au fur et à mesure et qui stock les rooms dans l'ordre de passage. Cela est une sorte d'historique de passage.

import nécessaire pour la stack:

- java.util.Stack

GameEngine:

- Creation d'un attribut: Stack<Room> a Previous Rooms
- initialisation dans le constructeur
- dans la méthode goRoom → on ajoute la nouvelle

aPreviousRoom dans la stack:
this.aPreviousRooms.push(this.aCurrentRoom);
le .push agrandie la tack et met en dernière position son paramètre, ici la currentRoom.

- dans la méthode back:

si il n y a pas de 2eme mot → this.aCurrentRoom = this.aPreviousRoom.pop();

C'est-à- dire que la currentRoom devient la dernière room de la liste. Le .pop() prend la dernière Room et la supprime afin de ne pas revenir au même point au prochain appel de back.

BB. Exercice 7.26.1

générer la javadoc via le terminal WIN (DOS). En se plaçant dans le répertoire courant on réalise 2 javadoc, une pour le “public / joueur” et une “private / programmeur" avec les informations des méthodes privées comprises.
Les commande utilisées sont:

```
cd C:\Program Files\BlueJ\jdk\bin
SET PATH="C:\Program Files\BlueJ\jdk\bin";%PATH%
PATH
javadoc
cd C:\Users\antoi\Desktop\perso_moi\travail\ipo\zuul\en cours\zuul_vXXX
javadoc -d userdoc -author -version *.java
javadoc -d progdoc -author -version -private -linksource *.java
```

(voir JAVA.BAT)

CC. Exercice 7.27 **option**

Cours sur réingénierie
Dans la version actuelle, une fonction de test permettrait de s’assurer que toutes les commandes, et réponses fonctionnent correctement, ainsi que s’assurer que les commandes qui ne doivent pas marcher ne marchent pas.

DD. Exercice 7.28 **option**

Pour s’assurer que tous les inputs soit valide sans tout faire et risquer d’en oublier il faut créer un méthode test qui d’elle même exécute les différentes méthodes → possibilité du joueur.
La méthode peut faire appel à un fichier texte type .txt qui recense ligne par ligne toutes les méthodes à tester. (un ou plusieurs en fonction des listes de commandes à tester).

-

EE. Exercice 7.28.1

CommandWords:

- rajout du command Word “test”

GameEngine:

- ajout dans interprète command d'un test qui s’assure que le mot “test” est bien un commandWords puis le l'exécute via this.test(vCommand);
- création de la méthode test qui prend un paramètre de type commande.
  1 on test si il y a un 2nd mot, si non → msg erreur
  si oui on crée un scanner qui a pour but de lire ligne à ligne.
  2 try: (on essaie sinon on passe a catch) On met dans une variable de type string le chemin d'accès relatif du fichier .txt String vTestPath = "test/" + pTest.getSecondWord() + ".txt";
  le / expriment un avant dans l'arborescence, le .txt permet de ne pas saisir le nom de l'extension. Le contenu du txt analysé par le scanner qui met ligne par ligne les instructions, et les stock dans ligne.
  this.interpretCommand(vLigne); Elles sont au final interprétées.
  3 catch: si le fichier n’est pas trouvé → msg d ‘err

FF. Exercice 7.28.2

Afin de faciliter la compréhension des tests nous créons plusieurs fichiers de test “complet”, “court”, “win”.
Le premier réalise un test de toutes les méthodes, pièces, items, et les potentiels bugs et faussent commandes puis quit.
avec le même principe court test juste qq cmd et win la solution la plus courte pour gagner.
"The the location file is \"zuul_vX.X\test\"

GG. Exercice 7.28.3 **option**

réalisation seulement d'un dossier avec tous les résultats "bons" afin de comparer avec les suivants de manière manuel.

HH. Exercice 7.29

Player:

- déclaration d'attribut privés aCurrentRoom / aPreviouseRoom (Stack)/ aNamePlayer
- import pour la stack
- initialisation des attribut dans le constructeur par defaut (this.aCurrentRoom est a null puis est initialiser plus tard)
- showInputDialog de la classe JOptionPane ont été nécessaire pour créer une boîte de dialogue qui récupère le nom du joueur dès le début du jeu.
- création de getter pour les 2 attributs puis 1 setter pour aCurrentRoom. Le setter permet de modifier si besoin le lieux de commencement.

GameEngine/ Player:

- La dépendance des attributs aCurrentRoom et aPreviousRoom à la classe GE ou Player est discutable mais ils concerne plus le joueur, et si il y a plusieurs joueurs en simultané il faut qu'ils puissent avoir différentes positions. → changement effectué grâce aux accesseurs.
- Modifications des méthodes Back et goRoom pour les mêmes raisons que ci-dessus. L'affichage est toujours dans GE mais ce qui est relatif à la position est traité par Player. → créations de méthodes qui fonctionnent en dualité: lastRoom et MovRoom.

GameEngine:

- modification de creatRoom pour initialiser le lieux courant via setCurrentRooom de Player.

