Auteur:

Baptiste HULIN

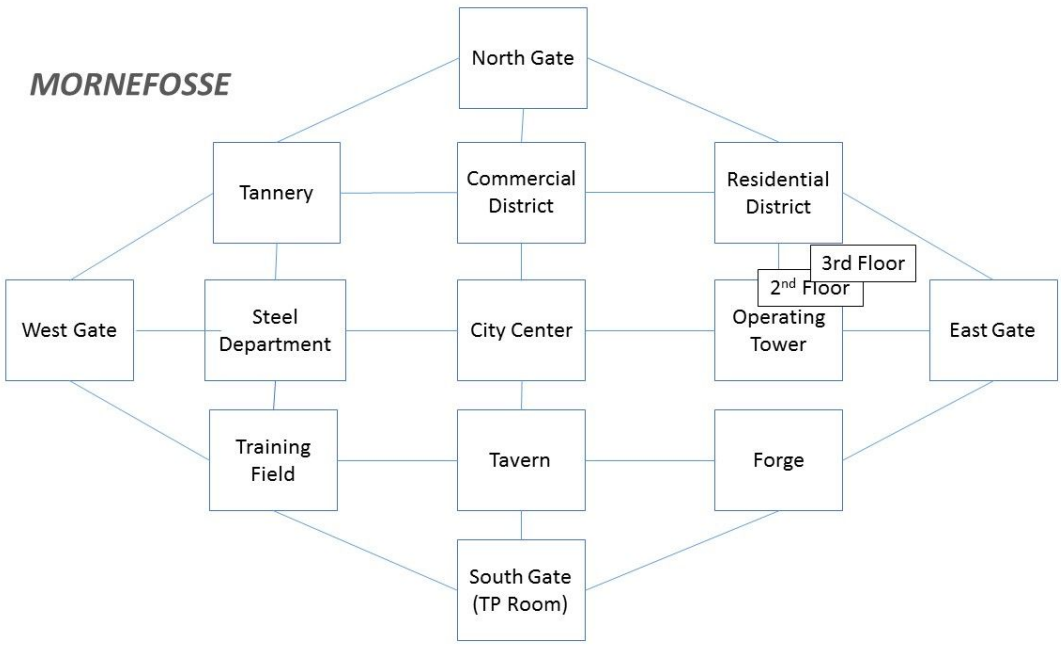
Phrase thème:

Le héros Nargael doit forger la plus puissante épée du monde.

Résumé du Scénario:

Vous incarnez Nargael, un jeune homme brave et vaillant et qui a acquis beaucoup d'expérience dans l'art guerrier depuis quelques années, où il n'était encore que diplômé de l'école nationale des épéistes (ENP). Une journée banale à MorneFosse débutait alors, et c'est alors qu'il discutait paisiblement avec son maître d'armes, Barnac, que ce dernier vous annonce qu'il détient une importante quête dont il veut vous faire part. Pour que la cité puisse se défendre contre de futures éventuelles attaques extérieures, Barnac vous demande alors d'entreprendre la création de l'épée la plus puissante du monde...

Plan de la carte du jeu



Scénario:

Vous devrez pour gagner le jeu; parler à Durothan qui vous demande de parler à Tensen pour obtenir du cuir.

Ce dernier vous donnera une attestation que vous devrez donner au Bourgeois pour enfin obtenir le cuir que vous pourrez échanger à Hina qui se trouve à la Tannerie contre le pommeau de votre future épée.

Retourner voir Durothan, qui en voyant que vous aurai récupéré le pommeau, vous donnera

la lame qu'il viendra de forger. Il vous enverra donc au quartier de l'Acier, pour constituer la fameuse épée.
Vous perdez si vous ne remplissez pas la quête avant la tombée de la nuit...

Réponse aux exercices:

EXERCICE 7.5 :
La procédure printLocationInfo() (code 7.2 donné dans le livre)
est créée et appellée dans les méthodes goRoom() et printWelcome() de la classe Game.
-Dans la méthode printWelcome(), il faut remplacer :

```
System.out.println("You are outside the main entrance of the university");
System.out.println("Exits: east south west");
System.out.println("");
```

par printLocationInfo().

-Dans la méthode goRoom(), il faut remplacer :

```
System.out.println ("You are "+this.aCurrentRoom.getDescription());
System.out.print ("Exits: ");
if (this.aCurrentRoom.aNorthExit != null)
System.out.print ("north ");
if (this.aCurrentRoom.aEastExit != null)
System.out.print ("east ");
if (this.aCurrentRoom.aSouthExit != null)
System.out.print ("south ");
if (this.aCurrentRoom.aWestExit != null)
System.out.print ("west");
System.out.println("");
```

par printLocationInfo().

