Rapport IPO - Projet Zuul

Sommaire:

Conception du Jeu

7.1.1 - Phrase thème

Dans la ville de Tokyo en l'an 2304 un pilote de robot a pour mission de défendre sa ville d'un danger imminent.

7.3 - Scénario du jeu

L'histoire commence donc dans la chambre de notre protagoniste Shinji Akira, un pilote de mecha (robot humanoïde de grande taille, environ 80 mètres de haut). Il reçoit un message de la base qui le sollicite pour lui donner une mission.

Il doit donc se rendre dans le quartier de Shibuya ou se trouve un ascenseur qui lui permet de descendre au Bunker de la base secrète militaire.

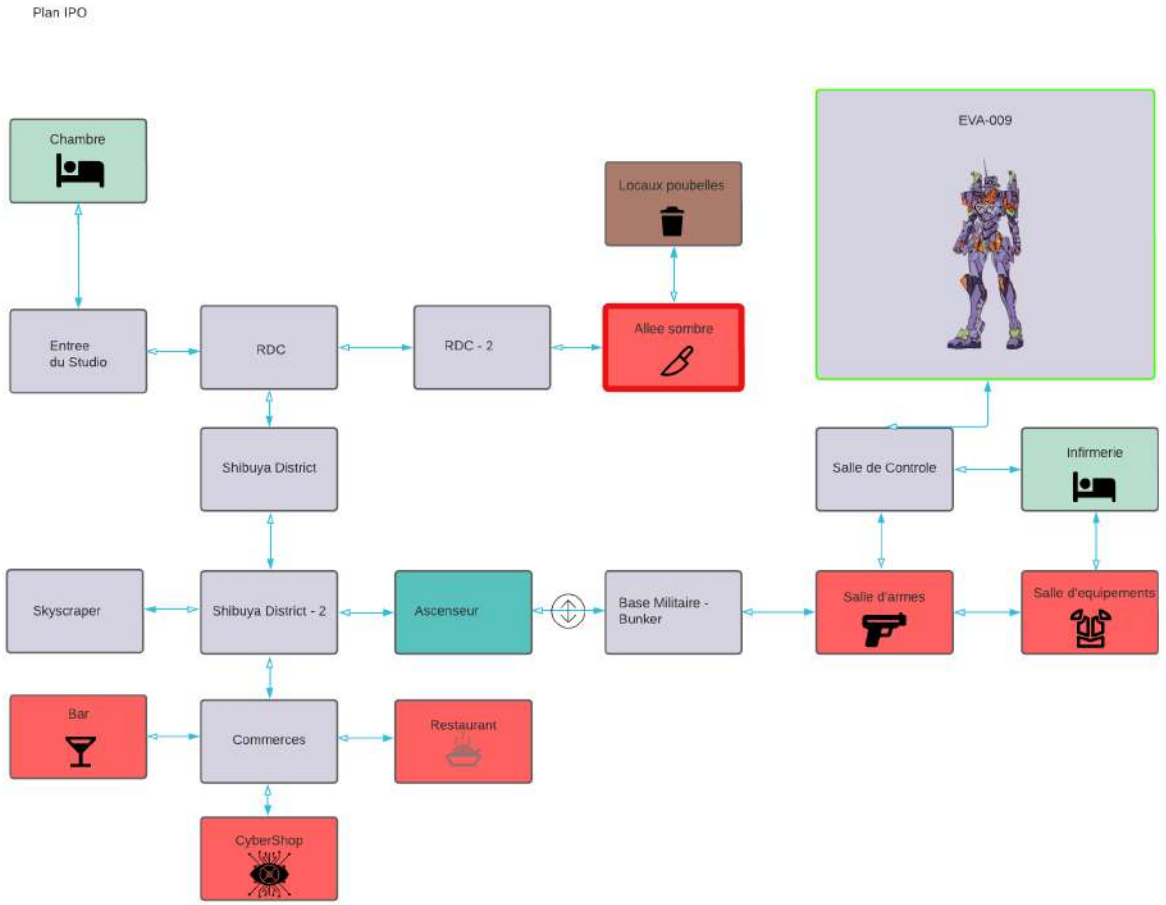
Pour cela il doit aller dans un bar rencontrer son informant qui lui donnerait des indications pour trouver une clé pour accéder à l'ascenseur.

Cette clé se trouve dans le Restaurant en face du Bar. On peut y manger pour regagner ses forces.

Par la suite il se rendra dans la base et il trouvera des militaires à qui parler qui le préparent aussi bien physiquement que mentalement à se battre contre le danger. (il devra réussir divers défi, ou répondre à des questions pour récupérer des objets qui l'aideront à réparer son robot)

Le joueur gagne lorsqu'il réussit à vaincre le danger à l'aide du mecha.

7.3.2 - Plan du Jeu



Réponses aux exercices numérotés

7.4

(v1 et vref supprimer)

7.5 - Procédure printLocationInfo()

Il faut créer cette procédure pour pouvoir éviter la duplication de code qui est présente dans les méthodes goRoom() et dans printWeclome()

private void printLocationInfo();
{

```
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.aWestExit != null){
    System.out.print("west");
}
if(this.aCurrentRoom.aSouthExit != null){
    System.out.print("south");
}
System.out.println();
```

}

7.6 - Méthode getExit()

Création de la méthode getExit() :

```
/**
 * methode getExit();
 */
public Room getExit(final String vDirection){
    if(vDirection.equals("north")){
        return this.aNorthExit;
    }
    if(vDirection.equals("east")){
        return this.aEastExit;
    }
    if(vDirection.equals("west")){
        return this.aWestExit;
    }
    if(vDirection.equals("south")){
        return this.aSouthExit;
    }
    return null;
}
```

Il faut donc changer la méthode goRoom() :

```
//## Apres l'exercice 7.6 ##
Room vNextRoom = aCurrentRoom.getExit(vDirection);
```

7.7 - Méthode getExitString() & changement de printLocationInfo()

```
/**
 * getExitString
 */
public String getExitString(){
    String vAll = "";
    if(this.aNorthExit != null){
        vAll = vAll+"north";
    }
    if(this.aNorthExit != null){
        vAll = vAll+"east";
    }
    if(this.aNorthExit != null){
        vAll = vAll+"west";
    }
    if(this.aNorthExit != null){
        vAll = vAll+"south";
    }
    return vAll;
}
```

Il est logique de demander a *Room* de produire les informations sur ses sorties car *Room* possède les attributs *aNorthExit* etc… Au contraire, la classe *Game* ne pouvait produire des informations sur les sorties que si les attributs étaient *public.*

