Titre : sauver sa famille du covid

Auteur : Firas Miladi, Groupe 8

Table des matières :

- Présentation du jeu

Auteur :
Je suis Miladi Firas du Groupe 8

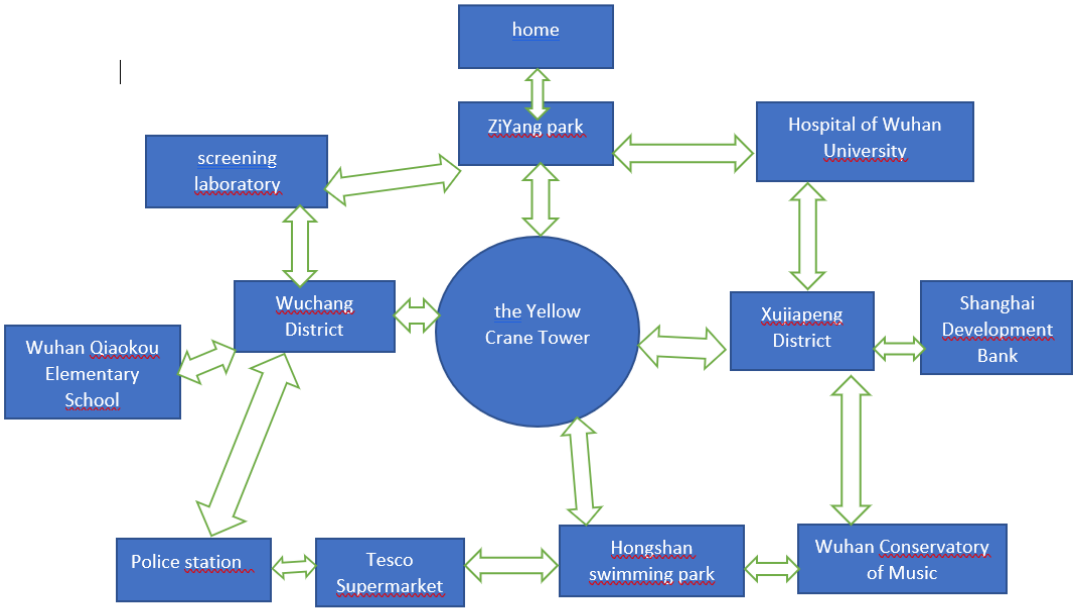
Thème :
A Wuhan Un père cherche à sauver sa famille du virus mortel.

Résumé du scénario :

Dans ce jeu, vous débutez la partie à Wuhan, et vous contrôlez un homme qui doit récupérer sa femme et ses enfants et retournez chez lui pour les sauver du covid.

Arriver chez vous, vous devez aller faire les courses puisqu'un confinement total est annoncé pour le lendemain. De plus, vous n'avez que 30 minutes de jeu pour récupérer ta famille, faire les courses et rentrer chez vous avant le couvre-feu.

Plan



Scénario détaillé

Au cours d'une journée comme toutes les autres, Kong est au travail il pense à faire une surprise à sa famille. Soudain, il aperçoit un mouvement et des bruits inhabituels dans l'entreprise. Les médias viennent de lancer un appel d'urgence a toute la population de regagner leurs domiciles en toute urgence. Un virus mortel vient de se propager dans la ville en faisant déjà des milliers de mort et de contaminés. Kong doit récupérer sa femme Hong qui travaille à l'administration de l'Hospital de Wuhan, sa fille qui suit des cours a l'école de musique et son fils élève à l'école Wuhan Qiaokou. Tout au long de son parcours, il va trouver des objets qu'ils vont l'aider dans sa mission tel que des masques, du gel hydroalcoolique, des gants, des visières…et il va rencontrer des personnes porteuses du virus qu'il doit les éviter. Arriver chez lui, il doit aller faire les courses puisqu'un confinement total est annoncé. De plus, il n'a que 30 minutes de jeu pour récupérer sa famille, faire les courses et rentrer chez lui avant le couvre-feu.