EXERCICE 7.6 :
Dans la classe Room, il faut créer la fonction getExit donnée dans le livre (code 7.4).
-Dans la méthode goRoom() , remplacer :

```
Room vNextRoom = null;
if ( vDirection.equals("north") )
vNextRoom=aCurrentRoom.aNorthExit;
else if ( vDirection.equals("east") )          vNextRoom=aCurrentRoom.aEastExit;
else if ( vDirection.equals("south"))          vNextRoom=aCurrentRoom.aSouthExit;
```

```
else if ( vDirection.equals("west" ))              vNextRoom=aCurrentRoom.aWestExit;
```

par : Room vNextRoom = this.aCurrentRoom.getExit(vDirection);

EXERCICE 7.7 :

Dans la classe Room, créer la fonction getExitString() qui retournera les différentes sorties possibles en chaîne de caractères.

```
public String getExitString()
{
        String vNorthExit="";
        String vEastExit="";
        String vSouthExit="";
        String vWestExit="";
        if (this.aNorthExit != null)
                { vNorthExit = "Nord ";}
        if (this.aEastExit != null)
                { vEastExit = "Est ";}
        if (this.aSouthExit != null)
                { vSouthExit = "Sud ";}
        if (this.aWestExit != null)
        { vWestExit = "Ouest ";}

        return vNorthExit+vEastExit+vSouthExit+vWestExit;
}
```

Dans la classe Game, il faut remplacer, dans la méthode printLocationInfo() :

```
System.out.print ("Exits: ");
if (this.aCurrentRoom.aNorthExit != null)
        System.out.print ("north ");
if (this.aCurrentRoom.aEastExit != null)
        System.out.print ("east ");
if (this.aCurrentRoom.aSouthExit != null)
        System.out.print ("south ");
if (this.aCurrentRoom.aWestExit != null)
        System.out.print ("west");
```

par : System.out.print("Sorties : "+this.aCurrentRoom.getExitString());

EXERCICE 7.8 :

Dans la classe Room, il faut remplacer:

tous les attributs de type Room (les différentes sorties)

par : private HashMap<String, Room> exits;

Le constructeur naturel anciennement utilisé devient :

```
public Room(final String pDescription)
{
        this.aDescription = pDescription;
        this.aExits = new HashMap<String, Room>();
}
```

Créer une méthode setExit() à 4 paramètres Room :

```
{
        if(pNorthExit != null)
        this.aExits.put("nord", pNorthExit);
        if(pEastExit != null)
                this.aExits.put("est", pEastExit);
         if(pSouthExit != null)
                this.aExits.put("sud", pSouthExit);
        if(pWestExit != null)
                this.aExits.put("ouest", pWestExit);
}
```

Dans la méthode getExit, il faut remplacer tout le corps par :

```
{
        return aExits.get(pDirection);
}
```

Remplacer la méthode setExit par : public void setExit(final String pDirection, final Room pRoom)

```
{
        this.aExits.put(pDirection, pRoom);
}
```

Dans la classe Game, dans la méthode createRooms(), redéfinir les sorties des salles en appelant la méthode setExit comme suit : vRoom.setExits(« Direction », Sortie) ; La méthode setExits() est désormais inutile.

EXERCICE 7.8.1 :
Ajouter un déplacement verticale : vP1.setExit("bas", vP1bas); vP1bas.setExit("haut", vP1);

EXERCICE 7.9 :

Dans la méthode getExitString(), réécrire la méthode comme suit : public String getExitString() { String vString = "Exits : "; Set<String> vKeys = this.aExits.keySet(); for(String vExit : vKeys) {vString += " " + vExit;} return vString; } La méthode keySet() permet de lister toutes les clés (ici les différentes directions de sortie possibles) de l'objet HashMap que l'on a créer au préalable. Un appel de cet méthode retourne les clés de la Map.

EXERCICE 7.10 :

La méthode getExitString() permet de récupérer dans une variable vString l'ensemble des sorties en String ,les concaténer et retourner la chaîne de caratères. La méthode créer d'abord une variable String vString qui aura comme chaîne de caratcères « Exits : ». Elle créer ensuite une variable Set<String> vKeys qui contiendra la liste des clés de notre objet HashMap (this.aExits.keySet()). Cependant, cette liste se contente d'énumérer les sorties à la suite sans espace, c'est pourquoi la méthode utilise une boucle « for(String vExit : vKeys) » pour pouvoir rajouter un espace entre chaque String de sortie. Elle renvoie enfin la valeur de vString constituée de « Exits : »+l'ensemble des sorties séparées par un espace.