Changements sur la méthode printLocationInfo() :

```java
/**
 * printLocation Methode
 */
private void printLocationInfo(){
    System.out.println(this.aCurrentRoom.getDescription());
    System.out.print("Exits :");
    this.aCurrentRoom.getExitString();
    System.out.println();
    }
```

7.8 - Ajout de Hashmap dans le code

Modification de getExitString() après l'ajout de HashMap :

```java
/**
 * getExitString
 */
public String getExitString(){
    String vAll = "";
    if(this.getExit("north") != null){
        vAll = vAll+"north";
    }
    if(this.getExit("east")!= null){
        vAll = vAll+"east";
    }
    if(this.getExit("west")!= null){
        vAll = vAll+"west";
    }
    if(this.getExit("south") != null){
        vAll = vAll+"south";
    }
    return vAll;
}
```

La méthode setExits(nord,est,ouest,sud) n'est plus du tout d'actualité à cause du changement en HashMap. Il faut donc la changer en setExit au singulier pour définir les sorties une par une. Il faut donc changer le classe Game en conséquence :

```
//## 2eme Partie (utiliser setExit)
//chambre
vChambre.setExit("south", vEntree);
//Entree
vEntree.setExit("east",vRDC);
vEntree.setExit("north",vChambre);
//RDC
vRDC.setExit("west",vEntree);
vRDC.setExit("east",vRDC2);
vRDC.setExit("south",vShibuyaDis);
//RDC2
vRDC2.setExit("east",vDarkAlley);
vRDC2.setExit("west",vRDC);
//Dark Alley
vDarkAlley.setExit("north",vLocauxPoubelles);
vDarkAlley.setExit("west",vRDC2);
//Locaux Poubelles
vLocauxPoubelles.setExit("south",vDarkAlley);
//Bar
vBar.setExit("east",vCommerces);
//Commerces
vCommerces.setExit("north",vShibuyaDis2);
vCommerces.setExit("east",vRestaurant);
vCommerces.setExit("west",vBar);
vCommerces.setExit("south",vCyberShop);
//Shibuya District
vShibuyaDis.setExit("north",vRDC);
vShibuyaDis.setExit("south",vShibuyaDis2);
//Shibuya District 2
vShibuyaDis2.setExit("east", vElevator);
vShibuyaDis2.setExit("north",vShibuyaDis);
vShibuyaDis2.setExit("west",vSkyScraper);
//SkyScraper
vSkyScraper.setExit("east",vShibuyaDis2);
//Elevator
vElevator.setExit("west",vShibuyaDis2);
vElevator.setExit("down",vBunker);
//Bunker
vBunker.setExit("up",vElevator);
vBunker.setExit("east",vWeaponRoom);
//Control Room
vControlRoom.setExit("south", vWeaponRoom);
vControlRoom.setExit("east", vInfirmerie);
//Robot Room
vRobotRoom.setExit("south",vControlRoom);
//vWeaponRoom
vWeaponRoom.setExit("west",vBunker);
vWeaponRoom.setExit("north",vControlRoom);
vWeaponRoom.setExit("east",vEquipRoom);
//vEquipRoom
vEquipRoom.setExit("west",vWeaponRoom);
vEquipRoom.setExit("north",vInfirmerie);
```

7.9 - Changement de getExitString()

On change la méthode getExitString comme ci-dessous grâce à l'import de Set et Iterator :

```
import java.util.HashMap;
import java.util.Set;
import java.util.Iterator;
```

```
/**
 * getExitString
 */
public String getExitString(){
    String vAll = "Exits:";
    Set<String> vExits = aExits.keySet();
    for(String vExit : vExits) {
      vAll += " "+ aExits;
    }
    return vAll;
}
```

7.10 - Documentation JavaDoc

La classe Game comprend beaucoup moins de méthodes que la classe Room car son but est de générer un Game alors que la classe Room doit créer une pièce avec beaucoup plus de paramètres à prendre en compte. Aussi beaucoup de méthodes sont privées dans la classe Game alors que dans Room il y en a plus qui sont public car elles sont utilisées par les autres classes.

7.11

```
/**
 * Renvoie une description plus detaillee de cette piece
 *
 * @return Une description de la piece avec ses sorties
 */
public String getLongDescription(){
    return "You are "+ aDescription + ".\n" + getExitString();
}
```

7.14.2 & 7.14.3 - look()

Dans la classe CommandWords on ajoute le mot “look” dans aValidCommands:

```
public class CommandWords
{
    // a constant array that will hold all valid command words
    private final String[] aValidCommands = {
        "go","quit","help","look","eat"
    };
```

Dans la méthode processCommand on ajoute :

```
else if(pCommand.getCommandWord().equals("look")){
    look(pCommand);
    return false;
}
```

Ensuite on créer la méthode look() et on vérifie s’il y a un deuxième mot:

```
/**
 * methode look()
 */
private boolean look(final Command pLookWhat){
    if(pLookWhat.hasSecondWord()){
     System.out.println("I dont know how to look at something in particular yet");
     return false;
    }
    else{
     printLocationInfo();
     return true;
    }
}
```

7.15 - eat()

Dans la classe CommandWords on ajoute le mot “eat” dans aValidCommands :

```
public class CommandWords
{
    // a constant array that will hold all valid command words
    private final String[] aValidCommands = {
        "go","quit","help","look","eat"
    };
```

Dans la méthode processCommand on ajoute :

```
}
else if(pCommand.getCommandWord().equals("eat")){
    eat();
    return false;
}
```

Ensuite on créer la méthode eat() :

```
/**
 * methode eat()
 */

private void eat(){
    System.out.println("You have eaten now and you are not hungry anymore");
}
```

7.16

Dans la classe CommandWords :

```
/**
 * Print all valid commands to System.out
 */
public void showAll(){
    for(String command : aValidCommands){
        System.out.println(command + " ");

    }
    System.out.println();
}//showAll()
```

Dans la classe Parser pour ne pas a avoir a relier Game et CommandWords

```
/**
* Print out a list of valid Command Words
*/
public void showCommands(){
    this.aValidCommands.showAll();

}
```

7.18

Changement de la méthode showAll(); :

```
/**
 * Print all valid commands to System.out
 */
public String getCommandList(){
    String vS = "";
    for(String command : aValidCommands){
        vS = vS + command + "  ";

    }
    return vS;
}//showAll()
```

il faut donc changer dans la classe game :

```
/**
 * methode printHelp()
 */
private void printHelp()
{
  System.out.println("There has been an attack on Tokyo.");
  System.out.println("You have as a pilot a duty to protect the citizens");
  System.out.println("                                          ");
  System.out.println("Your command words are:");
  System.out.println(this.aParser.showCommands());


}//printHelp()
```

