–Unité IGI-1102 –

A3P – Apprentissage Par Problème de la Programmation avec Java

Rapport

Projet Zuul

Gélin Antoine

version du 06/06/2017

Groupe : 9

Partie I : Objectif du projet

a) A propos de l'auteur

Ce programme est développé par Antoine Gélin, étudiant en 1ère année à ESIEE Paris

b) Thème du jeu

Le jeu est un jeu d'aventure comportant des énigmes à résoudre afin de résoudre un mystère .

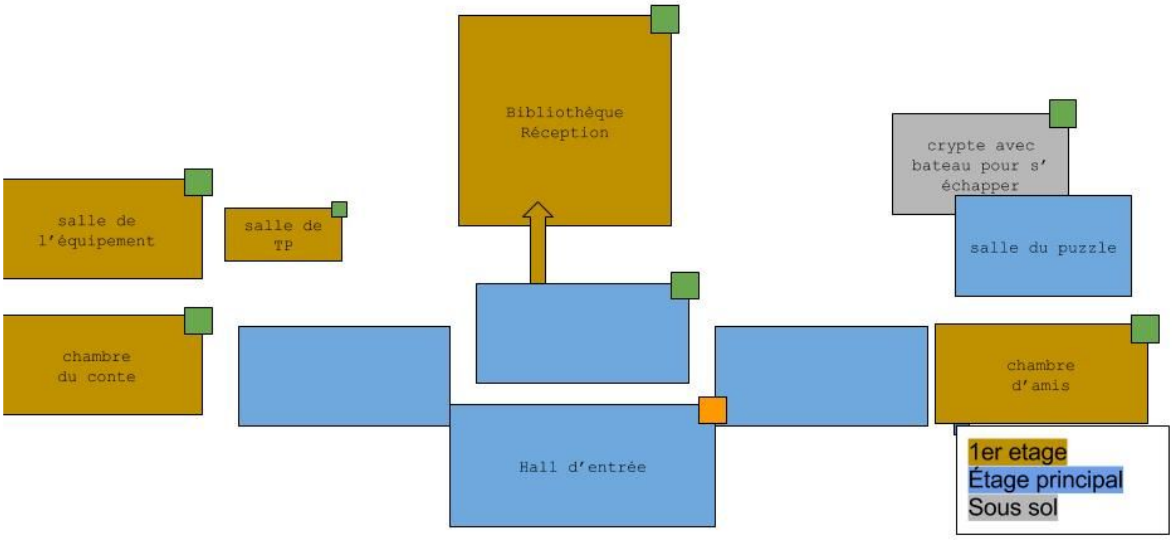
Le jeu est un jeu d'aventure .
La phrase thème telles qu'elle a été retenue est la suivante :
"Dans un manoir sur une île isolée, un journaliste doit enquêter puis s'échapper"
Le jeu se jouera au clavier à l'aide d'une interface agréable d'utilisation.

c) Résumé du scénario

Un journaliste est bloqué par une tempête sur une maison perdue au milieu de nulle part.

Le personnage principal est un journaliste Mike McAra qui doit faire le portrait d'un mystérieux comte sur son île au large de l'Ecosse à la fin des années 1980. Il est contraint de rester dormir dans la demeure de son hôte à cause d'une tempête. Au milieu de la nuit un bruit réveille le journaliste qui trouve le manoir vide est plein de mystères. Il enquête alors jusqu'à ce qu'il découvre qu'il doit s'échapper

d) Plan du jeu


Bibliothèque
Réception
crypte avec
bateau pour s'
échapper
salle de
l'équipement
salle de
TP
salle du puzzle
chambre
du conte
chambre
d'amis
Hall d'entrée
1er etage
Étage principal
Sous sol

e) Scénario détaillé du jeu

Le jeu s’ouvre sur le personnage principal qui se retrouve bloqué par une tempête et doit se réfugier dans la demeure d’un riche et mystérieux individu. Le comte a donc proposé qu’il passe la nuit dans sa chambre d’amis.
Le Journaliste suit donc Comte de la bibliothèque au hall d’entrée et jusqu'à la chambre d’amis. Il rentre dans la chambre et doit interagir avec le lit.
Il se réveille en pleine nuit par un grand bruit. Il va voir la chambre du comte et trouve qu’il n’est pas là. La curiosité journalistique le pousse à chercher des réponses qu’il n’a pu avoir plus tôt dans l’interview.
Il va trouver dans les salles de nombreux indices tels que une chambre secrète avec des gadgets à la James Bond, différents indices prouvant que le comte est un agent infiltré.
Il doit trouver l’indice pour ouvrir la dernière porte verrouillée.