II. Exercice 7.30

Pour créer take et drop j'ai du diviser les méthodes, c'est-à-dire répartir dans les classes player game engine et room plusieurs procédure qui effectue des tâches spécifiques. Mais avant de faire reconnaître les mots "take", "drop". L'intérêt de créer drop dans GE et dropItem dans Player est que l'affichage reste dans GE → pas de aGui dans Player. Cpdt tout ne peut être dans GE car les items dépendent du personnages. AddItem elle est dans Room car les items sont stockés dans une pièce.

CommandWords:

- ajout des "take", "drop" dans le tableau des commandes

GameEngine

```
else if ( vCommandWord.equals( "drop" ) )
    this.drop(vCommand);
```

- La commande “drop” équivaut à l’appel de la méthode drop.
- création de take → private et void.
  on test si il y a un second mots
  on test si il y a l’item dans la pièce ou est présent le player via hasItem
- si oui on appel takeItem de Player
- On met un message qui affirme le bon fonctionnement.

Player:

- création d’une hashmap:
  Private HashMap<String,Item> aMesItems; qui collecte tous les items portées par le joueur.
- création de takeItem qui dépend de la situation du joueur.
  public void takeItem(final String pNomItem)
  En public car GE a besoin d’y avoir accès. Ici le Paramètre désigne l’objet qu’il faut prendre.
- this.âMes Items.put(..) le .put met dans la hashmap.
- this.aCurrentRoom.removeItem(pNomItem);
  on retire de la current room l’objet de viens de prendre le player.

Room:

- création de removeItem(.)
  this.aItems.remove(pRemoveItem);
  supprime de la hashmap l’item associé à la clé
- création de hasItem(.)
  a pour paramètre le 2eme mot de la commande → l’item
  return this.aItems.containsKey(pItem); → retourne la key associé à l’item passé en paramètre.
- création de addItem(.) addItem ajoute l'item dans la pièce avec une hashmap
  this.aItems.put(pAddI.getItemNom(), pAddI); .put ajouté à la hashmap.

JJ. Exercice 7.31

déjà traitée avec le 7.30

KK. Exercice 7.31.1

création de la classe Item List.
imports pour hashmap et pour set.
ItemList:

- création de Inventaire HashMap<String, Item>. liste des Items présent que ce soit pour la room ou le player.
- transfert des addItem(*STRING*) → addItemList(*ITEM*)
- transfert des remove→ removeItemList
- transfert getItem
- getItemString return les item présent / possédé sous forme de string. utilise 2 boucles for each. 1 parcours le tableau static de tous les temps. 2 compare vItemKeys.
- création de hasItem pour savoir si l'item est présent. Utilisation de aInventaire.containsKey(pItem); containsKey() permet de savoir si la clef "String" est contenue dans la hashmap.

Player:

- adaptationde takeItem pour le parametre de addIteList (pNomItem) → this.aCurrentRoom.getItemList().getItem(pNomItem)

Game:

- dans addItem: pAddI.getItemNom() → pAddI

Dans cet exercice découverte des fonctions ctrl+B pour mettre un point d'arrêt et ainsi ouvrir le debugger qui permet de connaîtrez tout instant les états des différentes variables et l'ordre des méthodes appelées. Fonction découverte après recherche sur le net car dès j'avais des pbs de compilation / exécution. sources prb → initialisation des attributs mais non déclaration → NPE

LL. Exercice 7.32

Player:

- création de aWeight qui est le poids courant et de aWeightMax qui est le max. pas une valeur car peut être changé au cours du jeu.
- création de getter et setter.
- dropItem() → vérification que aItemList.getItem(pNomItem) != null

GameEngine:

- take: 1 test si le poids courant + poids obj est inf au max. 2 take et 3 on set le nouveau poid courant. sinon msg d'erreur
- drop: on calcule le poid sans l'obj drop. On set au niveau Poid etc.

MM. Exercice 7.33

CommandWord:

- ajout de "items" dans le tab

GE:

- ajout de l methode printUventaire() → donne les Item et le poids courant

NN. Exercice 7.34

ItemList:

- ajout de mon Item "Drugs" a static final String[] ITEMS.

CommandWords:

- ajout d'élément au tableau des commandes : "items"

GameEngine:

- déclaration de Item vDrugs
- vMagicalplace.addItem(vDrugs); → ajout a une Room
- ajout a interpreteCommand() du commandW vCommand pour que eat prenne un param
- modification de eat : param Command.
  On teste s' il y a un second mot puis si le joueur possède l'item dans son inventaire. Ici on oblige a ce que l'item soir "Drugs"
  ensuite on set le nouveau poid *2
  message de succès puis on remove l'item "Drugs" de ItemList via
  this.aPlayer.getItemList().removeItemList(pCommand.getSecondWord());

OO. Exercice 7.34.1

modification du fichier test complet et copie des résultats.
rajout de test et des nouvelles méthodes:

PP. Exercice 7.34.2

fichiers java doc fait avec le cmd et le JAVABAT.txt

QQ. Exercice 7.35 **option**

v

RR. Exercice 7.42

La forme de limite utilisée ici est un décompte du pétrole restant, il y a une diminution à chaque déplacement et la possibilité de gagner avec l'utilisation de l'item Petrol.