7.18.4

Titre du Jeu : EVA-009 : Revolution of 2304

```
/**
* procedure printWelcome()
* qui
*/
private void printWelcome()
{
    System.out.println("Hello to you fellow pilot !");
    System.out.println("Welcome to EVA-809 : Revolution of 2304");
    System.out.println("No time to lose you are called here today to help defend the city of the citizens of Tokyo");
    System.out.println("Type 'help' if you need help from your AI guide.");
    System.out.println("                                ');
    //System.out.println(this.aCurrentRoom.getLongDescription());
    System.out.println("Exits : south");
}//printWelcome()
```

7.18.6

Ajout d'une nouvelle interface graphique avec zuul-with-images

7.18.8

```
private JButton     aButtonN;
private JButton     aButtonS;
```

```
this.aButtonS = new JButton("South");


this.aButtonN = new JButton("North");
```

```
vPanel.add( this.aButtonS , BorderLayout.EAST);
vPanel.add( this.aButtonN , BorderLayout.WEST);
```

```
this.aButtonN.addActionListener(this);
this.aButtonS.addActionListener(this);
```

7.19.2
dans UserInterface

```
/**
 * Show an image file in the interface.
 */
public void showImage( final String pImageName )
{
    String vImagePath = "Images/" + pImageName + "/"; // to change the directory
    URL vImageURL = this.getClass().getClassLoader().getResource( vImagePath );
    if ( vImageURL == null )
        System.out.println( "Image not found : " + vImagePath );
    else {
        ImageIcon vIcon = new ImageIcon( vImageURL );
        this.aImage.setIcon( new ImageIcon( vIcon.getImage().getScaledInstance(640,400,java.awt.Image.SCALE_SMOOTH) ) );
        this.aMyFrame.pack();
    }
} // showImage(.)
```

```java
/**
 * Actionlistener interface for entry textfield.
 */
@Override public void actionPerformed( final ActionEvent pE )
{
    if(pE.getSource() == this.aButtonN){
        this.aEngine.interpretCommand("go north");
    }
    else if(pE.getSource() == this.aButtonS){
        this.aEngine.interpretCommand("go south");
    }
    else{
        this.processCommand();
    }
} // actionPerformed(.)
```

7.20 Item

La classe Item :

```java
public class Item
{
    private String aNom;
    private String aDescription;
    private int aPoid;

    public Item(final String pNom, final String pDescription, final int pPoid){
        this.aNom = pNom;
        this.aDescription = pDescription;
        this.aPoid = pPoid;
    }

    public String getNomItem(){
        return this.aNom;
    }

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

    public int getPoidItem(){
        return this.aPoid;
    }

    public String getLongDescriptionItem(){
        return "This " + this.aNom +" is a " +getDescriptionItem() + " it weighs " + this.aPoid;
    }
```

Dans la classe Room on ajoute donc :

```
private Item aItem;
```

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

public String getItemString(){
    if(this.aItem != null){
        return this.aItem.getLongDescriptionItem();
    }
    else{
        return "No item here";
    }
}
```

Dans GameEngine :

```
vChambre.setExit("south", vEntree);
Item vHelmet = new Item( "helmet","a sturdy helmet",20);
vChambre.setItem(vHelmet);
```

7.21

C’est la classe Item qui doit produire les infos.
C’est la classe Item qui doit produire la String de l’Item.
C’est la classe GameEngine qui doit l’afficher.
Aucun changement n’a été fait pour l’instant.

7.22

```
Item vPill = new Item("pill","a pill that increases strength", 0);
vCyberShop.getItemList().addItem("pill",vPill);
```

```
//## Objets
Item vJacket = new Item("jacket","faithful leather jacket",10);
Item vJacketB = new Item("B jacket","faithful leather black jacket",10);
Item vPants = new Item("pants","faithful leather pants",15);
vEquipRoom.setItems("B jacket",vJacketB);
vEquipRoom.setItems("pants",vPants);
vChambre.setItems("jacket",vJacket);
vChambre.addItem("B jacket",vJacketB);
```

7.23 back

```
private Room          aPreviousRoom = null;
```

```
}
else if( vCommandWord.equals("back")){
    if ( vCommand.hasSecondWord() ){
        this.aGui.println( "Back where?" );
        return;
    }
    else{
        if(this.aPreviousRoom != null){
            this.aCurrentRoom = this.aPreviousRoom;
            this.aGui.println( this.aCurrentRoom.getLongDescription() );
            if ( this.aCurrentRoom.getImageName() != null ){
            this.aGui.showImage( this.aCurrentRoom.getImageName() );
            }
        }
    }
}
```

```
if ( vNextRoom == null )
    this.aGui.println( "There is no door!" );
else {
    this.aPreviousRoom = this.aCurrentRoom;
    this.aCurrentRoom = vNextRoom;
    this.aGui.println( this.aCurrentRoom.getLongDescription() );
    if ( this.aCurrentRoom.getImageName() != null )
        this.aGui.showImage( this.aCurrentRoom.getImageName() );
}
```

NOUVEAU DESIGN APRÈS 7.44 **Beamer** pour cause de bugs

```java
/**
 * back() : utile pour la fonction interpretCommand()
 */
public boolean canBack(){
    if(this.aPreviousRooms.empty()){
        return false;
    }
    else{
        this.aPreviousRooms.pop();
        this.aPreviousRoom = (Room)(this.aPreviousRooms.peek());
        this.aPreviousRooms.pop();
        return true;
    }
}//back()
```

```java
case 5 ://
    if ( vCommand.hasSecondWord() ){
        this.aGui.println( "Back where?" );
        break;
    }
    else{
        if(this.aPlayer.canBack()){
            if(this.aPlayer.getPreviousRoom() != null){
                if(this.aPlayer.getCurrentRoom().isExit(this.aPlayer.getPreviousRoom())){
                    this.aGui.println("You are trapped");
                }
                else{
                    //Stack vPreviousRooms = this.aPlayer.getPreviousRooms();
                    this.aPlayer.setCurrentRoom(this.aPlayer.getPreviousRoom());
                    //this.aPlayer.setPreviousRooms(vPreviousRooms);
                    this.aTime += 1;
                    this.aGui.println( this.aPlayer.getCurrentRoom().getLongDescription() );
                    if ( this.aPlayer.getCurrentRoom().getImageName() != null ){
                        this.aGui.showImage( this.aPlayer.getCurrentRoom().getImageName() );
                        break;
                    }
                }
            }
        }
        else{
            this.aGui.println( "Can't back here" );
            break;
        }
    }
```

7.26

dans goRoom()