Détail des lieux, items, personnages
Personnage Kong banquier le père Hong la mère
13 lieu

1/home

2/Ziyang park

3/screening laboratory

4/Hospital of Wuhan University

5/the Yellow Crane Tower

6/Xujiapeng District

7/Shanghai Development Bank

8/Wuchang District

9/Wuhan Qiaokou Elementary School

10/Police station

11/Tesco Supermarket

12/ Hongshan swimming park

13/ Wuhan Conservatory of Music

Items :
-un téléphone
-des masques
-du gel hydroalcoolique

- des gants
- des visières

Situations gagnantes et perdantes

Il doit sauver sa famille en rentrant chez lui sans que quelqu'un d'entre eux n'attrape le covid et faire les courses en respectant le temps imposé le couvre-feu.
Si l'un des membres de sa famille attrape le covid, ou si vous vous faites embarquer par la police (ne respecte pas le couvre-feu) la partie est perdue.

Eventuellement énigmes, mini-jeux, combats, etc.

Le père doit éviter les personnes infectées et notamment les affronter

Réponses aux exercices :

Le rôle de chaque classe est :

La class Game : cette class c'est la classe principale de du programme notamment avec la méthode play() qui permet de commencer le jeu. Cette class permet de lire et d'exécuter les commandes.

La class Parser : cette classe lit le terminal où l'utilisateur tape ses commandes. Elle essaye d'interpréter les mots comme des commandes, et retourne des objets de la classe Command.

La class CommandWord : cette class définie les commandes valides pour le jeu

La class Command : cette class produit une commande de l'utilisateur

La class Room : cette class créer les lieux du jeu, et les connexions entre eux

Ex 7.5 printLocationInfo :

La méthode printLocationInfo a été créée dans la classe Game pour éviter la duplication de code, comme indiquée dans le livre. Elle affiche la position actuelle du joueur et également les différentes sorties.

Quand on modifie une partie du code, il faut apporter plusieurs changements dans le code pour éviter les incohérences notamment, les méthodes goRoom et printWelcome.

Après la création de printLocationInfo, Les méthodes printWelcome et goRoom pourront donc appeler la méthode printLocationInfo quand elles auront besoin d'afficher des informations sur la situation pour faciliter les futures modifications qui pourraient arriver,

```
private void printLocationInfo(){
   System.out.println("You are " + this.aCurrentRoom.getDescription());
   System.out.print("Exits: ");
   if(this.aCurrentRoom.northExit != null)
      System.out.print("north ");
   if(this.aCurrentRoom.eastExit != null)
      System.out.print(" east ");
   if(this.aCurrentRoom.southExit != null)
      System.out.print(" south ");
   if(this.aCurrentRoom.westExit != null)
      System.out.print(" west ");
   System.out.println();}
```

Ex 7.6 getExit()

Le guetteur getExit() de la classe Room permet de réduire le niveau de couplage

```
public Room getExit(String direction){
        if(pDirection.equals(« north »)) {
                return aNorthExit ;}
        if(pDirection.equals(« east »)) {
                return aEastExit ;}
        if(pDirection.equals(« south »)) {
                return aSouthExit ;}
```

```
if(pDirection.equals(« west »)) {
        return aWestExit ;}
return null ;}
```

Ce guetteur va être utiliser dans la classe Game (méthode goRoom()) en rendant le code beaucoup plus court
On passe de ce code :

```
Room vNextRoom = null;
if(vDirection.equals("north")) { vNextRoom = this.aCurrentRoom.
aNorthExit;}

if(vDirection.equals("east")) { vNextRoom = this.aCurrentRoom.
aEastExit;}

if(vDirection.equals("south")) { vNextRoom = this.aCurrentRoom.
aSouthExit;}

if(vDirection.equals("west")) { vNextRoom = this.aCurrentRoom.
aWestExit;}
```

a cela :

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

Ex 7.7 getExitString() :

Cette méthode se situe dans la class Room. Son rôle est de simplifier la méthode printLocationInfo. Cette méthode va simplifier les modifications futures du code.

```
Public Room getExitString() {

String vExits = "Exits : ";

if(this.aNorthExit != null ){ vExits +=" north ");}

if(this.aEastExit!= null){ vExits +=” east ");}

if(this.aSouthExit!= null){ vExits += “south ");}
```

```
if(this.aWestExit!= null){ vExits +=" west ");}

returrn vExits }}

private void printLocationInfo()
   {
      System.out.println("You are
"+this.aCurrentRoom.getDescription());
      System.out.print(this.aCurrentRoom.getExitString());
      System.out.println();
   }
```

Ex7.8 (HashMap)

Dans cet exercice on doit intégrer une HashMap pour faciliter la création des rooms et les directions

Le HashMap est un tableau ou les indices ne sont pas des entiers, mais des objets que l'on nommera "key" (ici les keys sont les String).