f) Détail des lieux, items, personnages

Personnages :
- Mike McAra, journaliste
- - le comte

Items :
- un cookie
- - une montre
- un bon téléporteur des familles

Lieux :
- 2 chambres
- Une bibliothèque
- une salle secrète pour les accessoires
- une salle secrète pour s'échapper
- de nombreux couloirs

g) Situations gagnantes et perdantes

Situation gagnante : il y arrive et trouve un bateau qu'il utilise pour s'enfuir.

Situation perdante : il n'arrive pas avant la fin du chrono ou ne reussit pas l'épreuve,il est capturé.

h) Énigmes et mini-jeux

- Énigme pour ouvrir le dernier verrou pour s’échapper

II) Réponses aux exercices

Exercice 7.4

Modification des salles pour la mise en place du plan du manoir
Renommage du projet Zuul-bad en Zuul
Réorganisation du projet en déplaçant les fichiers à la racine
suppression des dossiers v1 et veref

Exercice 7.5

utilisation d'une fonction printlocationInfo() dans Room qui permet d'alléger la classe Game
Évite la redondance

Room.java

```
    public void printLocationInfo(){
            System.out.print("Vous êtes dans ");
            System.out.println(this.getDescription());
            System.out.print("Vou
s pouvez aller vers: ");
        String vExits = "";
         if (this.aNorthExit != null){

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

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

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

           vExits += "Ouest ";
           }

+          System.out.println(vExits)  ;
    }
} // Room
```

Exercice 7.6

Implémentation de la fonction getExits() dans Room
Diminue la redondance et augmente le couplage

Room.java

```
public Room getExit(final String pDirection){
        if (pDirection.equals("north")){
            return this.aNorthExit;
        }

        if (pDirection.equals("south")){
            return this.aSouthExit;
        }

        if (pDirection.equals("east")){
            return this.aEastExit;
        }

        if (pDirection.equals("west")){
            return this.aWestExit;
        }

        return null;
    }
```

Exercice 7.7

Création de la fonction getExitString() dans Room qui retourne le String des sorties disponibles
Augmente le couplage et réduit la redondance

Room.java

```
    public String getExitString(){
        String vExits = "";
        if (this.getExit("north") != null){
                vExits += "Nord ";
            }
         if (this.getExit("south") != null){
           vExits += "Sud ";
         }
```

```
        if (this.getExit("east") != null){
          vExits += "Est ";
        }
        if (this.getExit("west") != null){
           vExits += "Ouest ";
          }
```

Exercice 7.8
Réécriture de la façon dont les sorties sont gérées
Utilisation de Hashmap
Récriture de Room
Réécriture de la définition des sorties dans Game

Exercice 7.10
Ajout et génération de la javadoc

Exercice 7.11
Implémentation de getLongDescription() dans Room et Game

Exercice 7.14
implémentation du commandWord look
nouvelle procédure look qui retourne la longdescription de la Room actuelle

Exercice 7.15
Ajout du commandWord eat
Ajout procédure eat qui affiche qu'on a mangé et qu'on a plus faim

Exercice 7.16
Nouvelle méthode pour afficher les commandWords quand on tape help
Va chercher chaque mot du tableau aValidCommands de CommandWords
Nouvelle façon d'afficher le message help
S.o.p est déplacé dans game
les 2 méthodes intermédiaires retournent maintenant une String

Exercice 7.18.5
L'ajout d'une nouvelle salle dans createRooms donne lui à l'ajout de ladite

salle dans une HashMap contenant le nom de la salle et cette dernière.