EXERCICE 7.11 :

Dans la classe Room, créer la méthode getLongDescription() donnée dans le livre. Appeler ensuite cette méthode depuis la classe game, dans la méthode printLocationInfo() : System.out.println(this.aCurrentRoom.getLongDescription()); Ce chagement permet une nouvelle fois de réduire le couplage entre la classe Room et la classe Game, et permettra plus tard de rajouter d'autres extensions comme les objets ou les monstres sans à

EXERCICE 7.14 Dans la classe CommandWords, ajouter dans le tableau sValidCommands la String « look » qui sera reconnue comme une commande valide. Dans la classe Game, définir la méthode look() : private void look() {
System.out.println(this.aCurrentRoom.getLongDescription()) ;
} puis dans la méthode processCommand, ajouter la condition : else if (vCommandWord.equals(« look »)) {look();}

EXERCICE 7.15

Dans la classe CommandWords, ajouter dans le tableau sValidCommands la String « eat » qui sera reconnue comme une commande valide. Dans la classe Game, définir la méthode eat() : private void eat() { System.out.println(« Vous venez de mangez et vous êtes repu. ») ; } puis dans la méthode processCommand, ajouter la condition : else if (vCommandWord.equals(« eat »)) {eat();} EXERCICE 7.16 Dans la classe CommandWords, ajouter la méthode showALL() donnée dans le livre. Cette méthode affiche, sous la forme d'une String, la liste des commandes valides du tableau séparées par un espace. Dans la

classe Parser, définir la méthode showCommands() donnée dans le livre, qui appelle la méthode showAll()de la classe CommandWords sur l'objet CommandWords aValidCommands. Enfin, dans la classe Game, dans la méthode printHelp(), remplacer : System.out.println(" go quit help"); System.out.println(""); par : this.aParser.showCommands(); Ces différentes modifications permettent d'éviter le couplage implicite qui nous obligeait à modifier la méthode printHelp() à chaque fois que l'on souhaitait créer un nouvelle commande de jeu.

EXERCICE 7.18

Dans la classe CommandWords, remplacer la méthode showALL() par getCommandList() comme suit : public String getCommandList() { String vString="" ; for(String vCommand : sValidCommands) {vString += " "+vCommand;} return vString;} Cette méthode retourne désormais une String qu'il faudra utilisé depuis la classe Game. Pour cela, la méthode doit d'abord être appelé par une méthode de la classe Parser : public String showCommands() { return this.aValidCommands.getCommandList(); } Puis dans la classe Game, appeler la méthode de la classe Parser en faisant un System.out dans la méthode printHelp() : System.out.println(this.aParser.showCommands());

EXERCICE 7.18.6

Zuul with images (UserInterface à compléter) Dans la classe Room, ajouter un attribut String aImageName, puis modifier le constructeur naturel de la classe comme suit : public Room(final String pDescription, final String pImage) { this.aDescription=pDescription; this.aExits = new HashMap<String, Room>();

this.aImageName = pImage; } et rajouter la fonction getImageName() qui permet de récupérer le nom de l'image sous la forme d'une String. Dans la classe Parser, changer l'import par import java.util.StringTokenizer; puis supprimer l'attribut aScanner. Il faut également modifier le constructeur puisqu'il n'y a plus d'attribut Scanner à initialiser. Ajouter un paramètre String pInputLine pour la méthode getCommand. Créer une classe GameEngine. Transférer dans la classe GameEngine l'ensemble du code de la classe Game, les modifications seront détaillés ci-dessous : Ajouter un attribut UserInterface aGui qui représente l'interface que le jeu utilisera. Ajouter une méthode setGUI qui permet d'initialiser l'interface : public void setGUI(final UserInterface pUserInterface) { this.aGui=pUserInterface; this.printWelcome; } Modifier la méthode printWelcome() en remplaçant tous les System.out.print par this.aGui.print : private void printWelcome()

{ this.aGui.print("/n"); this.aGui.println("Welcome to the World of Zuul !"); this.aGui.println("World of Zuul is a new, incredibly boring adventure game."); this.aGui.println("Type 'help' if you need help."); this.aGui.print("/n"); this.aGui.println(this.aCurrentRoom.getLongDescription()); this.aGui.showImage(this.aCurrentRoom.getImageName()); } Ceci va permettre d'afficher le texte non plus dans le terminal mais directement dans l'interface d'utilisateur. La dernière ligne permet d'afficher l'image de la salle de départ. Remplacer la méthode processCommand par une méthode interpretCommand qui ne prendra plus une commande