```java
this.aPreviousRooms.push(this.aCurrentRoom);
```

nouvelle methode back

```
public boolean back(){
    if(this.aPreviousRooms.empty()){
        return false;
    }
    else{
        this.aPreviousRoom = (Room)(this.aPreviousRooms.peek());
        this.aPreviousRooms.pop();
        return true;
    }
}
```

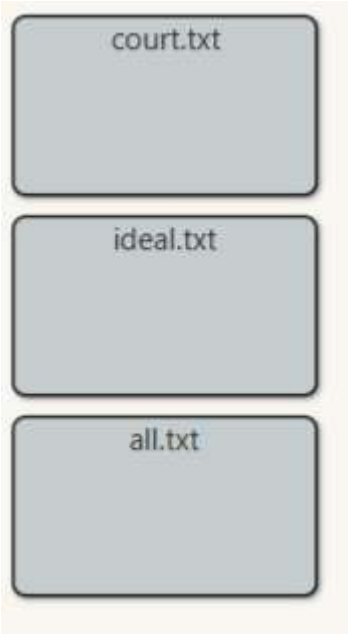
implémentation de back() dans interpretCommand()

```
else if( vCommandWord.equals("back")){
    if ( vCommand.hasSecondWord() ){
        this.aGui.println( "Back where?" );
        return;
    }
    else{
        if(back()){
            if(this.aPreviousRoom != null){
            this.aCurrentRoom = this.aPreviousRoom;
            this.aGui.println( this.aCurrentRoom.getLongDescription() );
            if ( this.aCurrentRoom.getImageName() != null ){
            this.aGui.showImage( this.aCurrentRoom.getImageName() );
            }
        }
        }
        else{
          this.aGui.println( "Can't back here" );
        }
    }
}
```

7.28.1 - test

après avoir ajouter un mot commande test dans CommandWords (* après implémentation du switch dans interpretCommand)

```
case 4 :
    if(!vCommand.hasSecondWord()){
        this.aGui.println("test what ?");
        return;
    }
    String vMotFichier = ""+ vCommand.getSecondWord()+".txt";
    Scanner vScan;
    try {
       vScan = new Scanner(new File(vMotFichier));
       while(vScan.hasNextLine()){
       String vLigne = vScan.nextLine();
       interpretCommand(vLigne);
       }
    }
    catch(final FileNotFoundException pFNFE ){
        this.aGui.println("File not found");
    }
    break;
```



7.29 Classe Player :

Après avoir oublié de prendre une capture d'écran de la classe Player apres sa creation je peux que vous montrer

```java
import java.util.Stack;
import java.util.HashMap;
import java.util.Set;

/**
 * Write a description of class Player here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player
{
    private Room aCurrentRoom;
    private String aUserName;
    private Room aPreviousRoom;
    private Stack  aPreviousRooms;
    private ItemList aInventory;
    private int aMaxPoids;
    private int aCurrentPoids;

    /**
     *  constructeur de player
     */
    public Player(final Room pCurrentRoom, final String pUserName){
        this.aCurrentRoom =pCurrentRoom;
        this.aUserName = pUserName;
        this.aPreviousRooms = new Stack<Room>();
        //# apres ItemList
        this.aInventory = new ItemList("inventory");
        this.aMaxPoids = 100;
        this.aCurrentPoids = 0;
    }

    /**
     * getter : retourne la room ou se situe le Joueur
     */
    public Room getCurrentRoom(){
        return this.aCurrentRoom;
    }
```

```java

    /**
     * getter : retourne le nom du joueur
     */
    public String getUserName(){
        return this.aUserName;
    }

    /**
     * getter : retourne la Stack des PreviousrRooms
     */
    public Stack getPreviousRooms(){
        return this.aPreviousRooms;
    }

    /**
     * getter : retourne la PreviousRoom
     */
    public Room getPreviousRoom(){
        return this.aPreviousRoom;
    }

    /**
     * getter : retourne le poids max du joueur
     */
    public int getMaxPoids(){
        return this.aMaxPoids;
    }

    /**
     * getter : retourne le poids courant du joueur
     */
    public int getCurrentPoids(){
        return this.aCurrentPoids;
    }
```

```java
/**
 *  un modificateurs du poids courant
 */
public void setCurrentPoids(final int pInt){
    this.aCurrentPoids = pInt;
}

  /**
 * Try to go to one direction. If there is an exit, enter the new
 * room, otherwise print an error message.
 */
public void goRoom( final String pDirection )
{
    // Try to leave current room.
    Room vNextRoom = this.aCurrentRoom.getExit( pDirection );
    this.aPreviousRooms.push(this.aCurrentRoom);
    this.aCurrentRoom = vNextRoom;
}//goRoom()

/**
 * back() : utile pour la fonction interpretCommand()
 */
public boolean back(){
    if(this.aPreviousRooms.empty()){
        return false;
    }
    else{
        this.aPreviousRoom = (Room)(this.aPreviousRooms.peek());
        this.aPreviousRooms.pop();
        return true;
    }
}//back()

/**
 *  set la Stack de PreviousRooms
 */
public void setPreviousRooms(final Stack pStack){
    this.aPreviousRooms = pStack;
}
```

```java
/**
 *  getter de l'inventaire
 */
public ItemList getInventory(){
    return this.aInventory;
}

/**
 *  methode take() dans GameEngine
 */
public void take(final String pItem){
    this.aInventory.addItem(pItem,this.aInventory.getItem(pItem));
    Item vItem =  null;
    if(this.aInventory.getItem(pItem) != vItem){
        this.aCurrentPoids += this.aInventory.getItem(pItem).getPoidsItem();
    }
    this.aCurrentRoom.getItemList().removeItem(pItem);
}

/**
 * methode drop() utile dans GameEngine
 */
public void drop(final String pItem){
    this.aCurrentRoom.getItemList().addItem(pItem,this.aInventory.getItem(pItem));
    if(this.aInventory.getItem(pItem) != null){
        this.aCurrentPoids -= this.aInventory.getItem(pItem).getPoidsItem();
    }
    this.aInventory.removeItem(pItem);
}
```

```java
/**
 * methode isInInventory regarde si l'item est dans l'inventaire du joueur
 */
public boolean isInInventory(final Item pItem){
    if(this.aInventory.getItems().containsValue(pItem)){
        return true;
    }
    else{
        return false;
    }
}

/**
 * methode modificateur setMaxPoids()
 */
public void setMaxPoids(final int pInt){
    this.aMaxPoids = pInt;
}

/**
  * methode poidsTotal()
  */
public int poidsTotal(){
    int vPoids = this.aCurrentPoids;
    Set<String> keys = this.aInventory.getItems().keySet();
    for(String item : keys) {
        vPoids += this.aInventory.getItem(item).getPoidsItem();
    }
    return vPoids;
}

/**
* methode accesseur setInventory()
*/
public void setInventory(final ItemList vItemList){
    this.aInventory = vItemList;
}
```

7.30 take() & drop()