Pour utiliser ce paquetage, il faut importer la java.util.HashMap;

```
import java.util.HashMap;
public class Room
{
   private String aDescription ;
   private HashMap<String, Room> aExits;

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

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

   public void setExits(final String pDirection,final Room pNeighbor )
   {
```

```
        aExits.put(pDirection, pNeighbor);
    }

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

    public String getExitString()
    {
        String vExits = "Exits : ";

if(this.aNorthExit != null ){ vExits +=" north ");}

if(this.aEastExit!= null){ vExits +=” east ");}

if(this.aSouthExit!= null){ vExits += “south ");}

if(this.aWestExit!= null){ vExits +=" west ");}

returrn vExits }}
    }

} // Room
```

Ex 7.9 (keySet)
Il faut modifier également la méthode getExitString()

```
public String getExitString()
    {
        String returnString = "Exits: ";
        Set<String> keys = aExits.keySet();
        for(String vexit :keys)
        {
            returnString+=" " +vexit;
        }
        return returnString;
    }
```

Ex 7.10

Les directions de sortie générées par la méthode getExitString() sont les "key" de la table de hachage aExits.

L'interface Set est une collection d'objets sans ordre avec des éléments uniques sans doublant.

La méthode keySet() créer un ensemble d' objet de type set<>

En faisant «java Set<String> vKeys = this.aExits.keySet();»
On stock toute les "key" (direction de sortie de l'objet courant) dans la collection de types Set<String> vKeys.

Ex 7.11 (getLongDescription)

on ajoute la méthode getLongDescription()dans la classe Room ,puis on modifie la méthode printLocationInfo().

```
public String getLongDescription()
    {
        return "You are "+ this.aDescription+"\n" +getExitString();
    }
```

Ex 7.14 look
Dans cet exercice, il faut ajouter une nouvelle commande « look ».

```
private void look()
    {
        System.out.println(this.aCurrentRoom.getLongDescription());
    }
```

Le problème c'est si on ajoute look seulement dans la class game l'utilisateur ne peut jamais l'utiliser car elle n'est pas connu donc on doit l'ajouter dans la class CommandWords

```
private static final String[] aValidCommands=
{"go","help","quit","look","eat"};
```

et egalement la méthode processCommand qui exécute les commandes dans la class game

```
private boolean processCommand  (final  Command pCom){
```

```
        boolean vReturn = false ;
        if(pCom.isUnknown())
        {
            System.out.println("I don't know what you mean...");
            return false;
        }
        //the command is unknown

        String vcomtostring = pCom.getCommandWord();

        if(vcomtostring.equals("quit"))
        {
            vReturn=true;
        }
        else if(vcomtostring.equals("help"))
        {
            printHelp();
        }
        else if(vcomtostring.equals("go"))
        {
            goRoom (pCom);
        }else if(vcomtostring.equals("look"))
        {
            look();
        }

        return vReturn;
        // execute the commands
    }
```

Exercice 7.15 (eat)

Pour ajouter la commande eat() , il faut faire les mêmes étapes que l'exercice 7.14

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

```
private boolean processCommand  (final  Command pCom){
        boolean vReturn = false ;
        if(pCom.isUnknown())
        {
            System.out.println("I don't know what you mean...");
            return false;
        }
        //the command is unknown

        String vcomtostring = pCom.getCommandWord();

        if(vcomtostring.equals("quit"))
        {
            vReturn=true;
        }
        else if(vcomtostring.equals("help"))
        {
            printHelp();
        }
        else if(vcomtostring.equals("go"))
        {
            goRoom (pCom);
        }else if(vcomtostring.equals("look"))
        {
            look();
        }
        else if(vcomtostring.equals("eat"))
        {
            eat();
        }

        return vReturn;
        // execute the commands
    }
```

Exercice 7.16

Si on utilise la méthode help on se rend compte que les commandes look et eat ne font pas partie des commandes proposées

