Rapport

ENIGMA MUSEUM

(A3P(AL) 2022/2023 G8)

Table des matières

I. Présentation 3

A) Auteur 3

B) Thème 3

D) Plan 3

E) Scénario détaillé 4

F) Détail des lieux, items, personnages 4

G) Situations gagnantes et perdantes 5

H) Énigmes 5

II. Exercices 5

Ex 7.2.1 5

Ex 7.5 5

Ex 7.6 6

Ex : 7.7 7

Ex : 7.8 8

Ex : 7.9 9

Ex : 7.10 10

Ex : 7.11 10

Ex : 7.14 10

Ex : 7.15 11

Ex :7.16 12

Ex :7.18 13

Ex : 7.18.6 13

Ex: 7.18.8 ..... **14**

Ex: 7.19.2 ..... **14**

Ex: 7.20 ..... **14**

Ex: 7.21 ..... **16**

Ex:7.22 ..... **17**

Ex: 7.23 ..... **19**

Ex: 7.26 ..... **20**

Ex :7.28 ..... **21**

Ex: 7.28.2 ..... **21**

Ex: 7.29 ..... **21**

Ex: 7.31 ..... **22**

Ex: 7.31.1 ..... **23**

Ex: 7.32 ..... **26**

Ex: 7.33 ..... **27**

Ex: 7.34 ..... **28**

Ex: 7.42 ..... **29**

Ex: 7.42.2 ..... **30**

Ex: 7.43 ..... **30**

Ex:7.44 ..... **31**

III. Mode d'emploi ***33***

IV. Déclaration anti-plagiat ***33***

I. Présentation

A) Auteur

Tsundyk Anastasia

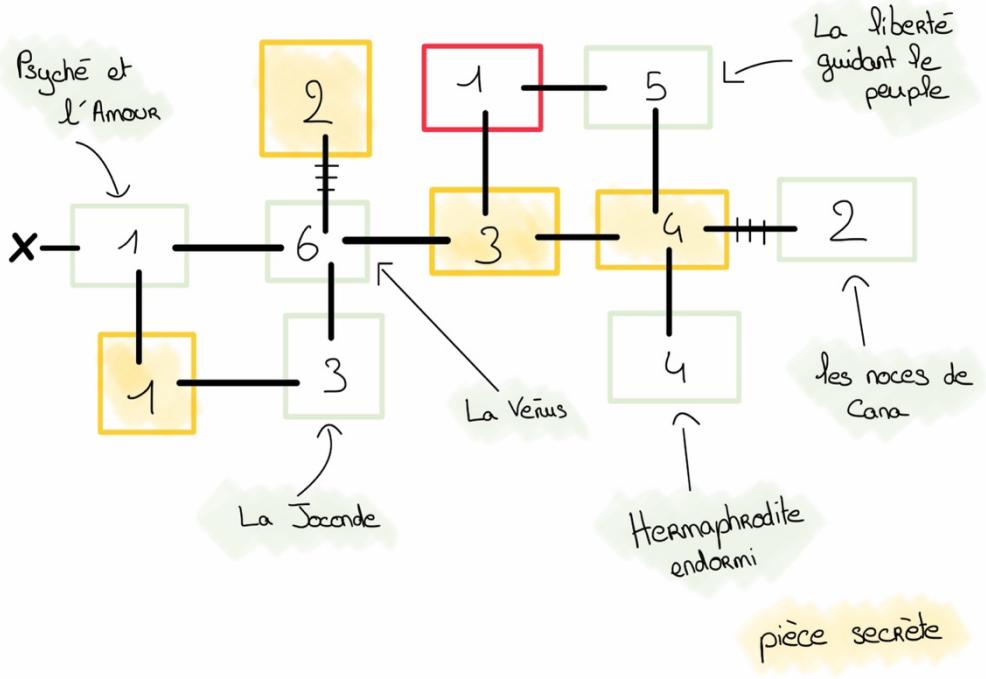
B) Thème

Dans le musée du Louvre, un jeune étudiant va devoir récupérer des indices, pour pouvoir répondre à une énigme finale qui permettra de désamorcer une bombe dans un temps donnée.

C) Résumé du scénario

Un étudiant du nom de Thomas Selly doit arrêter la mise en route d'une bombe au musée du Louvre. Il va devoir parcourir différentes pièces du musée dans lesquelles il y aura des énigmes ou des objets à trouver. Si le joueur réussit à répondre à l'énigme ou réussit à récupérer l'objet en question il lui sera donné un indice.
Le joueur devra récolter les indices qui lui seront donné et répondre à l'énigme final avant la limite de temps.

D) Plan



E) Scénario détaillé

Le personnage principal, est ici un étudiant (qui veut entrer dans le DGSE) du nom de Thomas Selly qui doit arrêter la mise en route d'une bombe au musée du Louvre. Il va devoir parcourir différentes pièces du musée dans lesquelles il y aura des énigmes ou des objets à trouver. Si le joueur réussit à répondre à l'énigme il lui sera donné un indice.
Il y aura en tous une dizaine de pièces à visiter et dans chacune, l'énigme sera en lien avec une œuvre du musée.
Le joueur devra récolter les indices qui lui seront donné, et devra trouver la pièce finale et répondre à l'énigme final avant la limite de temps, mais pour pouvoir résoudre l'énigme il faut d'abord pouvoir entrer dans la pièce finale.
Il y aura des pièces spéciales, elles peuvent enlever du temps ou en rajouter.
Ces pièces porteront le nom de pièces secrètes, le joueur aura le choix de prendre ou non l'objet.

F) Détail des lieux, items, personnages

Nous avons 6 pièces en rapport avec une œuvre du musée :

-Psyché et l'amour
-La Vénus
-La Joconde
-Hermaphrodite endormi
-La liberté guidant le peuple
-Les noces de Cana

Puis il y a 4 pièces dite « pièce secrète » et pour finir il y a la pièce finale ou il faudra répondre à l'énigme.

G) Situations gagnantes et perdantes

Le joueur gagne s'il réussit à répondre dans le temps donné à l'énigme final :
« **On doit généralement faire attention à mon tout.** ». A l'aide des indices qu'il aura récolté.
Il aura donc réussi à désamorcer la bombe et à sauver tout le monde.
Sinon le joueur aura perdu et la bombe explosera.

H) Énigmes

Voici les énigmes avec leurs (réponses) et les indices pour chacune des pièces vertes :

1 : Lequel des deux personnages est Eros ? (Celui avec les ailes)
Mon premier et la première l'être de l'alphabet.

2 : Combien de personnages sont sur le tableau ? (132)
Mon deuxième est utilisé par les prisonniers pour s'échapper dans les bandes-dessinées.

3 : Les sourcils de la Joconde ont été effacé Vraie ou Faux ? (Vraie)
Mon troisième est un verbe passif qu'on dit régulièrement pour décrire les vieux châteaux.

4 : Cette pièce permet de trouver Hermaphrodite et donne l'indice suivant.
Mon quatrième est la meilleure carte d'un jeu normal de 52.

5 : Quand a été faite cette peinture ? (1830)
Mon cinquième est un pronom personnel.

6 : Cette pièce permet de trouver un indice.
Mon sixième est un pronom personnel indéfini.

Réponse : A-Lime-Hante-As-On (Alimentation).

II. Exercices

Ex 7.2.1

Scanner :
La classe Scanner est une classe du JDK cad sa Javadoc est fournie par les concepteurs de Java. Elle peut servir à lire les mots ou lignes tapés au clavier.

Ex 7.5

Pour éviter la répétition du code qui affiche la sortie de la salle dans les fonctions printWelcome() et goRoom(), on a créé une méthode nommée printLocationInfo() dans la classe Game qui donne accès à l'information sur la sortie de la salle actuelle à chaque appel.

```java
/**
 * on crée une méthode
printLocationInfo() dans la classe Game qui permet à chaque appel
de connaître la sortie de la salle actuelle
 *
 */
public void printLocationInfo(){
    System.out.println("Vous êtes actuellemnt "+this.aCurrentRoom.getDescription());
    System.out.print("Exits: ");
    if(this.aCurrentRoom.aNorthExit != null){
        System.out.print("north");
    }
    if(this.aCurrentRoom.aSouthExit != null){
        System.out.print("south");
    }
    if(this.aCurrentRoom.aEastExit != null){
        System.out.print("east");
    }
    if(this.aCurrentRoom.aWestExit != null){
        System.out.print("West");
    }
    System.out.print("");
}
```

Il est recommandé d'appeler la méthode "printLocationInfo()" lorsqu'elle est nécessaire, comme dans l'exemple « goRoom()".