```java
import java.util.HashMap;
import java.util.Set;

/**
 * Write a description of class Player here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player
{
    private Room aCurrentRoom;
    private String aUserName;
    private Room aPreviousRoom;
    private Stack  aPreviousRooms;
    private HashMap<String, Item> aInventory;
    private int aMaxPoid;

    public Player(final Room pCurrentRoom, final String pUserName){
        this.aCurrentRoom =pCurrentRoom;
        this.aUserName = pUserName;
        this.aPreviousRooms = new Stack<Room>();
        this.aInventory = new HashMap<String,Item>();
        this.aMaxPoid = 100;
    }

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

    public String getUserName(){
        return this.aUserName;
    }

    public Stack getPreviousRooms(){
        return this.aPreviousRooms;
    }
```

```java
  /**
 * Try to go to one direction. If there is an exit, enter the new
 * room, otherwise print an error message.
 */
public void goRoom( final String pDirection )
{
    // Try to leave current room.
    Room vNextRoom = this.aCurrentRoom.getExit( pDirection );
    this.aPreviousRooms.push(this.aCurrentRoom);
    this.aCurrentRoom = vNextRoom;
}//goRoom()

/**
 * back() : utile pour la fonction interpretCommand()
 */
public boolean back(){
    if(this.aPreviousRooms.empty()){
        return false;
    }
    else{
        this.aPreviousRoom = (Room)(this.aPreviousRooms.peek());
        this.aPreviousRooms.pop();
        return true;
    }
}//back()

public void setPreviousRooms(final Stack pStack){
    this.aPreviousRooms = pStack;
}

public String getInventoryString(){
    String vAll = "Inventory :";
    Set<String> keys = this.aInventory.keySet();
    for(String inventory : keys) {
      vAll += " "+ inventory;
    }
    return vAll;
}
```

```
public Item addInventory(final String pItem){
    return this.aInventory.put(pItem,getItem(pItem));
}

public Item removeInventory(final String pItem){
    return this.aInventory.remove(pItem);
}

public void take(final String pItem){
    addInventory(pItem);
    this.aCurrentRoom.removeItem(pItem);
}

public Item getItem(final String pItem){
    return this.aInventory.get(pItem);
}

public void drop(final String pItem){
    this.aCurrentRoom.addItem(pItem,getItem(pItem));
    this.removeInventory(pItem);
}

public boolean isInRoom(final Item pItem){
    if(this.aCurrentRoom.getItems().containsValue(pItem)){
        return true;
    }
    else{
        return false;
    }
}

public boolean isInInventory(final Item pItem){
    if(this.aInventory.containsValue(pItem)){
        return true;
    }
    else{
        return false;
    }
}
```

```
public void setMaxPoid(final int pInt){
    this.aMaxPoid = pInt;
}
```

Après la classe Player j'ai remarquer que la méthode interpretCommand pouvait être changer en switch au lieu d'avoir des if/else/else if

```
public void interpretCommand( final String pCommandLine )
{
    this.aGui.println( "> " + pCommandLine );
    Command vCommand = this.aParser.getCommand( pCommandLine );
    String vCommand1 =""+vCommand.getCommandWord();
    String vCommand2 =""+vCommand.getSecondWord();

    int index = 0;
    if(vCommand1.equals("help")){
        index = 1;
    }
    else if(vCommand1.equals("quit")){
        index = 2;
    }
    else if(vCommand1.equals("go")){
        index = 3;
    }
    else if(vCommand1.equals("test")){
        index = 4;
    }
    else if(vCommand1.equals("back")){
        index = 5;
    }
    else if(vCommand1.equals("take")){
        index = 6;
    }
    else if(vCommand1.equals("drop")){
        index = 7;
    }
    else if(vCommand1.equals("bag")){
        index = 8;
    }
```

```
switch(index){
    case 1 :
        this.printHelp();
        break;
    case 2 :
        if ( vCommand.hasSecondWord() ){
            this.aGui.println( "Quit what?" );
            break;
        }
        else{
            this.endGame();
            break;
        }

    case 3 :
        if(!vCommand.hasSecondWord()){
            this.aGui.println("Go where?");
            break;
        }
        else{
            this.goRoom(vCommand);
            break;
        }
```

```
case 4 :
    if(!vCommand.hasSecondWord()){
        this.aGui.println("test what ?");
        return;
    }
    String vMotFichier = ""+ vCommand.getSecondWord()+".txt";
    Scanner vScan;
    try {
       vScan = new Scanner(new File(vMotFichier));
       while(vScan.hasNextLine()){
       String vLigne = vScan.nextLine();
       interpretCommand(vLigne);
       }
    }
    catch(final FileNotFoundException pFNFE ){
        this.aGui.println("File not found");
    }
    break;
case 5 :
    if ( vCommand.hasSecondWord() ){
        this.aGui.println( "Back where?" );
        break;
    }
    else{
        if(this.aPlayer.back()){
            if(this.aPlayer.getPreviousRoom() != null){
            Stack vPreviousRooms = this.aPlayer.getPreviousRooms();
            this.aPlayer = new Player(this.aPlayer.getPreviousRoom(),this.aPlayer.getUserName());
            this.aPlayer.setPreviousRooms(vPreviousRooms);
            if(this.aPlayer.back()){
                vPreviousRooms = this.aPlayer.getPreviousRooms();
                this.aPlayer = new Player(this.aPlayer.getPreviousRoom(),this.aPlayer.getUserName());
                this.aPlayer.setPreviousRooms(vPreviousRooms);
            }
            this.aGui.println( this.aPlayer.getCurrentRoom().getLongDescription() );
            }
            if ( this.aPlayer.getCurrentRoom().getImageName() != null ){
            this.aGui.showImage( this.aPlayer.getCurrentRoom().getImageName() );
            break;
            }
```

```
        }
        else{
          this.aGui.println( "Can't back here" );
          break;
            }
        }

    case 6 :
        if(this.aPlayer.isInRoom(this.aPlayer.getCurrentRoom().getItemRoom(vCommand2))){
            this.aPlayer.take(vCommand2);
            this.aGui.println(this.aPlayer.getInventoryString());
            this.aGui.println(this.aPlayer.getCurrentRoom().getItemString());
            if(this.aPlayer.getItem(vCommand2).getNomItem().equals("pill")){
                this.aPlayer.setMaxPoid(200);
                this.aGui.println("I feel way stronger ! (Max weight : 200)");
            }
        }
        else{
            this.aGui.println("not in room");
        }
        break;
    case 7:
        if(this.aPlayer.isInInventory(this.aPlayer.getItem(vCommand2))){
            this.aPlayer.drop(vCommand2);
            this.aGui.println(this.aPlayer.getInventoryString());
            this.aGui.println(this.aPlayer.getCurrentRoom().getItemString());
        }
        else{
            this.aGui.println("not in inventory");
        }
    case 8 :
        this.aGui.println(this.aPlayer.getInventoryString());
        //String vS = ""+this.aPlayer.getInventoryWeight();
        //this.aGui.println(vS);
        break;
    default :
        this.aGui.println( "I don't know what you mean..." );
        break;
}
'interpretCommand()
```

7.31 - Porter plusieurs Items

La HashMap permet le port de plusieurs objets de la part du joueur

7.31.1 - ItemList

TypeList sert à savoir si c'est un inventaire ou un liste d'objet dans une salle