en paramètre mais une String : public void interpretCommand (final String pCommandLine) { gui.println(pCommandLine); Command vCommand=this.aParser.getCommand(pCommandLine); if (vCommand.isUnknown()) {this.aGui.println("I don't know what you mean ..."); return;} String vCommandWord = vCommand.getCommandWord(); if (vCommandWord.equals("quit")) if(vCommand.hasSecondWord()) this.aGui.println("Quit what ?"); else endGame(); else if (vCommandWord.equals("go")) goRoom(vCommand); else if (vCommandWord.equals("help")) printHelp(); else if (vCommandWord.equals("look")) look(); else if (vCommandWord.equals("eat")) eat(); } Cette méthode est maintenant une procédure et non plus un boolean.
Modifier la méthode printHelp() en remplaçant les System.out par des this.aGui. Modifier la méthode goRoom : public void goRoom(final Command pCommand) { if (! pCommand.hasSecondWord()) {this.aGui.println("Go where ?"); return;} String vDirection = pCommand.getSecondWord(); Room vNextRoom = this.aCurrentRoom.getExit(vDirection); if (vNextRoom == null) {this.aGui.println("There is no door !"); return;} this.aCurrentRoom=vNextRoom; this.aGui.println(this.aCurrentRoom.getLongDescription()); if(this.aCurrentRoom.getImageName() != null) this.aGui.showImage(this.aCurrentRoom.getImageName()); } Créer une méthode endGame() qui remplacera la méthode quit() : private void endGame() { this.aGui.println("Thank you for playing. Good Bye."); this.aGui.enable(false); }
Supprimer la méthode play() Dans la classe Game, il ne reste plus que : public class Game { private UserInterface aGui; private GameEngine aEngine; /** * Create the game and initialise its internal map. */ public Game() { this.aEngine = new GameEngine(); this.aGui = new UserInterface(this.aEngine); this.aEngine.setGUI(this.aGui); } }

EXERCICE 7.18.8

Faire l'import : import javax.swing.Jbutton ; Ajouter un attribut privée JButton aButton. Dans createGUI(), ajouter les lignes : this.aButton = new JButton("quit"); vPanel.add(this.aButton, BorderLayout.EAST); this.aButton.addActionListener(this); Ceci permet d'initialiser un bouton « quit » et de l'ajouter dans l'interface. Dans actionPerformed, ajouter une condition : if (pE.getActionCommand() == this.aButton.getActionCommand()) this.aEngine.interpretCommand("quit"); else processCommand(); qui permet d'exécuter la commande si l'on appuie sur le bouton.

EXERCICE 7.20-7.21

Créer une classe Item avec deux attributs privés : private String aItemDescription et private int aItemWeight. Créer un constructeur naturel à 2 paramètres pour initialiser ces deux

attributs, et deux geter, un pour chaque attribut. Dans la classe Room, ajouter un attribut Item aItem. Puis créer une procédure setItem(final String pItemDescription, final int pItemWeight) qui permet d'initialiser un Item dans une salle. Modifier la méthode getLongDescription() et ajoutant une condition if(this.aItem == null) et ajouter dans la chaîne de caractères la description de l'Item grâce au geter de la classe Item. Dans la classe GameEngine, ajouter un Item avec la méthode setItem() appelé sur une Room. La classe s'occupant de créer les Items n'est autre que la classe Item, et la String de l'Item est géré par la classe Room grâce à getLongDescription().

EXERCICE 7.22

Dans la classe Room, remplacer l'attribut Item par un attribut HashMap<String, Item> et l'initialiser dans le constructeur de la classe. Créer une méthode addItem() qui remplacera la méthode setItem() : public void addItem(final String pString, final Item pItem) { this.aItems.put(pString, pItem); } Ajouter une méthode getItemString() qui reprend le même modèle que getExitString() : public String getItemString() { String vString = "Objets disponibles : "; Set<String> vKeys = this.aItems.keySet();
for(String vItems : vKeys) {vString += " " + vItems;} return vString;} Et redéfinir getLongDescription() comme suit : public String getLongDescription() { if(this.aItems.isEmpty()) {return "Vous êtes "+this.aDescription+".\n"+this.getExitString() +".\n";} else return "Vous êtes "+this.aDescription+".\n"+this.getExitString() +".\n"+this.getItemString(); }
Dans GameEngine, dans createRoom(), appeler la méthode addItem() sur une variable Room pour pouvoir lui ajouter un Item. La HashMap permet d'y mettre plus d'un objet.

EXERCICE 7.23

Dans GameEngine, créer un attribut Room aPrevious Room qui permettra de sauvegarder la Room précédente dans la variable. Dans la méthode goRoom, écrire en première ligne this.aPreviousRooms = this.aCurrentRoom; qui permet donc de sauvegarder aCurrentRoom avant de changer de pièce. Créer une méthode back() : public void back() {
this.aCurrentRoom = this.aPreviousRoom;
this.aGui.println(this.aCurrentRoom.getLongDescription());
if(this.aCurrentRoom.getImageName() != null)
this.aGui.showImage(this.aCurrentRoom.getImageName()); } Cette méthode ne permet cependant de ne back qu'une seule fois. EXERCICE 7.26 Importer java.util.Stack ; Créer un attribut Stack<Room> aPreviousRooms ; et l'initialiser dans createRooms() : this.aPreviousRooms = new Stack<Room>(); Dans goRoom, ajouter la ligne this.aPreviousRooms.push(this.aCurrentRoom); avant this.aCurrentRoom=vNextRoom; La Stack est chargé avec aCurrentRoom juste avant le changement de salle. Enfin ajouter dans back this.aCurrentRoom = this.aPreviousRooms.pop(); pour retourner la dernière valeur de la Stack et la supprimer de la Stack.