Ex 7.6

On utilise l'encapsulation pour réduire le couplage car là class Room et Game ont un couplage fort. L'encapsulation qui est un principe qui cache les informations liées à l'implantation. Dans la class Room on ajoute un accesseur.

```java
public Room getExit(final String pDirection)
{
    if(direction.equals("nord")){
        return aNorthExit;
    }
    if(direction.equals("sud")){
        return aSouthExit;
    }
    if(direction.equals("est")){
        return aEastExit;
    }
    if(direction.equals("ouest")){
        return aWestExit;
    }
    if(direction.equals("haut")){
        return aUpExit;
    }
    if(direction.equals("bas")){
        return aDownExit;
    }
    return null;
}
```

Du coup on peut modifier la partie b) de la méthode goRoom dans la class Game comme ça :

```java
// b)
Room vNextRoom = this.aCurrentRoom.getExit(vDirection);
```

Puis on ajoute deux nouvelles sorties up et down à la class Room, on les ajoute à la méthode setExits et a l'accesseur. Alors dans la class Game on modifie les méthodes createRooms() et printLocationInfo().

Exemple :

```
public void printLocationInfo(){
    System.out.println("Vous êtes actuellemnt "+this.aCurrentRoom.getDescription());
    System.out.print("Sorties: ");
    if(this.aCurrentRoom.aNorthExit != null){
        System.out.print("nord");
    }
    if(this.aCurrentRoom.aSouthExit != null){
        System.out.print("sud");
    }
    if(this.aCurrentRoom.aEastExit != null){
        System.out.print("est");
    }
    if(this.aCurrentRoom.aWestExit != null){
        System.out.print("ouest");
    }
    if(this.aCurrentRoom.aUpExit != null){
        System.out.print("haut");
    }
    if(this.aCurrentRoom.aDownExit != null){
        System.out.print("bas");
    }
    System.out.print("");
}
```

Ex : 7.7

getExitString() doit retourner la String contenant la liste de toutes les sorties de la pièce.

```
public String getExitString()
{
    String vE = "Sorties : ";
    if (this.getExit("nord") !=null){
        vE +="nord";
    }
    if (this.getExit("sud") !=null){
        vE +="sud";
    }
    if (this.getExit("est") !=null){
        vE +="est";
    }
    if (this.getExit("ouest") !=null){
        vE +="ouest";
    }
    if (this.getExit("haut") !=null){
        vE +="haut";
    }
    if (this.getExit("bas") !=null){
        vE +="bas";
    }
    return vE;
}
```

Puis la méthode printLocationInfo() dans la class Game devient :

```
public void printLocationInfo(){
    System.out.println("Vous êtes actuellemnt "+this.aCurrentRoom.getDescription());
    System.out.print("Sorties: ");
    System.out.print(this.aCurrentRoom.getExitString());
    System.out.print("");
}
```

Ex : 7.8

En ajoutant les directions verticales, il faut importer la classe HashMap dans la classe Room. Cette dernière permet de stocker plusieurs variables en utilisant un seul attribut, au lieu d'avoir un attribut distinct pour chaque direction. Dans la classe Game, il faut redéfinir les sorties de chaque pièce pour permettre les déplacements dans toutes les directions, y compris vers le haut et vers le bas. Si une direction ne possède pas de sortie, elle ne sera pas prise en compte.
La class Room, incluant l'intégration de la classe HashMap :

```
import java.util.HashMap;
/**
 * Classe Room - un lieu du jeu d'aventure Zuul.
 *
 * @Anastasia Tsundyk
 */

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

    public String getDescription()
    {
        return this.aDescription;
    }

    public void setExits(final String pDirection, final Room pSuivante)
    {
        aExits.put(pDirection,pSuivante);
    }

    public Room getExit(final String pDirection)
    {
        return aExits.get(pDirection);
    }
```

La procédure createRooms() dans la class Game :

```
//positionner les sorties

vOutside.setExits("est",vPsycheEtLamour);
vPsycheEtLamour.setExits("sud",vPieceSecreteUn);
vPsycheEtLamour.setExits("est",vLaVenus);
vPsycheEtLamour.setExits("ouest",vOutside);
vPieceSecreteUn.setExits("nord",vPsycheEtLamour);
vPieceSecreteUn.setExits("est",vJoconde);
vJoconde.setExits("nord",vLaVenus);
vJoconde.setExits("ouest",vPieceSecreteUn);
vLaVenus.setExits("sud",vJoconde);
vLaVenus.setExits("est",vPieceSecreteTrois);
vLaVenus.setExits("ouest",vPsycheEtLamour);
vLaVenus.setExits("haut",vPieceSecreteDeux);
vPieceSecreteDeux.setExits("bas",vLaVenus);
vPieceSecreteTrois.setExits("nord",vFinal);
vPieceSecreteTrois.setExits("est",vPieceSecreteQuatre);
vPieceSecreteTrois.setExits("ouest",vLaVenus);
vPieceSecreteQuatre.setExits("nord",vLaLiberteGuidantLePeuple);
vPieceSecreteQuatre.setExits("sud",vHermaphroditeEndormi);
vPieceSecreteQuatre.setExits("ouest",vPieceSecreteTrois);
vPieceSecreteQuatre.setExits("haut",vLesNocesDeCana);
vHermaphroditeEndormi.setExits("nord",vPieceSecreteQuatre);
vLesNocesDeCana.setExits("bas",vPieceSecreteQuatre);
vLaLiberteGuidantLePeuple.setExits("sud",vPieceSecreteQuatre);
vLaLiberteGuidantLePeuple.setExits("ouest",vFinal);
vFinal.setExits("sud",vPieceSecreteTrois);
vFinal.setExits("est",vLaLiberteGuidantLePeuple);
//initialiser lieu courrant
```

Ex : 7.9

En classe Room, on veut obtenir les directions liées à un objet en utilisant la classe Set. Pour cela, on suit la méthode. On doit également utiliser une boucle pour parcourir les données stockées dans un objet de type HashMap qui contient les directions. La méthode keySet de HashMap retourne un ensemble de clés associées à cet objet.

La class Room modifié :

```
import java.util.Set;
/**
```

```
public String getExitString()
{
    String returnString = "Sorties : ";
    Set<String>Keys= aExits.keySet();
    for(String exit : Keys)
    {
        returnString += " "+exit;
    }
    return returnString;
}
Room
```

Ex : 7.10

La méthode getExitString(), renvoie une chaîne de caractères (String) représentant les sorties d'une salle. Le code commence par initialiser une variable de type String appelée returnString avec la valeur "Sorties : ". Ensuite, il crée un ensemble (Set) de chaînes de caractères appelé "Keys", qui contient toutes les clés de aExits qui associe chaque sortie possible à une direction. Le code le fait ensuite sur chaque clé dans l'ensemble "Keys" à l'aide d'une boucle for-each. Pour chaque clé, le code ajoute la chaîne « exit » à la variable "returnString" à l'aide de « += ». Enfin, le code renvoie la variable returnString, qui contient toutes les sorties sous la forme d'une chaîne de caractères. Ainsi, si la map aExits contient les clés « nord », « sud », « est», « ouest », « haut » et « bas » la méthode getExitString renverra la chaîne de caractères « Sorties : nord sud est ouest haut bas ». Car getExitString() doit retourner la String contenant la liste de toutes les sorties de la pièce.

Ex : 7.11

Nous pouvons encore réduire le couplage des class Game et Room nous allons utiliser une méthode getLongDescription () dans la class Room qui permettra de créer la description d'une pièce et de donner les sorties. Ensuite il suffira d'appeler cette méthode dans printLocationInfo() de la classe Game.

```
 *
 */public String getLongDescription()
{
    return" Vous etes "+aDescription+".\n"+getExitString();
}
Room
```

```
*/
ublic void printLocationInfo(){
   System.out.println(aCurrentRoom.getLongDescription());
```

Ex : 7.14

On veut créer une méthode "look". Pour commencer, il sera nécessaire d'ajouter la commande "regarder" à la liste des commandes valides dans la class CommandWords. Ensuite, nous devrons ajouter une procédure "look" à la classe Game, qui permettra d'examiner le contenu de la pièce. Enfin, nous devrons modifier la méthode processCommand () de la class Game pour permettre au joueur d'entrer la commande "regarder", de sorte que le programme exécute la méthode "look" en conséquence.