```java
import java.util.HashMap;
import java.util.Set;
/**
 * Write a description of class ItemList here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class ItemList
{
    private HashMap <String, Item> aItems;
    private String aTypeList;

    public ItemList(final String pTypeList){
        this.aItems = new HashMap<String,Item>();
        this.aTypeList = pTypeList;
    }
```

On doit ensuite incorporer les différents getters éparpillés dans les classes Player et Room, puis créer un attribut dans chacune de ces classes de type ItemList puis utiliser les getters pour corriger les méthodes.

```java
/**
  * Getter de l'Item en String
  */
 public String getItemString(){
     if(this.aItems != null){
         if(this.aTypeList.equals("inventory")){
             String vAll = "Inventory :";
             Set<String> keys = this.aItems.keySet();
             for(String item : keys) {
               vAll += " "+ item;
             }
             return vAll;
         }
         else if(this.aTypeList.equals("room")){
             String vAll = "Items :";
             Set<String> keys = this.aItems.keySet();
             for(String item : keys) {
               vAll += " "+ item;
             }
             return vAll;
         }
         else{
             return "";
         }
     }
     else{
         return "No item here";
     }
 }//getItemString()

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

public HashMap getItems(){
     return this.aItems;
 }
```

```java
/**
  * Ajoute un Item
  */
 public void addItem(final String pNom,final Item pItem){
     this.aItems.put(pNom,pItem);
 }//addItem()

/**
  * Enleve un Item
  */
 public void removeItem(final String pNom){
     this.aItems.remove(pNom);
 }//removeItem()

public void setItems(final Item pItem){
    this.aItems.put(pItem.getNomItem(),pItem);
}
```

7.32 - Poids max du joueur

Attribut dans Player :

```java
private int aMaxPoids;
```

puis une condition dans take pour regarder le poids courant du joueur qui est aussi un attribut de Player

Changement après le 7.44 Beamer :

```java
if(this.aPlayer.getCurrentRoom().isInRoom(this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2))){
    if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)) != null){
        if(this.aPlayer.getCurrentPoids() + this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2).getPoidsItem()  <= this.aPlayer.getMaxPoids()){
            int vPoids = this.aPlayer.getCurrentPoids()+(this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)).getPoidsItem();
            this.aPlayer.setCurrentPoids(vPoids);
            if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)).getNomItem().equals(this.aBeamer.getNomItem())){
                this.aCheckBeamer = true;
                this.aGui.println("You picked up the beamer !");
            }
            else if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)).getNomItem().equals("pill")){
                this.aPlayer.setMaxPoids(1000);
            }
            this.aPlayer.take(vCommand2);
        }
        else{
            this.aGui.println("This item cannot be picked up, it's too heavy");
            return;
        }
    }
    this.aGui.println(this.aPlayer.getInventory().getItemString());
    this.aGui.println(this.aPlayer.getCurrentRoom().getItemList().getItemString());

}
else{
    this.aGui.println("not in room");
}
break;
```

//Pour l'instant le poids max ne peut pas être atteint mais j'implementerais cela pour la version finale// **implémentation réussie**

7.33

```
private ItemList aInventory;
```

```
this.aInventory = new ItemList("inventory");
```

dans ItemList

```
/**
  * Getter de l'Item en String
  */
 public String getItemString(){
     if(this.aItems != null){
         if(this.aTypeList.equals("inventory")){
             String vAll = "Inventory :";
             Set<String> keys = this.aItems.keySet();
             for(String item : keys) {
               vAll += " "+ item;
             }
             return vAll;
         }
         else if(this.aTypeList.equals("room")){
             String vAll = "Items in the room :";
             Set<String> keys = this.aItems.keySet();
             for(String item : keys) {
               vAll += " "+ item;
             }
             return vAll;
         }
         else{
             return "";
         }
     }
     else{
         return "No item here";
     }
 }//getItemString()
```

puis dans GameEngine pour l'afficher ensuite

7.34

La fonction ci-dessous va nous être utile pour regarder si l'objet "pill" est dans l'inventaire du joueur et change le aPoidsMax du joueur

```
/**
* methode MagicCookie()
*/
public boolean MagicCookie(final String pItem){
 if(isInInventory(this.aInventory.getItem(pItem))){
     this.setMaxPoids(200);
     return true;
 }
 else{
     return false;
 }
}
```

Après le 7.44 Beamer dans le case de take:

```
}
else if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)).getNomItem().equals("pill")){
    this.aPlayer.setMaxPoids(1000);
}
```

7.42 - Time Limit

Ajout d'un attribut aTime dans Game Engine

On implémente alors un compteur qui compte le nombre de déplacements du joueur dans case 3 qui gère cela.

Le joueur a alors droit qu'à 10 déplacements pour l'instant mais on changera cela par la suite.

```
case 3 :
    if(!vCommand.hasSecondWord()){
        this.aGui.println("Go where?");
        break;
    }
    else{
        if(this.aTime <= 10){
            this.goRoom(vCommand);
            this.aTime += 1;
            break;
        }
        else{
            this.aGui.println("You have no time left, the bomb is exploding.....END");
            this.endGame();
            break;
        }
    }
```

On ajoute également le compteur dans la fonction back.

7.42.2 - IHM

Je souhaite garder l'interface graphique actuelle.

7.43 - TrapDoor

Pour la commande goRoom il suffit juste de créer des sorties null pour ne avoir des sorties à sens unique

```
//Elevator
vElevator.setExit("west",vShibuyaDis2)
vElevator.setExit("down",vBunker);
//Bunker
vBunker.setExit("up",null);
vBunker.setExit("east",vWeaponRoom);
//Control Room
```

Pour la commande back il faut supprimer les stacks déjà existants lorsqu'on entre dans une "trap door" grâce à une nouvelle méthode de Room

isExit :

```
public boolean isExit(final Room pRoom){
    if(this.aExits.containsKey(pRoom)){
        return true;
    }
    else{
        return false;
    }
}//isExit
```

puis dans le case de back on vérifie si la sortie de la room est à sens unique :