EXERCICE 7.28.1

Dans la classe GameEngine, créer une méthode test : public void test(final Command pCommand) { if (pCommand.hasSecondWord() == false) {this.aGui.println("Test what ?"); return;} this.aTestMode = true; String vFichier = pCommand.getSecondWord(); try {Scanner vScanner=new Scanner(this.getClass().getClassLoader().getResourceAsStream("" +vFichier +".txt")); while(vScanner.hasNextLine()) {String vLigne = vScanner.nextLine(); this.interpretCommand(vLigne);} vScanner.close(); } catch(final Exception pException) {this.aGui.println("File not found");} this.aTestMode = false; this.aTransporterRoom.setSavedRoom(null); } Cette méthode utilise un objet Scanner pour lire les commandes du fichier test créer à la racine du projet, et les exécute. L'attribut aTestMode nous servira pour la commande alea plus tard. Si la méthode ne trouve pas le fichier, une exception surviendra, d'où l'intérêt du try / catch qui permettra d'afficher File not found si le fichier de test est introuvable.

EXERCICE 7.29

Cet exercice consiste à scinder notre classe GameEngine actuelle en deux classes : GameEngine et Player. La classe GameEngine s'occupera de l'exécution des différentes commandes du joueur alors que Player s'occupera de modifier les informations relatives au joueur, comme la salle courante, ou bien l'inventaire par exemple. Chaque classe sera alors responsable d'une tâche qui lui est propre. Dans GameEngine, les attributs aCurrentRoom et aPreviousRooms (de type Stack) sont transférés à la Classe Player. Les méthodes qui modifie un attribut de Player doivent appeler une méthode de Player afin de le modifier. La méthode goRoom() devient donc : public void goRoom(final Command pCommand) { if (! pCommand.hasSecondWord()) {this.aGui.println("Go where ?"); return;} String vDirection = pCommand.getSecondWord(); Room vNextRoom = this.aPlayer.getCurrentRoom().getExit(vDirection); if (vNextRoom == null) {this.aGui.println("There is no door !"); return;} this.aPlayer.move(vNextRoom); this.aGui.println(this.aPlayer.getCurrentRoom().getLongDescription()); if(this.aPlayer.getCurrentRoom().getImageName() != null) this.aGui.showImage(this.aPlayer.getCurrentRoom().getImageName()); } Dans Player, on a : public void move(final Room pRoom) { this.aPreviousRooms.push(this.aCurrentRoom); this.aCurrentRoom=pRoom; } Lorsque l'on créera une nouvelle méthode, il faudra à présent le faire en deux fois, dans Player, on modifiera les attributs concernés, alors que dans GameEngine, on se contentera d'appeler la méthode de Player et de générer l'affichage du message.

EXERCICE 7.30-31

Dans Player, ajouter un attribut de type HashMap<String, Item>. Dans Player et dans Room, créer pour chacune de ces classes deux méthodes qui permettront d'ajouter ou de retirer un Item de l'attribut HashMap. Dans GameEngine, créer une méthode take comme suit : public void take(final Command pCommand) { if (! pCommand.hasSecondWord()) {this.aGui.println("Take what ?"); return;} String vItemName = pCommand.getSecondWord(); Item vItem = this.aPlayer.getCurrentRoom().getRoomItem(vItemName); if (vItem == null) {this.aGui.println("There is no "+vItemName+" in this room");} else { this.aPlayer.addInventory(vItemName,vItem); this.aPlayer.getCurrentRoom().removeItem(vItemName); this.aGui.println("You've picked up "+vItemName); } } Méthode drop : public void drop(final Command pCommand) { if(! pCommand.hasSecondWord()) { this.aGui.println("Drop what ?"); return; } String vItemName = pCommand.getSecondWord(); Item vItem = this.aPlayer.getInventoryItem(vItemName); if (vItem == null) {this.aGui.println("You have no "+vItemName);} else {this.aPlayer.dropInventory(vItemName); this.aPlayer.getCurrentRoom().addRoomItem(vItemName, vItem); this.aGui.println("You've dropped "+vItemName);} } EXERCICE 7.31.1 Créer une classe ItemList qui a pour attribut une HashMap<String,Item>. Créer des méthodes dans cette classe afin d'ajouter ou retirer des éléments à la HashMap. Dans les classes Room et Player, remplacer l'attribut de type HashMap par un attribut de type ItemList. Pour ajouter ou retirer des objets, il faudra appeler les méthodes de la classe ItemList sur ce nouvel attribut de type ItemList.