Pour régler ce problème on doit ajouter une méthode showAll() pour afficher toutes les commandes sans avoir besoin de les ajouter manuellement une par une .

```
public void showAll()
   {
      for(String vcommand : aValidCommands){
         System.out.println(vcommand+" ");
      }
      System.out.println();
   }
```

Et puisqu'il n'y a pas de couplage entre la class Game et CommandWords on doit passer de CommandWords a Parser et de Parser a Game

```
//Parser
public void showCommands(){
      aValidCommands.showAll();
   }

//Game
private void printHelp ()
   {
      System.out.println("You are lost. You are alone. You wander
around at the university.");
      System.out.println("");
      System.out.println("Your command words are:");
      aParser.showCommands();
   }
```

Exercice 7.17

A chaque fois qu'on ajoute une nouvelles commandes Il faut changer la classe game comme nous l'avons fait pour eat() et look().

Exercice 7.18 : getCommandList :
On effectue les changements suivants

Dans la classe CommandWords : on remplace showAll() par getCommandsList():

```
public String getCommandList()
   {
```

```
        String commandList = "";
        for(String command : aValidCommands) {
            commandList += command + "  ";
        }
        return commandList;
    }
```

Dans la classe Parser on modifie de la procédure void showCommands() en String showCommands():

```
public String getCommandList()
    {
        return aValidCommands.getCommandList();
    }
```

Dans la classe Game
On modifie également la méthode printHelp():

Exercice 7.18.1
Modifications apportées comme demandé.

Exercice optionnel 7.18.2 : stringbuilder :
Grace a stringbuilder On peut construire une chaîne de caractère, avec la méthode append () qui convertie la data en String avant de l' ajoute à la suite des autres

Exercice 7.18.3 : chercher des images :
J'ai commencé la conception d'image sur unity 3d , mais puisque mon jeu se passe dans un lieu réel, j'ai eu l'idée d'utiliser des images des lieux pour ajouter un peu de réalisme dans le jeu . Donc j'ai trouvé les images dans google images.

Exercice 7.18.4 :Décider du titre du jeu :
J'ai décidé de nommer mon jeu « Wuhan War V». Je me suis inspiré du film World War Z. J'ai remplacé world par Wuhan puisque c'est le lieu ou le jeu se déroule et au lieu de «Z» pour zombie j'ai utilisé « V» pour virus .

Exercice 7.18.5 optionnel: HashMap des Room :

Modifications apportées comme demandé.

Exercice 7.18.6 :Zuul with images :
le projet zuul-with-image, apporte au projet une interface graphique, la possibilité d'apporter des photos (GameEngine)
et de gerer des panels (UserInterface)

Exercice 7.18.7 optionnel :
La methode addActionListener ajoute des listeners (des "écouteurs") reagir suite à un événement declanché par l'utilisateur
La methode actionPerformed permet de déclarer qu'une action a été effectué

Exercice 7.18.8 :
Pour cela j'ai créé 7 bouton colorés 4 pour la direction et 3 pour eat look et help
J'ai également programmé les boutons de tel facon ils affichent que les directions possibles

```
public void makeBoutonBar()
   {
       aButton = new JPanel();
       aButton.setLayout(new GridLayout(0,1,3,5));

       this.aButtonN = new JButton("north");
       this.aButtonN.addActionListener(this);
       this.aButtonN.setBackground(Color.RED);

       this.aButtonS = new JButton("south");
       this.aButtonS.addActionListener(this);
       this.aButtonS.setBackground(Color.GREEN);

       this.aButtonE = new JButton("east");
       this.aButtonE.addActionListener(this);
       this.aButtonE.setBackground(Color.YELLOW);

       this.aButtonW = new JButton("west");
       this.aButtonW.addActionListener(this);
       this.aButtonW.setBackground(Color.MAGENTA   );

       this.aButtonEat = new JButton("eat");
       this.aButtonEat.addActionListener(this);
```

```
        this.aButtonEat.setBackground(Color.CYAN);

        this.aButtonLook = new JButton("look");
        this.aButtonLook.addActionListener(this);
        this.aButtonLook.setBackground(Color.WHITE);

        this.aButtonHelp = new JButton("help");
        this.aButtonHelp.addActionListener(this);
        this.aButtonHelp.setBackground(Color.ORANGE);

        aButton.add( this.aButtonN);
        aButton.add( this.aButtonS);
        aButton.add( this.aButtonE);
        aButton.add( this.aButtonW);
        aButton.add( this.aButtonEat);
        aButton.add( this.aButtonHelp);
        aButton.add( this.aButtonLook);

        this.aMyFrame.getContentPane().add(aButton,
BorderLayout.EAST);
    }
```