Voilà les modifications :

```java
public class CommandWords
{
    // a constant array that will hold all valid command words
    private final String[] aValidCommands;

    /**
     * Constructor - initialise the command words.
     */
    public CommandWords()
    {
        this.aValidCommands = new String[4];
        this.aValidCommands[0] = "go";
        this.aValidCommands[1] = "aide";
        this.aValidCommands[2] = "quitter";
        this.aValidCommands[3] = "regarder";
    } // CommandWords()
```

```java
    /**
     * pour afficher la commande donne
     */
    private void look()
    {
        System.out.println(aCurrentRoom.getLongDescription());
    }
    Fonction pour les commandes principales.

    /private boolean processCommand(final Command pCommand)

      if (pCommand.isUnknown())
      {
          System.out.println("Je ne comprend pas...");
          return false;
      }
      else
      {
          if(pCommand.getCommandWord().equals("quitter"))
          {
              return quit(pCommand);
          }
          else if (pCommand.getCommandWord().equals("aide"))
          {
             printHelp();
          }
          else if (pCommand.getCommandWord().equals("go"))
          {
              goRoom(pCommand);
          }
          else if (pCommand.getCommandWord().equals("regarder"))
          {
              look();
          }
          return false;
          // Game
      }
```

Ex : 7.15

On ajoute une commande manger, il suffit de faire comme à la question précédente.

```java
     * Constructor - initialise the command words.
     */
    public CommandWords()
    {
        this.aValidCommands = new String[5];
        this.aValidCommands[0] = "go";
        this.aValidCommands[1] = "aide";
        this.aValidCommands[2] = "quitter";
        this.aValidCommands[3] = "regarder";
        this.aValidCommands[4] = "manger";
    } // CommandWords()
```

```java
private void eat()
{
    System.out.println("Vous avez mange maintenant et vous n avez plus faim")
}
```

```
else
{
    if(pCommand.getCommandWord().equals("quitter"))
    {
        return quit(pCommand);
    }
    else if (pCommand.getCommandWord().equals("aide"))
    {
       printHelp();
    }
    else if (pCommand.getCommandWord().equals("go"))
    {
        goRoom(pCommand);
    }
    else if (pCommand.getCommandWord().equals("regarder"))
    {
        look();
    }
    else if (pCommand.getCommandWord().equals("manger"))
    {
        eat();
    }
    return false;
    // Game
```

Ex :7.16

Afin de ne pas oublier toutes les commandes affichées dans aide, il faut créer une méthode (showAll) dans la classe CommandWords qui permettra de voir toutes les commandes valides.

```
/**
 * Affiche un eliste des commandes valides.
 */
public void showCommands()
{
    this.aValidCommands.showAll();
}
```

```
 * Procedure pour afficher les phrases d'aide.
 *
 */
private void printHelp(){
   System.out.println("You are lost. You are alone.");
   System.out.println("You wander around at the university.");
   System.out.println("Your command words are:");
   System.out.println("go quit help");
   System.out.println("Vos commandes sont : ");
   aParser.showCommands();
```

```
/**
 * Affiche toutes les commandes valides sur System.out.
 */
public void showAll()
{
    for(String command : aValidCommands){
        System.out.print(command+' ');
    }
    System.out.println();
}
```

Ex :7.18

Tout d'abord, la méthode showAll doit être renommée et convertie en getCommandList. Dans cette méthode, une nouvelle variable doit être créée pour stocker toutes les commandes disponibles, puis la méthode doit retourner cette liste de commandes sous forme de chaîne de caractères. Ensuite, la méthode showCommands de la classe Parser doit également être modifiée pour retourner la chaîne de caractères contenus dans getCommandList. Enfin, dans la classe Game, la méthode printHelp doit être modifiée pour afficher la liste de commandes à l'aide de la méthode getCommandList de la classe CommandWords, au lieu d'utiliser la méthode showCommands de la classe Parser qui affichait la liste de commandes.

Voici les modifications :

```java
/**
 * Affiche toutes les commandes valides.
 */
public String getCommandList()
{
    String vResult =" ";
    for(String command : aValidCommands){
        vResult += command+ " ";
    }
    return vResult;
}
```

```java
/**
 * Affiche un eliste des commandes valides.
 */
public String showCommands()
{
    return this.aValidCommands.getCommandList();
}
```

```java
/**
 * Procedure pour afficher les phrases d'aide.

 */
private void printHelp(){
    System.out.println("Vous êtes perdu.");
    System.out.println("go quitter aide");
    System.out.println("Vos commandes sont : ");
    System.out.println(aParser.showCommands());
}
```

Ex : 7.18.6

Il n'y a pas de changement dans les classes Command et CommandWords. La classe Room a été mise à jour avec une méthode et un attribut supplémentaire, ce qui nécessitera des modifications dans la méthode createRooms. Une nouvelle version de la classe Parser a été développée et ne nécessite plus de Scanner. La majeure partie de la classe Game a été transférée dans la classe GameEngine. La classe GameEngine remplace essentiellement l'ancienne classe Game. La méthode printLocationInfo, est adapter pour qu'elle affiche également l'image de la pièce.
Pour améliorer la programmation du début de la classe UserInterface, il est recommandé de remplacer chaque ligne contenant l'instruction import ... .*; par les imports spécifiques des classes nécessaires.

Ex: 7.18.8

Afin de créer un bouton dans la classe UserInterface, il est nécessaire d'importer la classe JButton et de déclarer un nouvel attribut aButton. Dans le constructeur, il est également nécessaire d'ajouter l'appel de la méthode setGUI, sinon l'interface graphique ne sera pas affichée. Dans la méthode createGUI, il faudra créer un nouveau bouton et lui attribuer une position.

Modifications :

```
this.aImage = new JLabel();
JPanel vPanel = new JPanel();
this.aButton = new JButton("regarder");

vPanel.setLayout( new BorderLayout() ); // ==> only five places
vPanel.add( this.aImage, BorderLayout.NORTH );
vPanel.add( vListScroller, BorderLayout.CENTER );
vPanel.add( this.aEntryField, BorderLayout.SOUTH );
vPanel.add( this.aButton, BorderLayout.EAST );

this.aMyFrame.getContentPane().add( vPanel, BorderLayout.CENTER );

// add some event listeners to some components
this.aEntryField.addActionListener( this );
this.aButton.addActionListener( this );
```

```
/**
 * Actionalise interface pour les entrees de texte.
 */
@Override public void actionPerformed( final ActionEvent pE )
{
    if (pE.getSource() == this.aButton) {
        this.aEngine.interpretCommand("regarder");
    }
    else {
        this.processCommand(); // never suppress this line
    }
} // actionPerformed(.)
```

Ex: 7.19.2

Pour commencer, il faut créer un dossier pour stocker nos images à la racine du jeu, puis les intégrer dans notre code.

```
Room vOutside = new Room(" a l exterieur du musee","Images/Louvre.jpg.jpeg");
Room vPsycheEtLamour = new Room(" dans la piece de l oeuvre Psyche et l amour", "Images/Psyche.jpg");
Room vPieceSecreteUn = new Room(" dans une piece secrete", "Images/piece.jpg");
Room vJoconde = new Room(" dans la piece de l oeuvre La Joconde","Images/Joconde.jpg");
Room vLaVenus = new Room(" dans la piece de l oeuvre La Venus", "Images/Venus.jpg");
Room vPieceSecreteDeux = new Room(" dans une piece secrete", "Images/piece2.jpg");
Room vPieceSecreteTrois = new Room(" dans une piece secrete", "Images/piece3.jpg");
Room vPieceSecreteQuatre = new Room(" dans une piece secrete", "Images/piece4.jpg");
Room vHermaphroditeEndormi = new Room(" dans la piece de l oeuvre Hemaphrodite Endormi", "Images/endormi.JPG");
Room vLesNocesDeCana = new Room(" dans la piece de l oeuvre Les noces de Cana", "Images/noce.jpg");
Room vLaLiberteGuidantLePeuple = new Room(" dans la piece de l oeuvre La liberté guidant le peuple", "Images/liberte.jpg");
Room vFinal = new Room(" dans la piece avec la bombe", "Images/final.jpg.jpeg");
```