```
if(this.aPlayer.getPreviousRoom() != null){
if(!this.aPlayer.getCurrentRoom().isExit(this.aPlayer.getPreviousRoom())){
    this.aGui.println("You are trapped");
}
```

7.44 - Beamer

Création d'une classe Beamer :

```
/**
 * Write a description of class Beamer here.
 *
 * @author LEFEVRE Gregoire
 * @version vFinale
 */
public class Beamer extends Item
{
    private Room aChargeRoom;
    private boolean aCharge = false;
    public Beamer(){
        super("beamer","an item which can used to teleport",10);
        this.aChargeRoom = null;
    }

    public void chargeBeamer( final Room pRoom){
        this.aChargeRoom = pRoom;
        this.aCharge = true;
    }

    public Room fireBeamer(){
        return this.aChargeRoom;
    }

    public boolean isCharged(){
        return this.aCharge;
    }
}
```

Création d'attributs dans GameEngine :

```
private Beamer          aBeamer;
private boolean         aCheckBeamer;
```

```
/**
 * Constructor for objects of class GameEngine
 */
public GameEngine()
{
    this.aParser = new Parser();
    this.aBeamer = new Beamer();
    this.aCheckBeamer = false;
    this.createRooms();
}//GameEngine()
```

Il faut désormais regarder grâce au boolean aCheckBeamer si le Beamer est dans l'inventaire du joueur a chaque fois qu'il take ou drop un objet :

Dans take & drop on test pour savoir si le beamer est dans l'inventaire/pièce du joueur pour changer la valeur de aCheckBeamer :

```
case 6 :


    if(this.aPlayer.getCurrentRoom().isInRoom(this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2))){
        if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)) != null){
            if((this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2)).getNomItem().equals(this.aBeamer.getNomItem())){
                this.aCheckBeamer = true;
                this.aGui.println("You picked up the beamer !");
            }
        }
        this.aPlayer.take(vCommand2);
        this.aGui.println(this.aPlayer.getInventory().getItemString());
        this.aGui.println(this.aPlayer.getCurrentRoom().getItemList().getItemString());
    }
    else{
        this.aGui.println("not in room");
    }
    break;
case 7:
    if(this.aPlayer.isInInventory(this.aPlayer.getInventory().getItem(vCommand2))){
        this.aPlayer.drop(vCommand2);
        if(this.aPlayer.getCurrentRoom().isInRoom(this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2))){
            this.aGui.println("You droped the beamer !");
            this.aCheckBeamer = false;
        }
        this.aGui.println(this.aPlayer.getInventory().getItemString());
        this.aGui.println(this.aPlayer.getCurrentRoom().getItemList().getItemString());

    }
    else{
        this.aGui.println("not in inventory");

    }
    break;
```

Ajout des nouveaux mots de commande dans mon switch :

```
}
else if(vCommand1.equals("charge")){
    index = 10;
}
else if(vCommand1.equals("fire")){
    index = 11;
}
```

```
case 10 :

    if(this.aCheckBeamer){
       this.aGui.println("The Beamer is charging");
       this.aBeamer.chargeBeamer(this.aPlayer.getCurrentRoom());
    }
    else{
        this.aGui.println("You dont have the beamer ....");
        }

    break;


case 11 :
    if(this.aCheckBeamer){
        if(this.aBeamer.isCharged()){
        this.aGui.println("The Beamer is firing");
        this.aPlayer.setCurrentRoom(this.aBeamer.fireBeamer());
        }
        else{
        this.aGui.println("The beamer is not charged");
        }
    }
    else{
        this.aGui.println("You don't have the beamer ....");
    }


    if( this.aPlayer.getCurrentRoom().getImageName() != null ){
            this.aGui.showImage( this.aPlayer.getCurrentRoom().getImageName() );
            break;
            }

    break;
```

7.45.1

Mise à jour des fichiers tests

7.45.2

Régénération de la documentation

7.46
Création de la classe RoomList similaire à ItemList ou l'on va stocker nos Rooms

```java
public class RoomList
{
    private HashMap <Integer, Room> aRooms;
    /**
     * constructeur de RoomList
     */
    public RoomList(){
        this.aRooms = new HashMap<Integer, Room>();
    }


     /**
    * getter getRoom() retourne la Room dans la Hashmap
    */
    public Room getRoom(final int pInt){
        return this.aRooms.get(pInt);
     }

    /**
    * getter getRooms() retourne la Hashmap
    */
    public HashMap getRooms(){
        return this.aRooms;
     }

    /**
      * Ajoute une Room
      */
     public void addRoom(final int pInt,final Room pRoom){
        this.aRooms.put(pInt,pRoom);
     }

    /**
    * getter getRooms() retourne la Hashmap
    */
    public Room getRandomRoom(){
        int vRandom = (int)(Math.random()*(17));
        if(vRandom <= 17){
            return getRoom(vRandom);
        }
        else{
            return getRoom(0);
        }
     }
}
```

Dans Game engine on rajoute un attributs RoomList
puis on va l'initialiser dans la fonction create Rooms :

```
//## RoomList

this.aRoomList.addRoom(0,vChambre);
this.aRoomList.addRoom(1,vEntree);
this.aRoomList.addRoom(2,vRDC);
this.aRoomList.addRoom(3,vRDC2);
this.aRoomList.addRoom(4,vDarkAlley);
this.aRoomList.addRoom(5,vLocauxPoubelles);
this.aRoomList.addRoom(6,vBar);
this.aRoomList.addRoom(7,vCyberShop);
this.aRoomList.addRoom(8,vShibuyaDis);
this.aRoomList.addRoom(9,vShibuyaDis2);
this.aRoomList.addRoom(10,vSkyScraper);
this.aRoomList.addRoom(11,vCommerces);
this.aRoomList.addRoom(12,vRestaurant);
this.aRoomList.addRoom(13,vElevator);
this.aRoomList.addRoom(14,vBunker);
this.aRoomList.addRoom(15,vControlRoom);
this.aRoomList.addRoom(16,vRobotRoom);
this.aRoomList.addRoom(17,vInfirmerie);
```

Il suffit ensuite de mettre une sortie de notre Transport Room en une salle aléatoire, ici vRobotRoom sortie sud :

```
//Robot Room
vRobotRoom.setExit("south",this.aRoomList.getRandomRoom());
```

Améliorations effectuées ;

On obtient la description d'un objet lorsqu'on le prend :

```
if(this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2).getDescriptionItem() != null){
    this.aGui.println("You have picked up " + this.aPlayer.getCurrentRoom().getItemList().getItem(vCommand2).getDescriptionItem());
}
```

7.46.1 fonction alea

Mot de commande aléa avec un attributs aModeTest qui vérifie si l'on est en mode Test, puis un switch qui va choisir une la Room passe en String et y transporter le joueur.