Et pour que les boutons affichent que les directions possibles

```
public void boutonvisibility(Room pRoom){
        this.aButtonN.setVisible(pRoom.getExit("north")!=null);
        if (pRoom.getExit("north")==null){this.aButtonN.setVisible(false);}
        else {this.aButtonN.setVisible(true);}
        if (pRoom.getExit("south")==null){this.aButtonS.setVisible(false);}
        else {this.aButtonS.setVisible(true);}

        if (pRoom.getExit("west")==null){this.aButtonW.setVisible(false);}
        else {this.aButtonW.setVisible(true);}

        if (pRoom.getExit("east")==null){this.aButtonE.setVisible(false);}
        else {this.aButtonE.setVisible(true);}

        this.aMyFrame.pack();
    }
```

Exercice 7.19 (OPTIONNEL)

Ce modèle de conception est un modèle architectural qui sépare une application en trois composants logiques principaux : modèle (responsable de l'interaction entre la base de données et le code pour traiter ces données), vue (responsable de l'interface utilisateur) et contrôleur (responsable de la synchronisation entre le modèle et les vues)

Exercice 7.19.2 : Déplacer toutes les images
Modifications apportées comme demandé.

Exercice 7.20/7.21/7.22
J'ai créé une classe nommée « Item »

```
public class Item
{
    private String aNoun;
    private String aDescription;
    private double aPrix;
    private double aPoids;

    public Item(final String pNoun, final String pDescription, final double
pPrix, final double pPoids)
    {
        this.aNoun        = pNoun;
        this.aDescription = pDescription;
        this.aPrix        = pPrix;
        this.aPoids       = pPoids;
    }
    public String getNounItem(){return this.aNoun;}
    public String getDescriptionItem(){return this.aDescription + " || price
: "+ this.aPrix +" Yuan || Weight : "+ this.aPoids +" Kg";}
    public double getPriceItem(){return this.aPrix;}
    public double getWeightItem(){return this.aPoids;}
}
```

J'ai ajouté dans Room une liste d'objets (HashMap) c'est les items presents dans la room la création des objets se passe dans la classe GameEngine .
Pour ajouter une description au item :

```
public String getLongDescription()
    {
```

```
        return "You are "+ this.aDescription+"\n" +getExitString()+"\n"+
this.alist.getItemListStringRoom();
    }
```

Exercice 7.22.1
L'utilisation d'une HashMap nous permet d'ajouter autant d'objet on peut ajouter des objets grâce à la méthode put() et les retrouver directement grâce à la méthode get() de la HashMap .

Exercice 7.22.2
Modifications apportées comme demandé.

Exercice 7.23 (back)
On utilise pour cela une Stack private Stack<Room>   aBackRooms;
On ajoute cette methode dans la classe GameEngine :

```
private void Back(){
        if(aBackRooms.empty()){this.aGui.println("you can't go back");}
        else{
            this.aPlayer.setCurrentRoom(this.aBackRooms.pop());
            this.printLocationInfo();
            if ( this.aPlayer.getCurrentRoom().getImageName() != null )
                this.aGui.showImage(
this.aPlayer.getCurrentRoom().getImageName() );
            }
    }
```

On modifie la méthode interpretCommand()

```
if ( vCommandWord.equals( "test" ) ){
            this.test(vCommand);
        }
```

Dans la classe CommandWords :
On ajoute «back» dans le tableau de Strings aValidCommands.

Exercice 7.24
La commande fonctionne

Exercice 7.25
Utiliser deux fois la commande back permet de revenir deux salles en arrière

Exercice 7.26/ 7.26.1

Modifications apportées comme demandé.