Ex: 7.20

On créer une classe appelée "Item" pour mettre des objets dans les pièces. Chaque objet aura une description et un poids. Dans la classe "Room", on ajoute un attribut "aItem"

pour stocker un seul objet par pièce. On peut créer un objet en utilisant la méthode "setItem" et le récupérer en utilisant la méthode "getItem". Dans la méthode "createRooms" de la classe "Game Engine", on peut créer un objet en définissant sa description et son poids avec "setItem" pour une pièce spécifique. Pour que le joueur sache quels objets sont dans la pièce lorsqu'il y entre, on peut modifier la procédure "printLocationInfo". Cette dernière affichera un message avec la description de l'objet (obtenue avec la méthode "getDescription" de la classe "Item") et le poids de l'objet (obtenu avec la méthode "getPoids" de la classe "Item") dans la pièce où se trouve le joueur.

```java
public class Item {
    private String aDescription;
    private int aWeight;

    /**
     * Constructeur de la classe Item.
     */
    public Item(final String pDescription,final int pWeight) {
        this.aDescription = pDescription;
        this.aWeight = pWeight;
    }

    /**
     * Accesseur pour la description de l'objet.
     * @return la description de l'objet.
     */
    public String getDescription() {
        return this.aDescription;
    }

    /**
     * Accesseur pour le poids de l'objet.
     * @return le poids de l'objet.
     */
    public int getWeight() {
        return this.aWeight;
    }
}
```

```java
public void setItem(final Item pItem)
{
    this.aItem = pItem;
}

public Item getItem()
{
    return this.aItem;
}
```

```java
vOutside.setExits("est",vPsycheEtLamour);
vPsycheEtLamour.setExits("sud",vPieceSecreteUn);
vPsycheEtLamour.setExits("est",vLaVenus);
vPsycheEtLamour.setExits("ouest",vOutside);

vPieceSecreteUn.setExits("nord",vPsycheEtLamour);
vPieceSecreteUn.setExits("est",vJoconde);
vPieceSecreteUn.setItem(new Item("Pomme", 4));

vJoconde.setExits("nord",vLaVenus);
vJoconde.setExits("ouest",vPieceSecreteUn);

vLaVenus.setExits("sud",vJoconde);
vLaVenus.setExits("est",vPieceSecreteTrois);
vLaVenus.setExits("ouest",vPsycheEtLamour);
vLaVenus.setExits("haut",vPieceSecreteDeux);

vPieceSecreteDeux.setExits("bas",vLaVenus);
vPieceSecreteDeux.setItem(new Item("Cle", 10));

vPieceSecreteTrois.setExits("nord",vFinal);
vPieceSecreteTrois.setExits("est",vPieceSecreteQuatre);
vPieceSecreteTrois.setExits("ouest",vLaVenus);
vPieceSecreteTrois.setItem(new Item("Chocolat", 4));

vPieceSecreteQuatre.setExits("nord",vLaLiberteGuidantLePeuple);
vPieceSecreteQuatre.setExits("sud",vHermaphroditeEndormi);
vPieceSecreteQuatre.setExits("ouest",vPieceSecreteTrois);
vPieceSecreteQuatre.setExits("haut",vLesNocesDeCana);
vPieceSecreteQuatre.setItem(new Item("Cle", 10));

vHermaphroditeEndormi.setExits("nord",vPieceSecreteQuatre);

vLesNocesDeCana.setExits("bas",vPieceSecreteQuatre);

vLaLiberteGuidantLePeuple.setExits("sud",vPieceSecreteQuatre);
vLaLiberteGuidantLePeuple.setExits("ouest",vFinal);

vFinal.setExits("sud",vPieceSecreteTrois);
vFinal.setExits("est",vLaLiberteGuidantLePeuple);
//initialiser lieu courrant
```

```java
/**
 * on crée une méthode
printLocationInfo() dans la classe Game qui permet à chaque appel
de connaître la sortie de la salle actuelle.
 *
 */
public void printLocationInfo(){
    aGui.println( this.aCurrentRoom.getLongDescription() );
    aGui.println("Description de l item : "+this.aCurrentRoom.getItem().getDescription());
    aGui.println("Poids de l objet : "+this.aCurrentRoom.getItem().getWeight());
    this.aGui.showImage(this.aCurrentRoom.getImageName());
}
```

A partir de cet exercice j'ai reçu de l'aide de la part de mes camarades, de personnes externes et j'ai utilisée dans certaine partie ChatGPT.

Ex: 7.21

Nous allons apporter quelques modifications à la class Item

```java
public String getItemDescription()
{
    return "Description de l item : "+this.aDescription+ "\n"+"Poids de litem : "+this.aWeight;
}
```

```java
* Renvoie une description detaiillee de cette piece sous la forem
*      vous etes dans la cuisine.
*      Sorties : nord ouest
*@return Une description de la piece, avec les sorties.
*
*
*/public String getLongDescription()

   return "Vous etes actuellement"+this.aDescription+".\n"+aItem.getItemDescription()+"\n"+getExitString()+"\n";
```

```java
 * on crée une méthode
rintLocationInfo() dans la classe Game qui permet à chaque appel
le connaître la sortie de la salle actuelle.
 *
 */
ublic void printLocationInfo(){
    aGui.println( this.aCurrentRoom.getLongDescription() );
    this.aGui.showImage(this.aCurrentRoom.getImageName());
```

Ex:7.22

Il faut utiliser une collection de données appelée "HashMap" pour stocker un plus grand nombre d'objets dans une pièce. Cette collection est caractérisée par une clé qui identifie chaque objet et une valeur qui sera le nom de la collection. On créer cette HashMap dans la classe "Room". Ensuite, on peut ajouter des objets en utilisant la méthode "put" et accéder aux objets en utilisant la méthode "get". Enfin on utilise la procédure "ajouteItem" pour ajouter un nouvel objet dans la pièce.

```java
private String aDescription;
private String aName;
private int aWeight;
/**
 * Constructeur de la classe Item.
 */
public Item(final String pDescription,final String pName, final int pWeight) {
    this.aDescription = pDescription;
    this.aName = pName;
    this.aWeight = pWeight;
}
```

```java
public String aDescription;
private HashMap<String, Room>aExits;
private String aImageName;
private HashMap<String, Item>aItems;
/**
 * Constructeur Room() naturel
 *
 */public Room(final String pDescription, final String pImage )
{
    this.aDescription = pDescription;
    aExits = new HashMap<String, Room>();
    aItems = new HashMap<String, Item>();
    this.aImageName = pImage;
}
```

```java
/**
 * ajout item
 */
public void ajoutItem(final String pNom, final Item pItems)
{
    this.aItems.put(pNom, pItems);
}

public Item getItem(final String pString)
{
    return this.aItems.get(pString);
}
```

```java
/**
 * accesseur donne ddescription items
 */
public String getItemString()
{
    StringBuilder vItemString = new StringBuilder("Items : ");
    Set <String> vItemsNames = this.aItems.keySet();
    for (String vName : vItemsNames) {
        vItemString.append(" "+vName);
    }
    return vItemString.toString();
}
```

```java
/**
 * Renvoie une description detaillee de cette piece
 *
 */public String getLongDescription()
{
    return "Vous etes actuellement"+this.aDescription+".\n"+ getItemString() +"\n"+getExitString()+"\n";
}
```

```java
//creation Items
Item vPain =new Item("Pain","nourriture",2);
Item vChocolat =new Item("Chocolat","nourriture",5);
Item vCle = new Item("Cle","pour ouvrir la porte final",10);
Item vCle2 = new Item("Cle2","pour ouvrir la porte final",10);
Item vPomme =new Item("Pomme","nourriture",3);
Item vPiment =new Item("Piment","nourriture",2);
Item vSablier = new Item("Sablier","minuteur",0);
```

```
//positionner les sorties

vOutside.setExits("est",vPsycheEtLamour);
vOutside.ajoutItem("Sablier",vSablier);

vPsycheEtLamour.setExits("sud",vPieceSecreteUn);
vPsycheEtLamour.setExits("est",vLaVenus);
vPsycheEtLamour.setExits("ouest",vOutside);

vPieceSecreteUn.setExits("nord",vPsycheEtLamour);
vPieceSecreteUn.setExits("est",vJoconde);
vPieceSecreteUn.ajoutItem("Pomme",vPomme);
vPieceSecreteUn.ajoutItem("Piment",vPiment);

vJoconde.setExits("nord",vLaVenus);
vJoconde.setExits("ouest",vPieceSecreteUn);

vLaVenus.setExits("sud",vJoconde);
vLaVenus.setExits("est",vPieceSecreteTrois);
vLaVenus.setExits("ouest",vPsycheEtLamour);
vLaVenus.setExits("haut",vPieceSecreteDeux);

vPieceSecreteDeux.setExits("bas",vLaVenus);
vPieceSecreteDeux.ajoutItem("Cle",vCle);

vPieceSecreteTrois.setExits("nord",vFinal);
vPieceSecreteTrois.setExits("est",vPieceSecreteQuatre);
vPieceSecreteTrois.setExits("ouest",vLaVenus);
vPieceSecreteTrois.ajoutItem("Chocolat",vChocolat);
vPieceSecreteTrois.ajoutItem("Pain",vPain);

vPieceSecreteQuatre.setExits("nord",vLaLiberteGuidantLePeuple);
vPieceSecreteQuatre.setExits("sud",vHermaphroditeEndormi);
vPieceSecreteQuatre.setExits("ouest",vPieceSecreteTrois);
vPieceSecreteQuatre.setExits("haut",vLesNocesDeCana);
vPieceSecreteQuatre.ajoutItem("Cle2",vCle2);
```