EXERCICE 7.32

Dans la classe Player, ajouter un attribut de type int aMaxWeight. Ajouter une méthode getCurrentWeight() qui retourne le poids que le joueur porte actuellement. public int getCurrentWeight() { int vWeight = 0; for (Item vItem : this.aInventory.getItemList().values()) { vWeight += vItem.getItemWeight(); } return vWeight; } Dans la méthode take() de GameEngine, ajouter une condition comme suit : public void take(final Command pCommand) { if (! pCommand.hasSecondWord()) {this.aGui.println("Take what ?"); return;} String vItemName = pCommand.getSecondWord(); Item vItem = this.aPlayer.getCurrentRoom().getRoomItem(vItemName); if (vItem == null) {this.aGui.println("There is no "+vItemName+" in this room");} else if(this.aPlayer.getCurrentWeight() + vItem.getItemWeight() > this.aPlayer.getMaxWeight()) {this.aGui.println("You have no more space in your bag !"+"\n"+"Trouvez une barre

énergétique pour de transporter plus d'objets !");} else { this.aPlayer.addInventory(vItemName,vItem); this.aPlayer.getCurrentRoom().removeItem(vItemName); this.aGui.println("You've picked up "+vItemName); } } EXERCICE 7.33 Dans Player, créer une méthode getInventoryString() : public String getInventoryString() { return this.aInventory.getItemString(); } getItemString() est une méthode d'ItemList qui retourne l'ensemble de clés d'une HashMap <String, Item> séparé d'un espace. Dans GameEngine, créer un méthode inventaire() public void inventaire() { this.aGui.println("Inventaire : "+this.aPlayer.getInventoryString()); this.aGui.println("Current Weight : "+this.aPlayer.getCurrentWeight()); this.aGui.println("Max Weight : "+this.aPlayer.getMaxWeight()); } EXERCICE 7.34 public void eat(final Command pCommand) { if(! pCommand.hasSecondWord()) { this.aGui.println("eat what ?"); return; } if(pCommand.getSecondWord().equals("Bar énergétique")) {if(this.aPlayer.hasItem("Bar énergétique")) { this.aPlayer.dropInventory("Bar énergétique"); this.aPlayer.setMaxWeight(20); this.aGui.println("Vous venez de manger une barre énergétique."+"\n"+"Vous pouvez à présent porter plus d'objets !"); } else this.aGui.println("Vous n'avez pas de bar énergétique dans votre inventaire");} else this.aGui.println(("Vous ne pouvez pas manger ça !")); } Cette méthode se contente de retirer l'item de l'inventaire du joueur et augmente le poids maximum du joueur.

EXERCICE 7.33

Dans la classe GameEngine, il faut créer une méthode LookInventory qui retourne la liste des items du Player :

```
private void lookInventory()
    {
    aGUI.println(this.aPlayer.getInventoryList());
    }
```

et ajouter dans interpretcommand la commande "items" :

```
[...]
else if (commandWord.equals(CommandWord.ITEMS))
    lookInventory();
```

EXERCICE 7.34

Dans la classe GameEngine, il faut modifier la commande eat() qui n'affichait auparavant qu'un message et remplacer :

```
 aGUI.println("You've just eaten the local Fosse'Burger, you're not hungry anymore");
```

par :

```
if (!pCommand.hasSecondWord())
 {
        aGUI.println ("Eat what ?");
        return;
  }
String vName = pCommand.getSecondWord();
if(!aPlayer.getInventory().hasItem(vName))
{
        aGUI.println("You haven't any " + vName + ", you can't eat it !");
        return;
}
Item vItem = this.aPlayer.getInventory().getItem(vName);

if (!vItem.isEatable())
{
        aGUI.println("You can't eat " + vName);
        return;
}
aPlayer.getInventory().itemTaken(vName);
aGUI.println("You have eaten " + vName);
if (vName.equals("Magic_Cookie"))
{
        aGUI.println("You can now carry more items !");
}
```

EXERCICE 7.34.1

Le fichier test est modifié en conséquence des précédents changements et teste maintenant le jeu de manière exhaustive.