Exercice 7.27/ 7.28/ 7.28.1 / 7.28.2/ 7.28.3

J'ai cree trois documents .txt dans lequel on a mis des commandes et apres qu'on a ajouté test dans le tableau de Strings aValidCommands On ajoute cette methode dans la classe GameEngine

```
private void test(Command pCommand){
        if(!pCommand.hasSecondWord()){
            this.aGui.println("What file do you want to use? ");
            return;
        }else{
            String vFichier = pCommand.getSecondWord();
            Scanner vScan = new
Scanner(this.getClass().getClassLoader().getResourceAsStream("./"+vFic
hier+".txt") );
            String vSecond = vScan.nextLine();

            while(vScan.hasNextLine()){
                interpretCommand(vSecond);
                vSecond=vScan.nextLine();
            }
        }

    }
```

l'exercice 7.29 (Player)
Dans cet exercice il faut créer une classe Player ainsi que tout les attributs qui concerne le joueur (Nom , argent initialement fourni + collecté ,le poids max qui peut le soulever).

Exercice 7.30/7.31 (take, drop)
Pour cela il faut ajouter un attribut Initialisé dans la classe player de type HashMap<String,Item>aInventory pour répertorier les items pris par le joueur pendant le jeu

Dans la creation de ces deux methodes j'ai pris compte l'utilisation de plusieurs joueur au meme temps

Pour la methode take j'ai cree cette methode dans la classe player

```
public String takeString(String pCommand){
        Item vItemRoom =
this.getCurrentRoom().getRoomItemlist().getItemList(pCommand);
        if (vItemRoom == null) {
            return "\n there is no such as this thing in this room ";

        }else {
            if (pCommand.equals("gold")){

            this.addItemInventory(pCommand,vItemRoom);

this.getCurrentRoom().getRoomItemlist().removeItemFromTheList(pCom
mand);
            return "You added a " + vItemRoom.getDescriptionItem() + " to
your wallet  ";

          }
          else {
          this.addItemInventory(pCommand,vItemRoom);

this.getCurrentRoom().getRoomItemlist().removeItemFromTheList(pCom
mand);
            return "You added a " + vItemRoom.getDescriptionItem() + " to
your inventory  ";
         }
       }
    }
```

Et dans la classe gameEngine j'ai fait appel a cette methode pour faire celle-ci

```
private void take(final Command pCommand) {
            if ( !pCommand.hasSecondWord() ) {
                this.aGui.println("\n" + "what do you want to take?" + "\n");
                return;
            }
            String vCommandWord = pCommand.getSecondWord();
```

```
        Item vItem =
this.aPlayer.getCurrentRoom().getRoomItemlist().getItemList(vCommand
Word);
        if(vItem==null){
            this.aGui.println("\n" + "there is no item" + "\n");
            return;
        }

if(vItem.getWeightItem()+this.aPlayer.getWeightInventory()<=this.aPla
yer.getOneRepMax() && vItem.getPriceItem()<=this.aPlayer.getwallet())
{
            String vCommand = pCommand.getSecondWord();
            this.aGui.println(this.aPlayer.takeString(vCommand));

this.aGui.println(this.aPlayer.getCurrentRoom().getRoomItemlist().getIte
mListStringRoom());

        }
        else
if(vItem.getWeightItem()+this.aPlayer.getWeightInventory()<this.aPlaye
r.getOneRepMax() && vItem.getPriceItem()>this.aPlayer.getwallet()) {
            this.aGui.println("it's to expensive man!\n you don't have
enough money to pick this item");
            return;

        }

        else
if(vItem.getWeightItem()+this.aPlayer.getWeightInventory()>this.aPlaye
r.getOneRepMax() && vItem.getPriceItem()<this.aPlayer.getwallet()) {
            this.aGui.println("it's to heavy man!\n you don't have enough
power to pick this item");
            return;

        }
        else
if(vItem.getWeightItem()+this.aPlayer.getWeightInventory()>this.aPlaye
r.getOneRepMax() && vItem.getPriceItem()>this.aPlayer.getwallet()) {
            this.aGui.println("it's to heavy and to expensive man!\n Try to
pick another item");
            return;
```

```
        }
    }
```