Ex: 7.23

On veut créer une nouvelle fonctionnalité dans le jeu qui permet au joueur de revenir à la pièce précédente en utilisant la commande "retour". Pour cela, on ajoute un attribut à la classe GameEngine pour stocker la pièce précédente, crée une procédure pour gérer la commande "retour", ajoute une vérification pour cette commande dans la procédure "interpretCommand" de la classe GameEngine, et on ajoute "retour" à la liste des commandes valides dans la classe CommandWords.

```
// c)
if (vNextRoom == null) {
    this.aGui.println("Ici,il n'y a pas de porte !");
    return;
}else
{this.aPreceRoom = this.aCurrentRoom;
 this.aCurrentRoom= vNextRoom;
 printLocationInfo();
}

// d)
```

```
 * retour en arriere
 *
 */
private void back(final Command pCommand) {
  if(!pCommand.hasSecondWord()){
  Room vCurrentRoom = this.aCurrentRoom;
  this.aCurrentRoom = this.aPreceRoom;
  this.aPreceRoom = vCurrentRoom;
  printLocationInfo();
}else{
    this.aGui.println("Je ne comprend pas.");
}
}
```

```
else if (vCommandWord.equals("retour")){
    this.back();
}
```

```
 * Constructor - initialise the command words.
 */
public CommandWords()
{
    this.aValidCommands = new String[6];
    this.aValidCommands[0] = "go";
    this.aValidCommands[1] = "aide";
    this.aValidCommands[2] = "quitter";
    this.aValidCommands[3] = "regarder";
    this.aValidCommands[4] = "manger";
    this.aValidCommands[4] = "retour";
} // CommandWords()
```

Ex: 7.26

On cherche à créer une pile de pièces pour un jeu. Pour cela, on utilise une classe appelée Stack et on ajoute un attribut de type pile de pièces dans la classe GameEngine. Dans le constructeur de la classe GameEngine, on initialise l'attribut pour qu'il soit une nouvelle pile de pièces. Dans la procédure goRoom, on utilise la méthode push pour ajouter la pièce courante à la pile de pièces. Dans la procédure back, on vérifie d'abord si la pile de pièces est vide en utilisant la méthode empty. Si la pile est vide, on affiche qu'il n'y a pas de porte. Sinon, on prend la première pièce de la pile et on la retire de la pile. Cette pièce devein alors la pièce courante.

```
/**
 * Affiche les détails d'un élément spécifique ou la description complète de la pièce.
 * @param pCommand Commande de l'utilisateur transmise.
 */
private void look(final Command pCommand) {
    if (pCommand.hasSecondWord()) {
        String vItemName = pCommand.getSecondWord();
        Item vItem = this.aPlayer.getCurrentRoom().getRoomItems().getItem(vItemName);
        if (vItem == null) {
            this.aGui.println("Il n'y a pas d'objet ici avec ce nom.");
        } else {
            this.aGui.println(vItem.getItemDescription());
        }
    } else {
        this.aGui.println(this.aPlayer.getCurrentRoom().getLongDescription());
    }
}
```

```
public class GameEngine
{
    private Parser        aParser;
    private Room          aCurrentRoom;
    private UserInterface aGui;
    private Player aPlayer;
    private HashMap <String, Room> aRooms;
    /**
     * Constructor for objects of class GameEngine
     */
    public GameEngine()
    {
        this.aParser = new Parser();
        this.createRooms();
        this.aRooms = new HashMap <String, Room>();

    }
```

```
String vDirection = pCommand.getSecondWord();

// c)
if (this.aPlayer.getCurrentRoom().getExit(vDirection) == null) {
    this.aGui.println("Ici,il n'y a pas de porte !");
    return;
}
this.aPlayer.goRoom(vDirection);
printLocationInfo();
```

Ex :7.28

Un nouveau commandement appelé "test" est ajouté au jeu, qui prendra un fichier texte en paramètre contenant des commandes à exécuter. Le tableau de CommandWords est mis à jour et la méthode test() est ajoutée à interpretCommand() dans GameEngine. La méthode test() vérifie s'il y a un deuxième mot, ajoute l'extension de fichier si nécessaire et gère les erreurs à l'aide d'un bloc TRY/CATCH. Dans le bloc TRY, la classe Scanner est utilisée pour lire chaque ligne du fichier texte et interpréter les commandes dans la ligne via interpretCommand(). Si le fichier n'existe pas ou a un nom invalide, le bloc CATCH gère l'erreur en affichant un message d'erreur. La méthode test() est ajoutée à GameEngine dans interpretCommand().

```
private void test(final Command pCommand)
{
    if (!pCommand.hasSecondWord()){
       this.aGui.println("Pouvez vous ecrire le nom de votre fichier apres la commande test
    }
    String vFile=pCommand.getSecondWord();
    if(!vFile.contains(".txt")){
        vFile +=".txt";
    }
    try{//on effectue les instructions dans le cas où le fichier existe.
        Scanner vScan = new Scanner (new File (vFile));
        while(vScan.hasNextLine()){
            String vLine = vScan.nextLine();
            interpretCommand(vScan.nextLine());
        }
    }
    catch(final FileNotFoundException pE)//Ce dernier va nous permettre de de traiter l'err
    {
        this.aGui.println("Le dossier "+ vFile + " n'existe pas.");
    }
}
```

Ex: 7.28.2

Creation de 3 fichier .TXT.

Ex: 7.29

La classe Player est créée pour contenir toutes les informations relatives au joueur, telles que sa position actuelle, ses objets et son historique de déplacement. GameEngine contiendra désormais une instance de la classe Player pour gérer les déplacements et les actions du joueur. Certaines méthodes de GameEngine sont modifiées pour faire appel aux méthodes de Player. Enfin, des méthodes spécifiques à la classe Stack sont déplacées dans Player pour faciliter la gestion des déplacements.

```java
public class Player
{
    // variables d'instance -
    private String aPseudo;
    private double aPoids;
    private Room aCurrentRoom;
    private Stack<Room> aPreceRooms;
    private HashMap <String, Item> aItems;
    /**
     * Constructeur d'objets de classe Player
     */
    public Player(final String pPseudo, final Room pCurrentRoom)
    {
        this.aPseudo = pPseudo;
        this.aPoids = 18.0;
        this.aPreceRooms = new Stack<Room>();
        this.aItems = new HashMap<String, Item>();
        this.aCurrentRoom = pCurrentRoom;
    }

    /**
     * Accesseur du pseudonyme
     *
     */
    public String getPseudo ()
    {
        return this.aPseudo;
    }

    public Room getCurrentRoom(){
        return this.aCurrentRoom;
    }
```

```java
/**
 * retourne pile de piece precedent
 */
public Stack <Room> getPreceRoom()
{
    return this.aPreceRooms;
}

/**
 * Procédure déplaçant le joueur de pièces en pièces
 *
 * @param pStringRoom Direction de la pièce à récupérer
 */
public void goRoom (final String pDirection)
{
    Room vNextRoom = this.aCurrentRoom.getExit(pDirection);
    this.aPreceRooms.push(this.aCurrentRoom);
    this.aCurrentRoom = vNextRoom;
}// goRoom ()

 /**
 * Permet de revenir sur ses pas
 */
public void back ()
{
    this.aCurrentRoom = this.aPreceRooms.pop(); //Change la pièce courante par celle d'après
}// back ()
```