EXERCICE 7.35

A l'aide du fichier jar *zuul-with-enums-v1* fourni, on créé une classe enum CommandWord :

```
public enum CommandWord
{
        GO, QUIT, HELP, UNKNOWN, LOOK, EAT, BACK, TEST, TAKE, DROP, ITEMS;
```

```
}
```

et modifier interpretCommand() dans la classe GameEngine

```
[...]
CommandWord vCommandWord = command.getCommandWord();

if (vCommandWord.equals(CommandWord.HELP))
        {
        aGUI.println ("You are alone, try to go to another place");

        aGUI.println("Your command words are:");

        aGUI.println (aParser.showCommands());
        }

   else if (vCommandWord.equals(CommandWord.LOOK))
        look();

   else if (vCommandWord.equals(CommandWord.EAT))
        eat(vCommand);

   else if (vCommandWord.equals(CommandWord.GO))
        goRoom(vCommand);

   else if (vCommandWord.equals(CommandWord.BACK))
        back();

   else if (vCommandWord.equals(CommandWord.TEST))
        test(vCommand);

   else if (vCommandWord.equals(CommandWord.DROP))
        drop(vCommand);

   else if (vCommandWord.equals(CommandWord.TAKE))
        take(vCommand);

   else if (vCommandWord.equals(CommandWord.ITEMS))
        lookInventory();

   else if (vCommandWord.equals(CommandWord.QUIT))
{
```

```
        if(vCommand.hasSecondWord())
        aGUI.println("Quit what?");
        else
                endGame();
}
```

EXERCICE 7.35.1

La méthode interpretCommand() de la classe GameEngine est donc devenue bien longue. Pour y remédier, nous faisons l'usage d'un switch et remplacons la série de else if par:

```
switch (vCommandWord)
{
        case HELP:
        aGUI.println ("You are alone, try to go to another place");

        aGUI.println("Your command words are:");

        aGUI.println (aParser.showCommands());
        break;

        case GO:
        goRoom(vCommand);
        break;

        case QUIT:
        if(vCommand.hasSecondWord())
        aGUI.println("Quit what?");
        else
        endGame();
        break;

        case LOOK:
        look();
        break;

        case EAT:
        eat(vCommand);
        break;

        case BACK:
        back();
```

```
        break;

        case TEST:
        test(vCommand);
        break;

        case TAKE:
        take(vCommand);
        break;

        case DROP:
        drop(vCommand);
        break;

        case ITEMS:
        lookInventory();
        break;

        default:

        aGUI.println("I don't know what you mean...");
        break;
        }
```

EXERCICE 7.41.1

A l'aide du fichier jar *zuul-with-enum-v2*, on modifie la classe enum CommandWord et on remplace:

```
public enum CommandWord
{
        GO, QUIT, HELP, UNKNOWN, LOOK, EAT, BACK, TEST, TAKE, DROP, ITEMS;
}
```

par:

```
public enum CommandWord
{
        GO("go"), QUIT("quit"), HELP("help"), UNKNOWN("?"), LOOK("look"), EAT("eat"),
BACK("back"), TEST("test"), TAKE("take"), DROP("drop"), ITEMS("items");
        private String aCommandString;
```

```
    CommandWord(final String pCommandString)
    {
        this.aCommandString = pCommandString;
    }

    public String toString()
    {
        return aCommandString;
    }
}
```

EXERCICE 7.42

Dans la classe GameEngine, on ajoute une limite de temps (ici, de déplacement), soit un attribut private int aLimit que l'on initialise à 60 dans le constructeur.

EXERCICE 7.42.2

Après avoir cherché les différents Layouts que propose Java j'ai décidé de crééer d'autres boutons, de les colorer et de les placer selon ce code :

EXERCICE 7.43

EXERCICE 7.44

Création d'une nouvelle méthode Beamer qui hérite de la classe Item puisque l'on peut dire que c'est "une sorte d'objet".

```
public class Beamer extends Item
    {
    private Room aSavedRoom;
    private boolean aCharged;

    public Beamer(final String pName,  final int pWeight, final String pDescription)
    {
    super(pName, pWeight, pDescription);
    }

    public Room getSavedRoom()
    {
    return aSavedRoom;
    }
```

```
public void setSavedRoom(final Room pSavedRoom)
{
this.aSavedRoom = pSavedRoom;
}

public boolean isCharged()
{
return this.aCharged;
}

public void setCharged(final boolean pCharged)
{
this.aCharged = pCharged;
}
```

EXERCICE 7.45.1

Mise à jour des fichiers test avec le contenu des dernières modifications.

EXERCICE 7.46

Création d'une classe RoomRandomizer afin de pouvoir créer une TransporterRoom.

```
public class RoomRandomizer
{
        private static Long seed = null;
        private Room[] aTabRoom;
        private Random aRandom;

        public RoomRandomizer (final HashMap <String, Room> pListRoom)
        {
        if (seed == null)
        this.aRandom = new Random();
        else
        this.aRandom = new Random(seed);
        this.aTabRoom = new Room [pListRoom.size()];
        int vI = 0;

        for (String vS : pListRoom.keySet())
        {
        this.aTabRoom[vI] = pListRoom.get(vS);
```

```
        vI += 1;
        }
        }

        public Room nextRoom()
        {
        if (seed != null)
        this.aRandom = new Random(seed);
        return this.aTabRoom[aRandom.nextInt(this.aTabRoom.length)];
        }

        public static void setSeed(final Long pSeed)
        {
        RoomRandomizer.seed = pSeed;
        }
}
```