```
case 12 :
    if(this.aPlayer.getCurrentRoom().getDescription().equals("in the huge space where they store EVA-809, your mecha")){
        this.aBooleanAlea = true;
    }

    if(this.aModeTest && this.aBooleanAlea){
        if(vCommand.hasSecondWord()){
            String vS = vCommand.getSecondWord();
            int vRoomNumber = 0;
            switch(vS){
                case "entree" :
                    vRoomNumber = 1;
                    break;
                case "rdc" :
                    vRoomNumber = 2;
                    break;
                case "rdc2" :
                    vRoomNumber = 3;
                    break;
                case "darkalley" :
                    vRoomNumber = 4;
                    break;
                case "locaux" :
                    vRoomNumber = 5;
                    break;
                case "bar" :
                    vRoomNumber = 6;
                    break;
                case "cybershop" :
                    vRoomNumber = 7;
                    break;
                case "shibuya" :
                    vRoomNumber = 8;
                    break;
                case "shibuya2" :
                    vRoomNumber = 9;
                    break;
```

```
                vRoomNumber = 9;
                break;
            case "skyscraper" :
                vRoomNumber = 10;
                break;
            case "commerces" :
                vRoomNumber = 11;
                break;
            case "restaurant" :
                vRoomNumber = 12;
                break;
            case "elevator" :
                vRoomNumber = 13;
                break;
            case "bunker" :
                vRoomNumber = 14;
                break;
            case "control" :
                vRoomNumber = 15;
                break;
            case "robotroom" :
                vRoomNumber = 16;
                break;
            case "infirmerie" :
                vRoomNumber = 17;
                break;
            default :
                vRoomNumber = 0;
                break;
        }
        this.aPlayer.setCurrentRoom(this.aRoomList.getRoom(vRoomNumber));
        this.aModeTest = false;
        if( this.aPlayer.getCurrentRoom().getImageName() != null ){
            this.aGui.showImage( this.aPlayer.getCurrentRoom().getImageName() );
            break;
        }
    }
}
break;
```

7.48 - Character

Création d'une classe character (similaire a Item) :

```
public class Character
{
    private String      aDialogue;
    private String      aName;
    private Room        aRoomChara;

    public Character(final String pName, final String pDialogue, final Room pRoomChara){
        this.aName = pName;
        this.aDialogue = pDialogue;
        this.aRoomChara = pRoomChara;
    }

    public String getName(){
        return this.aName;
    }

    public String getDialogue(){
        return this.aName;
    }

    public Room getRoomChara(){
        return this.aRoomChara;
    }

    public void setName(final String pString){
        this.aName = pString;
    }

    public void setDialogue(final String pString){
        this.aDialogue = pString;
    }

    public void setRoomChara(final Room pRoom){
        this.aRoomChara = pRoom;
    }
}
```

Création d'une classe CharacterList (similaire a ItemList) :

```java
private HashMap<String,Character> aCharacters;
/**
 * Constructeur CharcaterList()
 */
public CharacterList(){
    this.aCharacters = new HashMap<String,Character>();
}//CharacterList

/**
 * Accesseur en String des Personnages present dans la piece
 */
public String getCharacterString(){
    if(this.aCharacters != null){
        String vAll = "People here :";
        Set<String> keys = this.aCharacters.keySet();
        for(String character : keys) {
            vAll += " "+ character;
        }
        return vAll;
    }
    else{
        return "";
    }
}//getCharacterString()

/**
 * Accesseur du Personnage present dans la HashMap
 * @param pCharacter
 */
public Character getCharacter(final String pCharacter){
    return this.aCharacters.get(pCharacter);
}//getCharacter()

/**
 * Permet d'enlever un Personnage de la HashMap
 * @param pName
 */
public void removeCharacter(final String pName){
    this.aCharacters.remove(pName);
}//removeCharacter()

/**
 * Methode permettant d'ajouter un Personnage a la HashMap
 *
 * @param pCharacter
 */
public void setCharacters(final Character pCharacter){
   this.aCharacters.put(pCharacter.getName(),pCharacter);
}//setCharacters()
```

nouveau mot de commande talk :

```java
// a constant array that will hold all valid command words
private final String[] aValidCommands = {
    "go","quit","help","look","eat","back","test","take","drop","bag","weight","charge","fire", "alea" , "talk"
};
```

Ceci nous permet donc de parler avec les personnages mais seulement leur unique ligne de dialogue.

```
case 13 :
    if ( !vCommand.hasSecondWord() ){
        this.aGui.println( "Talk to who ?" );
        break;
    }
    else{
        if(this.aPlayer.getCurrentRoom().getCharacterList() != null){
            if(this.aPlayer.getCurrentRoom().getCharacterList().getCharacter(vCommand2) != null){
               this.aGui.println(this.aPlayer.getCurrentRoom().getCharacterList().getCharacter(vCommand2).getDialogue());
            }
            else{
                this.aGui.println("There's no people by this name here.");
            }
        }
        break;
    }
```

Pour ce faire on ajoute une condition qui change le dialogue du personnage dans talk :

```
if(this.aPlayer.getCurrentRoom().getCharacterList().getCharacter(vCommand2).getName() == "Doctor"){
    this.aPlayer.getCurrentRoom().getCharacterList().getCharacter(vCommand2).setDialogue("You need to get a pill somwhere in the CyberShop to increase physical strength");
}
```

```
People here : Doctor
> talk Doctor
Hello how can I help you?
> talk Doctor
You need to get a pill somwhere in the CyberShop to increase physical strength
```

1ere condition de défaite du jeu :
Si l'on arrive a vRobotRoom sans la 'pill' on ne peut pas prendre le robot pour sauver la ville a moins d'avoir chargé le beamer avant l'ascenseur car après 20 passage de piece en piece on perd automatiquement (voir TimeLimit)

2eme condition de défaite du jeu :

```
if(this.aPlayer.getCurrentRoom() != null){
    if(this.aPlayer.getCurrentRoom().getDescription().equals("where the trash is disposed.")){
    this.aGui.println("You feel dizzy as if you were just stabbed....");
    this.endGame();
    }
}
```

Si on va jusqu'au bout de l'allée sombre.

Condition de victoire :
Aller chercher l’item ‘pill’ pour pouvoir aller dans le robot puis sortir de la salle du robot en moins de 20 passages.

Déclaration obligatoire : Anti-Plagiat

Cette première partie et deuxième partie a été réalisée sans plagiat. J'ai utilisé les codes fournis dans le livre mais cela n’est pas considéré comme du plagiat.