Ex: 7.31

Dans cet exercice, nous ajoutons la possibilité pour le joueur de ramasser des objets dans la pièce. Nous implémentons deux nouvelles commandes, "prendre" et "lâcher", et créons une structure de données pour stocker les objets ramassés par le joueur. Nous ajoutons également des méthodes pour analyser et exécuter ces commandes, ainsi que pour afficher le contenu du sac du joueur.

```java
public void takeItemPlayer( final String pNameItem, final Item pItem)
{
    this.aItems.put(pNameItem, pItem);
}

public void dropItemPlayer(final String pNameItem, final Item pItem)
{
    this.aItems.remove(pNameItem, pItem);
}
}
```

```java
public void removeItem(final String pNameItem, final Item pItem)
{
    this.aItems.remove(pNameItem,pItem);
}
```

Room

```java
private void take(final Command pCommand)
{
    if(!pCommand.hasSecondWord()){
        this.aGui.println("Prendre quoi?");
    }
    String vItem = pCommand.getSecondWord();
    Item vPrendre = this.aPlayer.getCurrentRoom().getItem(vItem);
    if(vPrendre == null) {
        this.aGui.println("Il n y a pas "+ vPrendre.getName()+" ici.");
    }else{
        this.aPlayer.takeItemPlayer(vItem, vPrendre);
        this.aPlayer.getCurrentRoom().removeItem(vItem, vPrendre);
        this.aGui.println("Tu a l item "+ vPrendre.getName() +" !");
    }

}

private void drop(final Command pCommand)
{
    if(!pCommand.hasSecondWord()){
        this.aGui.println("Laisser quoi?");
    }
    String vItem = pCommand.getSecondWord();
    Item vLaisser = this.aPlayer.getItemPlayer(vItem);
    if(vLaisser == null) {
        this.aGui.println("Tu n a pas lache "+ vLaisser.getName());
    }else{
        this.aPlayer.dropItemPlayer(vItem, vLaisser);
        this.aPlayer.getCurrentRoom().ajoutItem(vItem, vLaisser);
        this.aGui.println("Tu a laisse l item, "+ vLaisser.getName() +" dans la piece!");
    }

}
```

```java
public void take(final String pItem)
{
    this.aItemsBag.put(pItem,this.aCurrentRoom.getItem(pItem));
    this.aCurrentRoom.removeItem(pItem);
}
}
```

```java
public void drop(final String pItem)
{
    this.aItemsBag.addItem(pItem,this.getItem(pItem));
    this.aItemBag.remove(pItem);
}
}
```

Ex: 7.31.1

On créé une classe ItemList pour gérer une liste d'items et éviter la duplication de code entre les classes Room et Player. La classe ItemList contient une HashMap pour stocker les items, ainsi que des méthodes pour ajouter et enlever des items de la liste, vérifier si un item est présent dans la liste et obtenir une chaîne de caractères contenant tous les objets de la

pièce. Dans la classe Room, on crée une liste d'items dans le constructeur et on modifie la méthode getLongdescription pour utiliser la méthode getItemString de la classe ItemList. Dans la classe GameEngine, on utilise getItemList pour ajouter un objet à la pièce. Dans la classe Player, on modifie les méthodes take et drop pour ajouter ou enlever des items de la liste d'items du joueur et de la pièce. Des messages sont affichés pour indiquer si l'opération a réussi ou échoué.

```java
ublic class ItemList
{
    private HashMap <String, Item> aItems; // HashMap d'items
    private String aLocation; // La où les items sont (inventaire ou pièce)

    /**
     * Constructeur naturel
     *
     * @param pLocation Localisation de la liste d'items
     */
    public ItemList (final String pLocation)
    {
        this.aItems = new HashMap <String, Item> ();
        this.aLocation = pLocation;
    }// ItemList ()

    /**
     * Accesseur de l'item
     *
     * @param pItem Chaine de caractères de l'item auquel on souhaite acceder dans la liste d'items
     * @return L'item recherché par son nom
     */
    public Item getItem (final String pItem)
    {
        return this.aItems.get(pItem);
    }// getItem ()

    /**
     * Accesseur de la liste d'items
     *
     * @return HashMap contenant tous les items
     */
    public HashMap <String, Item> getItems ()
    {
        return aItems;
    }// getItems ()
```

```java
    **
    * Fonction qui retourne sous forme de chaine de caractère la liste d'items
    *
    * @return Chaine de caractères de la liste
    */
    ublic String getItemString ()

        if (this.aItems.isEmpty()){
            return "There is no items in " + this.aLocation;
        }// if ()

        StringBuilder vChaine = new StringBuilder ("Items in " + this.aLocation + " :");
        Set <String> vItemsNames = this.aItems.keySet();
        for (String vName : vItemsNames){
            vChaine.append(" " + vName);
        }// for ()
        return vChaine.toString();
    // getItemString ()

    **
    * Procédure qui ajoute un item à la liste d'items
    *
    * @param pName Clef de la HashMap (nom de l'item)
    * @param pItem Item à insérer dans la HashMap
    */
    ublic void ajoutItem(final String pName, final Item pItem){
        this.aItems.put(pName, pItem);
    // addItem ()

    **
    * Procédure qui supprime un item de la HashMap
    *
    * @param pName Clef de l'item à supprimer
    */
    ublic void removeItem (final String pName){
        this.aItems.remove(pName);
    // removeItem ()
```

```java
// variables d'instance -
private String aPseudo;
private double aPoids;
private Room aCurrentRoom;
private Stack<Room> aPreceRooms;
private ItemList aInvItems;
/**
 * Constructeur d'objets de classe Player
 */
public Player(final String pPseudo, final Room pCurrentRoom)
{
    this.aPseudo = pPseudo;
    this.aPoids = 18.0;
    this.aPreceRooms = new Stack<Room>();
     this.aInvItems = new ItemList ("Ton inventaire");
    this.aCurrentRoom = pCurrentRoom;
}
```

```java
public ItemList getInvItems()
{
    return this.aInvItems;
}
}
```

```java
public String aDescription;
private HashMap<String, Room>aExits;
private String aImageName;
private ItemList aRoomItems;
/**
 * Constructeur Room() naturel
 *
 */public Room(final String pDescription, final String pImage )
{
    this.aDescription = pDescription;
    this.aExits = new HashMap<String, Room>();
    this.aImageName = pImage;
    this.aRoomItems = new ItemList("piece");
}
```

```java
PsycheEtLamour.setExits("est",vLaVenus);
PsycheEtLamour.setExits("ouest",vOutside);

PieceSecreteUn.setExits("nord",vPsycheEtLamour);
PieceSecreteUn.setExits("est",vJoconde);
PieceSecreteUn.getRoomItems().ajoutItem("Pomme",vPomme);
PieceSecreteUn.getRoomItems().ajoutItem("Piment",vPiment);
Joconde.setExits("nord",vLaVenus);
Joconde.setExits("ouest",vPieceSecreteUn);

LaVenus.setExits("sud",vJoconde);
LaVenus.setExits("est",vPieceSecreteTrois);
LaVenus.setExits("ouest",vPsycheEtLamour);
LaVenus.setExits("haut",vPieceSecreteDeux);

PieceSecreteDeux.setExits("bas",vLaVenus);
PieceSecreteDeux.getRoomItems().ajoutItem("Cle",vCle);

PieceSecreteTrois.setExits("nord",vFinal);
PieceSecreteTrois.setExits("est",vPieceSecreteQuatre);
PieceSecreteTrois.setExits("ouest",vLaVenus);
PieceSecreteTrois.getRoomItems().ajoutItem("Chocolat",vChocolat);
PieceSecreteTrois.getRoomItems().ajoutItem("Pain",vPain);

PieceSecreteQuatre.setExits("nord",vLaLiberteGuidantLePeuple);
PieceSecreteQuatre.setExits("sud",vHermaphroditeEndormi);
PieceSecreteQuatre.setExits("ouest",vPieceSecreteTrois);
PieceSecreteQuatre.setExits("haut",vLesNocesDeCana);
PieceSecreteQuatre.getRoomItems().ajoutItem("Cle2",vCle2);
```

```java
tem vItem = this.aPlayer.getCurrentRoom().getRoomItems().getItem(pCommand.getSecondWord());
f (vItem == null){ // Vérification de la présence de cet item dans la pièce
```