EXERCICE 7.46.1

Ajout d'une commande aléa(ici, "random") disponible uniquement en mode test

```
 public void random(final Command pCommand)
        {
        if(!pCommand.hasSecondWord())
        {
        aGUI.println("Random enabled.");
        RoomRandomizer.setSeed(null);
        }
        else
        {
                try
                {
                        RoomRandomizer.setSeed(
Long.parseLong(pCommand.getSecondWord(), 10));
                }
                catch (NumberFormatException E)
                {
                        aGUI.println("Wrong seed.");
                        return;
                }
        aGUI.println("You changed the seed.");
        }
```

}

Et ajout de la commande dans la méthode interpretCommand de GameEngine et dans la classe CommandWord.

EXERCICE 7.47

EXERCICE 7.47.1

Ajout des paquetages :

Les classes Item, ItemList et Beamer dans pkg_items
Les classes GameEngine, UserInterface et Player dans pkg_engine
Les classes Room, TransporterRoom et RoomRandomizer dans pkg_rooms
Toutes les classes relatives aux commandes, Parser, Command, CommandWord et CommandWords dans pkg_commands.

EXERCICE 7.48

Ajout de la classe Character :

```
public class NPC
{
	private Room aCurrentRoom;
	private ItemList aNPCInventory;
	private String aText;
	private String aNPCName;
	private HashMap<String, NPC> aNPCList ;

	public NPC (final String pNPCName)
	{
		this.aNPCName = pNPCName;
		this.aNPCInventory = new ItemList();
		this.aNPCList = new HashMap<String, NPC>();
	}

	public void setCurrentRoom(final Room pCurrentRoom)
	{
		this.aCurrentRoom = pCurrentRoom;
```

```
	}


	public Room getCurrentRoom()
	{
		return this.aCurrentRoom;
	}
	public void ajoutItemNPC(final String pNomItem, final Item pItem)
		{
			this.aNPCInventory.addItem(pNomItem, pItem);
		}

	public ItemList getAllItemsPNJ()
	{
		return this.aNPCInventory;
	}

	public String getText()
	{
	return this.aText;
	}

	public String getNPCName()
	{
	return this.aNPCName;
	}

	public void setText(final String pText){
	aText = pText;
	}

	public void addNPC(final String pName, final NPC pNPC)
	{
	this.aNPCList.put(pName, pNPC);
	}
}
```

EXERCICE 7.49

Ajout d'une classe NPCList et MovingNPC

MovingNPC:

```
public class MovingNPC extends NPC
{
	public MovingNPC (final String pName)
	{
	super(pName);
	}

	public void moveNPC()
	{
	HashMap<String, Room> vExits = this.getCurrentRoom().getExits();
	int vI = 0;
	int vO = 0;
	Room vOut = null;
	vO = RoomRandomizer.generateRandomNumber(0,RoomRandomizer.countRooms(vExits));
	Set<String> keys = vExits.keySet();
	for(String key : keys)
	{
	if(vI == vO)
	vOut = vExits.get(key);
	vI++;
	}
	if(vOut == null)
	{
	return;
	}
	super.getCurrentRoom().getNPCList().removeNPC(this);
	super.setCurrentRoom(vOut);
	super.getCurrentRoom().addNPCRoom(this.getNPCName(), this);
	}
}
```

NPCList:

```
public class NPCList
 {
	private HashMap<String, NPC> aNPCList ;

	public NPCList()
	{
	this.aNPCList = new HashMap<String, NPC>();
```

```
}

public void addNPC(final String pName, final NPC pNPC)
{
this.aNPCList.put(pName, pNPC);
}

public boolean hasNPC(final String pI)
{
return aNPCList.containsKey(pI);
}

public NPC getNPC(final String pI)
{
return aNPCList.get(pI);
}

/**
* Retourne les NPC disponibles
*/public String getNPCString()
{
if (!aNPCList.isEmpty())
{
String vNPCString = "";
Set<String> vKeys = aNPCList.keySet();

for(String vS : vKeys)
{
vNPCString+= "There is NPC " + this.aNPCList.get(vS).getNPCName() + " here" + "\n";
}

return vNPCString;
}

else
{
return "";
}
}

public HashMap<String, NPC> getNPCList()
{
```

```
        return this.aNPCList;
        }

        public boolean isEmpty()
        {
        return aNPCList.isEmpty();
        }

        public void removeNPC(final NPC pNPC)
        {
        this.aNPCList.remove(pNPC.getNPCName());
        }
}
```

EXERCICE 7.53

Ajout de la méthode main dans Game :

```
public static void main(final String[] pArgs)
{
        new Game();
}
```

EXERCICE 7.54

Le jeu s'exécute sans l'aide de BlueJ.