Pour la methode drop j'ai fait

```
public String dropString (String pCommand) {
        Item vItemdrop = this.aCatalogue.getItemList(pCommand);
        if (vItemdrop == null) {
            return "\n You don't own this item";
        }
        else {
            this.removeItemInventory(pCommand);

this.getCurrentRoom().getRoomItemlist().addItemToTheList(vItemdrop.g
etNounItem(),vItemdrop);
            return "You threw a " + vItemdrop.getDescriptionItem() + "
from your inventory ";
        }
    }
```

Et dans la classe GameEngine

```
private void drop(final Command pCommand) {
            if (!pCommand.hasSecondWord()) {
               this.aGui.println("\n What do you want to drop? \n");
               return;
            }
            String vCommand = pCommand.getSecondWord();
            this.aGui.println(this.aPlayer.dropString(vCommand));

this.aGui.println(this.aPlayer.getCurrentRoom().getRoomItemlist().getIte
mListStringRoom());
    }
```

Ce qui concerne la condition gold (if (pCommand.equals("gold")) ,jai cree un nouveau type d'item qui ne sera pas pris comme un simple item qui ne va pas être ajouter a l' inventaire mais de l'argent qui va directement s'ajouter dans le portfolio du jouer

Apres on a cree la classe ItemList qui nous permet de regrouper les methode sur les HashMap<String,Item> ce qui nous permet d'optimiser notre code et d'eviter la duplication

exercice 7.32. Poids Max
Cet exercice est déjà fait en prenant en compte le poids max et le prix max
exercice 7.33 (inventaire)
Dans la classe ItemList on ajoute une methode qui nous permet d'afficher tous les items sous forme de string en utilisant les StringBuilder. Apres l'ajout de la commande « items» dans CommandWords et dans interpretCommand() de la classe GameEngine, l'utilisation de la commande items nous permet d'afficher les items le poids de l'inventaire ainsi que l'argent restant
exercice 7.34 magic cookie
cet objet permet au joueur de soulever une charge deux fois plus lourde donc j'ai ajouté la methode doubleForce() .Dans la classe GameEngine J'ai créé cet objet et puis dans la méthode interpretCommand() j'ai ajouté a l'utilisateur la possibilité de « eat cookie » .
Exercice 7.34.1 : Mettre à jour les fichiers de test
Exercice 7.34.2 : Re-générer les 2 javadoc

Exercice 7.35. zuul--with--enums--v1
Modifications apportées comme demandé.

Exercice 7.35.1. Switch
Modifications apportées comme demandé.

Exercice 7.41.1. zuul--with--enums--v2
Modifications apportées comme demandé.

Exercice 7.41.2
Modifications apportées comme demandé.

exercice 7.42
j'ai créé un compteur qui sera pris en compte par la methode interpretCommand() et quand il est a 0 il fait un retour if (this.aCount==0){return;}

exercice 7.42.1

apres l'import de
import javax.swing.Timer;

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```
j'ai utilisé les methode de Timer pour créer un compteur

```
public void actionPerformed(ActionEvent pevent){
        if(pevent.getSource() == this.atimer){
            this.aGui.println("Time is over");
            quit();
            ((Timer)(pevent.getSource())).stop();
        }

    }
```
J'ai voulu l'afficher mais après une longue recherche j'ai décidé d'abandonner cette idée pour des raisons esthétiques

exercice 7.42.2
Pour l'amélioration de l'interface j'ai eu plein d'idée mais faute du temps j'ai pas pu les appliqué .
Ces améliorations seront appliquées pour la version final du projet

exercice 7.43
dans la methode createRooms(), On supprime une sortie dans notre cas c'est celle de vHouse J'ai bloqué la sortie dans cette room car lorsque le joueur entre ici apres avoir realisé toutes les missions (sauvé sa famille …),ci ce n'est pas le cas il a perdu.

exercice 7.44 beamer
Le Beamer permet de se téléporter en le chargeant grace a la commande load et de se teleporter grace a la commande
Fire
Une classe beamer est crée et deux methodes load et teleport sont créées dans la classe GameEngine

exercice 7.45 (optionnel)
Modifications apportées comme demandé j'ai un bug dans la fonction back que j'ai pas pu le résoudre en faite quand je fait back il retourne en arrière deux fois dans le cas ou il vient de traverser une porte verrouillée .

Exercice 7.45.1
Modifications apportées comme demandé.

Exercice 7.45.2
Modifications apportées comme demandé.