Ex: 7.32

On va implémenter une limite de poids et faire une gestion d'inventaire. Pour ce faire, on ajoute une propriété de poids à chaque objet dans la classe "Item" et on initialise cette propriété dans le constructeur. Dans la classe "Player", on ajoute deux propriétés pour représenter le poids maximal pouvant être porté et le poids de l'inventaire en temps réel. On vérifie ensuite si un objet peut être pris dans la méthode "take()" de la classe "GameEngine". En simplifiant la méthode "take()" et "drop()" pour utiliser des objets entiers plutôt que des chaînes de caractères, on facilite également la programmation.

```java
// variables d'instance -
private String aPseudo;
private int aMaxWeight;
private Room aCurrentRoom;
private Stack<Room> aPreceRooms;
private ItemList aInvItems;
private int aInvWeight;
/**
 * Constructeur d'objets de classe Player
 */
public Player(final String pPseudo, final Room pCurrentRoom)
{
    this.aPseudo = pPseudo;
    this.aMaxWeight = 18;
    this.aPreceRooms = new Stack<Room>();
    this.aInvItems = new ItemList ("Ton inventaire");
    this.aCurrentRoom = pCurrentRoom;
    this.aInvWeight=0;
}
```

```java
public int getMaxWeight()
{
    return this.aMaxWeight;
}

public int getInvWeight()
{
    return this.aInvWeight;
}
```

```java
/**
 * Procédure permettant de ramasser un item
 *
 * @param pItem Item à ramasser
 */
public void take (final Item pItem)
{
    this.aInvItems.ajoutItem(pItem.getName(),pItem); // Ajoute à l'inventaire l'item actue
    this.aCurrentRoom.getRoomItems().removeItem(pItem.getName()); // Supprime l'item de la
    this.aInvWeight += pItem.getWeight(); // Ajoute le poids de l'item au poids total de l
}// take ()

/**
 * Procédure permettant de déposer un item
 *
 * @param pItem Item à déposer
 */
public void drop (final Item pItem)
{
    this.aCurrentRoom.getRoomItems().ajoutItem(pItem.getName(), pItem); // Ajoute l'item a
    this.aInvItems.removeItem(pItem.getName()); // Supprime l'item de l'inventaire
    this.aInvWeight -= pItem.getWeight(); // Retire le poids de l'item au poids total de l
}// drop ()
```

```java
private void take (final Command pCommand)
{
    if (!pCommand.hasSecondWord()){ // Vérification de la présence d'un second mot
        this.aGui.println("Tu veux prendre quoi ?");
        return;
    }// if ()

    Item vItem = this.aPlayer.getCurrentRoom().getRoomItems().getItem(pCommand.getSecondWord());
    if (vItem == null){ // Vérification de la présence de cet item dans la pièce
        this.aGui.println("Il n y a pas d item ici...");
        return;
    }// if ()

    if (!(vItem.getWeight() + this.aPlayer.getInvWeight() <= this.aPlayer.getMaxWeight())){ // Vérification si
        this.aGui.println("Tu n as plus de place dans ton inventaire");
        return;
    }// if ()
    this.aPlayer.take(vItem);
    this.aGui.println("Tu as l item: " + vItem.getName() + "\n");
    this.aGui.println(vItem.getItemDescription());
    this.aGui.println(this.aPlayer.getInvItems().getItemString()); // Affichage du contenu de l'inventaire
}// take ()
```

```java
/**
 * Procédure qui permet au joueur de déposer un item
 *
 * @param pCommand Commande de l'utilisateur transmise
 */
private void drop(final Command pCommand) {
    if (pCommand.hasSecondWord()) {
        Item vItem = this.aPlayer.getInvItems().getItem(pCommand.getSecondWord());
        if (vItem != null) { // Vérification de l'absence de l'item dans l'inventaire
            this.aPlayer.drop(vItem);
            this.aGui.println("Tu as laisse l'item : " + vItem.getName());
        } else {
            this.aGui.println("Tu n'as pas cet item dans ton inventaire !");
        }
    } else {
        this.aGui.println("Tu veux laisser quoi ?");
    }
}
```

Ex: 7.33

Pour ajouter une nouvelle commande, on créer une nouvelle méthode dans une class CommandWords. on la rajoute ensuite dans la procédure "interpretCommand()" pour qu'elle puisse être utilisée.

```java
private void inventaire (final Command pCommand)
{
    if (pCommand.hasSecondWord()){ // Vérification de la présence d'un second mot
        this.aGui.println("Tape inventaire si tu veux le voir !");
        return;
    }// if ()

    int vInventoryRound = (int)Math.round((this.aPlayer.getInvWeight()));
    this.aGui.println("Ton inventaire est rempli a " + vInventoryRound + "%"); //
}
```

```java
/**
 * Constructor - initialise the command words.
 */
public CommandWords()
{
    this.aValidCommands = new String[10];
    this.aValidCommands[0] = "go";
    this.aValidCommands[1] = "aide";
    this.aValidCommands[2] = "quitter";
    this.aValidCommands[3] = "regarder";
    this.aValidCommands[4] = "manger";
    this.aValidCommands[5] = "retour";
    this.aValidCommands[6] = "test";
    this.aValidCommands[7] = "prendre";
    this.aValidCommands[8] = "laisser";
    this.aValidCommands[9] = "inventaire";
} // CommandWords()
```

```java
} else if(vCommandWord.equals("inventaire")){
    this.inventaire(vCommand);
}
```

Ex: 7.34

Les modifications apportées consistent à ajouter deux modificateurs dans la classe "Player" pour permettre la modification du poids maximum et du poids des objets. Dans la méthode "interpretCommand" de la classe "GameEngine", la commande est maintenant envoyée directement à la procédure correspondante. La méthode "eat" a été modifiée pour permettre de manger la pomme en vérifiant les conditions nécessaires, retirer le poids de la pomme et augmenter le poids maximum.

```java
public void setMaxWeight (final int pWeight){
    this.aMaxWeight = pWeight;
}

/**
 * Modificateur du poids de l'inventaire du joueur
 *
 * @param pWeight Poids que l'on souhaite soustraire du poids actuel de l'inventaire
 */
public void removeInvWeight (final int pWeight){
    this.aInvWeight -= pWeight;
}// removeInventoryWeight ()
}
```

```java
else if ( vCommandWord.equals( "manger" ) )
    this.eat(vCommand);
```

```java
 * pour afficher la commande eat
 */
private void eat (final Command pCommand)
{
    if (!pCommand.hasSecondWord()){ // Vérification de la présence d'un second mot
        this.aGui.println("Que veux tu manger?");
        return;
    }

    String vFood = pCommand.getSecondWord();
    if (vFood.equals("pomme")){ // Vérification si le second mot est bien "cookie"
        if (this.aPlayer.getInvItems().getItem("pomme") == null){
            this.aGui.println("Tu n as pas de pomme dans ton inventaire !");
            return;
        }
    }
    // Si l'on arrive jusque ici alors le joueur veut manger la graine et l'a dans son inventaire
    this.aPlayer.removeInvWeight(this.aPlayer.getInvItems().getItem("pomme").getWeight());
    this.aPlayer.setMaxWeight(10); // Augmentation du poids possible transportable
    this.aPlayer.getInvItems().removeItem("pomme"); // Suppression de la graine de l'inventaire
    this.aGui.println("Tu as mange la pomme, tu as donc augmente la capacite de ton inventaire de 10 % !");
}
```

Ex: 7.42

On ajoute un élément au jeu qui consiste à compter le nombre de déplacement et lorsque le joueur atteint le max il perdra.