Item:

- création d'un setter pour le poids des Items.

Player:

- création d'un Item à Pétrole qui sera ma variable de pétrole courant.
  initialement dans le constructeur avec un valeur à défaut qui est la même que l'item pétrole existant en A40.
- création d'un getter du poid de l'item Petrol
- création du setter via le setter de Item

GameEnegine:

- création d'une méthode limitePetrol(.) avec un param de type commande. La création d'une méthode me semble plus approprié que modifier la valeur a la suite de go, back … pour plus de possibilités.
  1/ test quel méthode est responsable de l'appel de limitePetrol et définit la variation souhaitée.
  2/ on est la nouvelle qté et on voit si elle est positive.
  3/ en fonction appel de quit(.) avec un param Command fait avec 2 Strings.
  - appel de limitPetrol(.) dans go back et *use*

SS. Exercice 7.42.2

Je ne change pas d'IHM pour gagner du temps et implémenter des méthodes plus importantes.

TT. Exercice 7.43

Room:

- création de isExit(Room pRoomExit) qui renvoie true || false.
  en fonction de si la hashmap contenant les sorties contient

la Room passé en paramètre qui est la room que le l'on souhaite accéder.

GameEngine:

- dans back test si isExit() est true sinon affiche un message.

Trop de sens unique ?

uu. Exercice 7.44

Création d'un beamer un peu particulier. Se charge ssi on possède l'item Kit-beamer et Beamer sur soi. L'invocation du chargement se fait via la commande déjà existante qui détruit le Kit-Beamer et initialise la pièce de téléportation la ou use est appelé. Pour utiliser la fonction de téléportation il faut use le Beamer.

Beamer:

- Beamer est une sorte d'item avec des particularites en plus, d'ou le extends Item
- création de 2 attributs: aUsable ctd l'etat v ou f de la charge du beamer et aRoomCharge ctd la Room ou le beamer a été chargé et donc la ou il va nous ramener.
- initialisation dans le constructeur qui appel celui de Item via la commande super(...) puis this.a… = p…
- création de getter et setter des 2 nouveaux attributs pour connaître l'état des beamer.

Game Engine:

- on récupère le beamer via player item list get item → pas besoin de aBeamer
- installation de aBeamer dans le constructeur avec son état d'utilisation à faux et ça room a null car inconnue.

  1/ dans use() on fait le cas de la charge.
- Si le second mot de la commande est "Kit-beamer" eet que le player possède l'item "Beamer" (ici considéré comme iteml et non pas beamer) on charge la rom via le setter et on rend son son état utilisable a vrai. Puis l'on retire "Kit-beamer" de l'inventaire
- Message de succès ou d'échec.

  2/ dans use() le cas ou beamer est chargé
- test si il est bien utilisable
- this.aPlayer.movRoom(this.aBeamer.getRoomCharge()); pour changer de pièce.
- on redefinit son état d'utilisation à faux

vv. Exercice 7.45.1

Mise à jour du fichier de test et résultat dans le dossier prévu à cet effet.

1 seul probleme perte de l'item après avoir appelé creatRoom→ inclure dans

scénario

ww. Exercice 7.45.2

java doc Ok

xx. Exercice 7.46

TransporterRoom: {extends Room}

- création d'attribut RoomRandomizer aRoomRandomizer; permet d'appeler les méthodes de la class Room Randomizer
- création d'attribut boolean aBooleanAllow; qui détermine si la transporteurRoom est opérationnel (une unique teleportation pour la A41)
- constructeur qui appelle celui de room via super(..)
- this.aRoomRandomizer = new RoomRandomizer(pRooms); appel le constructeur de RoomRandomizer et passe un param de Room[]
- getter et setter de aBooleanAllow
- méthode getExit qui est redefinit par rapport à celle de Room → Override. Si la téléportation a déjà eu lieu on va en A50 grâce à determineRoom(.) qui choisit une certaine room choisie sinon on change le booléen et on appel findRandomRoom(.)

GameEngine:

- Déplacement de l'instruction VA41
- = new TransporterRoom après le tableau de Room.

RoomRandomizer:

- création d'attribut Room[] aTabRandomRoom;
- création d'attribut Random aRandom;

- Constructeur qui initialise l'attribut avec le tableau passé en paramètre et random est juste un new random. (le tableau est la même que dans GE)
- public Room findRandomRoom(final int pNbMax) choisis un random limùité par le param via nex.int(.) le . désigne la borne sup^positive qui doit être l'indice max du tableau auquel correspondent les room voulu pour une potentiel teleportation.
- Détermine Room permet de choisir une Room définit ici qui

dépend du param dans TranspR. (ici A50)

YY. Exercice 7.46.1

ZZ. Exercice 7.

v

II)

Mode d'emploi

néant

III)

Déclaration obligatoire anti-plagiat

néant