En même temps je vais permettre a certain de mes objets d'ajouter ou d'enlever du « temp ».
Dans Player :

```
private int aMoves;             // Nombre de déplacements maximum avant de perdre
/**
 * Constructeur d'objets de classe Player
 * @param pPseudo Pseudonyme du joueur
 * @param pCurrentRoom Pièce de départ du joueur
 */
public Player(final String pPseudo, final Room pCurrentRoom, final int pMoves) {
    this.aPseudo = pPseudo;
    this.aMaxWeight = 100;
    this.aPreceRooms = new Stack<Room>();
    this.aInvItems = new ItemList("Ton inventaire");
    this.aCurrentRoom = pCurrentRoom;
    this.aInvWeight = 0;
    this.aMoves = pMoves;
}
```

```
/**
 * Accesseur du nombre de déplacements restants
 *
 * @return le nombre de déplacements avant de perdre
 */
public int getMovesRemaining ()
{
    return this.aMoves;
}

/**
 * Modificateur du nombre de déplacements restants
 *
 */
public void addMoves (final int pAdd)
{
    this.aMoves += pAdd;
}

/**
 * Modificateur du nombre de déplacements restants
 *
 */
public void subMoves (final int pSub)
{
    this.aMoves -= pSub;
}
```

```
public void goRoom(final String pDirection) {
    Room vNextRoom = this.aCurrentRoom.getExit(pDire
    this.aPreceRooms.push(this.aCurrentRoom);
    this.aCurrentRoom = vNextRoom;
    this.aMoves -= 1;
}

/**
 * Permet de revenir sur ses pas
 */
public void back() {
    this.aCurrentRoom = this.aPreceRooms.pop();
    this.aMoves -= 1;
}
```

Dans GameEngine :

```
private void time (){
    if (this.aPlayer.getMovesRemaining() == 0){ // Vérifie si le joueur a des déplacements restants
        this.aGui.println("Le minuteur est arrive a 0, la bombe a explose !");
        this.endGame();
    }
}
```

Dans eat() :

```
if (vFood.equals("piment")){
    if (this.aPlayer.getInvItems().getItem("piment") == null){
        this.aGui.println("Tu n as pas de piment dans ton inventaire !");
        return;
    }else {this.aPlayer.removeInvWeight(this.aPlayer.getInvItems().getItem("piment").getWeight());
        this.aPlayer.addMoves(25);// Augmentation du nbr de mvt
        this.aPlayer.getInvItems().removeItem("piment");
        this.aGui.println("Tu as mange le piment, tu as donc augmente ton nombre de mouvement!");

    }
}

if (vFood.equals("pomme")){
    if (this.aPlayer.getInvItems().getItem("pomme") == null){
        this.aGui.println("Tu n as pas de pomme dans ton inventaire !");
        return;
    }else {this.aPlayer.removeInvWeight(this.aPlayer.getInvItems().getItem("pomme").getWeight());
        this.aPlayer.subMoves(10);// Diminution du nbr de mvt
        this.aPlayer.getInvItems().removeItem("pomme");
        this.aGui.println("Tu as mange la pomme, tu as donc diminue ton nombre de mouvement!");

    }
}
```

Ex: 7.42.2

Je décide de garder l'IHM actuelle.

Ex: 7.43

Pour empêcher la possibilité de revenir en arrière en utilisant la commande "back", on peut ajouter une condition supplémentaire à la fonction qui gère cette commande. On introduit une méthode booléenne appelée "isExit()" dans la classe "Room" qui vérifie si la pièce actuelle a effectivement la pièce cible comme sortie.

```
/**
 * Fonction qui vérifie si une pièce est accessible
 * @return True si la pièce est une sortie possible, False dans le cas contraire
 */
public boolean isExit (final Room pRoom)
{
    if (this.aExits.containsValue(pRoom)){
        return true;
    }
    return false;
}
```

```java
/**
 * Permet de revenir sur ses pas
 */
public boolean back() {
    if (this.aCurrentRoom.isExit(this.aPreceRooms.peek())){ // Vérifie si la pièce est dans
        this.aCurrentRoom = this.aPreceRooms.pop();
        this.aMoves -= 1;
        return true;
    }
    return false;
}
```

Dans GameEngine :

```java
// Vérifie si la pièce actuelle est accessible (sinon il s'agit d'une trappe ou bien d'une téléportat
if (!this.aPlayer.back()){
    this.aGui.println("Vous ne pouvez pas revenir en arriere car vous avez deja utilise ce passage.")
}
printLocationInfo();
this.time();
```

Ex:7.44

Afin de permettre le fonctionnement de cet élément, il sera nécessaire de créer une classe spécifique pour celui-ci, étant donné qu'il possède un comportement particulier. De plus, deux commandes devront également être créées pour le manipuler. En ce qui concerne la classe, nous allons créer la classe "Beamer" avec un attribut représentant la pièce à stocker. Étant donné que le beamer est un type d'élément, nous ferons hériter la classe "Beamer" de la classe "Item".

```java
public class Beamer extends Item
{
    private Room aChargedRoom; // Pièce dans laquelle le beamer a été chargé

    /**
     * Constructeur naturel
     *
     * @param pName Nom du beamer
     * @param pDescription Description du beamer
     * @param pWeight Poids du beamer
     */
    public Beamer(final String pName, final String pDescription, final int pWeight)
    {
        super(pName, pDescription, pWeight);
    }// Beamer ()

    /**
     * Accesseur de la pièce stockée dans le beamer
     *
     * @return Pièce stockée
     */
    public Room getChargedRoom ()
    {
        return this.aChargedRoom;
    }// getChargedRoom ()

    /**
     * Modificateur de la pièce chargée dans le beamer
     *
     * @param pRoom Pièce dans laquelle on charge le beamer
     */
    public void setChargedRoom (final Room pRoom)
    {
        this.aChargedRoom = pRoom;
    }// setChargedRoom ()

    /**
     * Fonction booléenne qui vérifie si le beamer est chargé
     *
     * @return True si une pièce est stockée dans le beamer, False dans le cas contraire
     */
    public boolean isCharged ()
    {
        return this.aChargedRoom != null;
    }// isCharged ()
}// Beamer
```

```java
/**
 * Procédure qui charge le beamer
 *
 * @param pCommand Commande de l'utilisateur transmise
 */
private void charge(Command pCommand) {
    if (!pCommand.hasSecondWord()) {
        this.aGui.println("Que veux-tu charger ?");
        return;
    }

    String vSecondWord = pCommand.getSecondWord();
    if (!vSecondWord.equals("beamer")) {
        this.aGui.println("Je ne peux pas charger cet objet.");
        return;
    }

    Item vItem = this.aPlayer.getInvItems().getItem(vSecondWord);
    if (vItem == null || !(vItem instanceof Beamer)) {
        this.aGui.println("Tu n as pas le beamer dans ton inventaire !");
        return;
    }

    Beamer vBeamer = (Beamer) vItem;
    this.aPlayer.charge(vBeamer);
    this.aGui.println("Ton beamer est charge !");
    this.time();
}
```

```java
 * Procédure qui décharge le beamer
 *
 * @param pCommand Commande de l'utilisateur transmise
 */
private void fire(Command pCommand) {
    if (!pCommand.hasSecondWord()) {
        this.aGui.println("Que veux-tu activer ?");
        return;
    }

    String vSecondWord = pCommand.getSecondWord();
    if (!vSecondWord.equals("beamer")) {
        this.aGui.println("Je ne peux pas activer cet objet !");
        return;
    }

    Item vItem = this.aPlayer.getInvItems().getItem(vSecondWord);
    if (vItem == null) {
        this.aGui.println("Tu n'as pas le beamer dans ton inventaire !");
        return;
    }

    if (!(vItem instanceof Beamer)) {
        this.aGui.println("Cet objet n'est pas un beamer !");
        return;
    }

    Beamer vBeamer = (Beamer) vItem;
    if (!vBeamer.isCharged()) {
        this.aGui.println("Tu dois le charger avant de l'utiliser !");
        return;
    }

    this.aPlayer.fire(vBeamer);
    this.aGui.println("Ton beamer a été déclenché !");
    printLocationInfo(); // Affiche les informations sur la pièce
}
```

```java
/**
 * Procédure qui charge le beamer
 *
 * @param pBeamer Item beamer transmit par GameEngine
 */
public void charge (final Beamer pBeamer)
{
    pBeamer.setChargedRoom(this.aCurrentRoom); // Charge la pièce actuelle dans le beamer
    this.aMoves -= 1; // Retire un point de déplacement
}// charge ()

/**
 * Procédure qui utilise le beamer
 *
 * @param pBeamer Item beamer transmit par GameEngine
 */
public void fire (final Beamer pBeamer)
{
    this.aPreceRooms.push(this.aCurrentRoom); // Ajoute la pièce actuelle aux anciennes pièces
    this.aCurrentRoom = pBeamer.getChargedRoom(); // Remplace la pièce actuelle par celle stockée dans le b
    this.aInvItems.removeItem(pBeamer.getName()); // Retire le beamer de l'inventaire
    this.aInvWeight -= pBeamer.getWeight(); // Retire le poids du beamer du poids total
    this.aMoves -= 1; // Retire un point de déplacement
}
```

J’ai ajouté des modifications pour faire apparaitre des énigmes.

III. Mode d’emploi

Pour commencer il faut créer un objet game, puis suivre les instructions données au début du jeu.

IV. Déclaration anti-plagiat

Tout emprunt direct ou indirect à des sources externes a été clairement cité afin d'éviter tout plagiat.