Exercice 7.18.6

Suite à l'étude du .jar Zuul-with-images avec différentes modifications
Déplacement de l'intérieur de Game dans GameEngine
importation de la nouvelle classe game du jar Zuul-with-Images
changement de la fonction processCommand qui prend maintenant une String en paramètre
Les system.out.print sont maintenant remplacé par des appels à une fonction de la nouvelle classe User Interface créée depuis le jar fourni
Ajout d'images au dossier du projet
Ajout des images à la création des salles
*modification des appels d'import pour importer des classes spécifiques à la place des **

Exercice 7.18.8

Ajout de deux boutons à l'aide de la classe JButton importée à droite et à gauche du texte
Ajout d'un Eventlistener pour capturer le clic sur ces deux boutons
Dans ActionPerformed , différenciation de l'action en fonction que l'on tape du texte ou que l'on appuie sur l'un des boutons
Dans ce dernier cas, on effectue l'action qui est affiché sur le texte du bouton.

Exercice 7.20

Ajout d'une nouvelle classe Item ayant pour attributs la description et le poids
Ajout des deux accesseurs dans cette classe
Ajout dans GameEngine de la déclaration de trois nouveaux objets : une bague une montre et une brique (avec un poids très élevé)
Ajout d'une nouvelle fonction addItem dans Room pour ajouter un objet mettant cet Item dans un nouveau tableau en attribut de Room

GameEngine.java

```
Item vBague = new Item(15,"une dague rouillée");
vHall.setItem(vBague);
```

GameEngine.java

```
public class Item
{
   // instance variables - replace the example below with your own
   private int aPoids;
   private String aDescription;



   /**
    * default constructor for objects of class Item
    */
   public Item(final int pPoids, final String pDescription)
   {
      // initialise instance variables
      this.aPoids = pPoids;
      this.aDescription = pDescription;
   }



   /**
    * accesseur du poids
    * @return     poids de l'objet
    */
   public int getPoids()
   {
      return this.aPoids;
   } // sampleMethod(.)



   /**
    * accesseur e la description
    * @return description
    */
   public String getDescription()
   {
      return this.aDescription;
   } // sampleMethod(.)
} // Item
```

Exercice 7.22
Modification de l'attribut aItem de Room par aItems contenant une map associant la description de l'objet et les Item correspondants
Modification des méthodes get et setItem pour les adapter à la Hashmap
Ajout de deux objets dans l'escalier

Exercice 7.23
Implémentation d'une commande back
Modification des CommandWords valides
Ajout d'un cas "back" dans game engine et d'une fonction back associée
Cette fonction va prendre le dernier élément d'un attribut aRoomS créé et retourner à cette Room.
Création de goToRoom qui prend une room en paramètre et y va
Découplage de goToRoom et goRoom (qui appelle goToRoom)

Exercice 7.28
Ajout d'un dossier tests à la racine du projet
Ajout d'une commande test acceptant uniquement un second mot dans GameEngine et CommandWords
Ajout d'une méthode test activée par la commande test et allant chercher le fichiers de test en .txt dans le dossier tests
Envoie chaque ligne du fichier de test à la fonction InterpretCommand jusqu'à la dernière ligne du fichier.

Exercice 7.28.2
Ajout de des fichiers de test : test-opti.txt et test-complet.txt
Ajout de la fermeture du fichier de test après utilisation par le programme

Exercice 7.29
Ajout de la nouvelle classe Player avec deux attributs : aCurrentRoom stockant la salle où le joueur est actuellement
ainsi que aHistory , stack contenant les room précédentes
Changement des méthodes utilisant la currentRoom dans GameEngine au profit de celles de Player
Ajout également de l'attribut aPlayer dans GameEngine.

Exercice 7.30
Possibilité de prendre et garder des items trouvés dans les salles
Ajout d'une hashmap d'Items dans la classe Player à la manière de ce qui a été fait dans Room
Ajout dans Room et Player d'une fonction TakeItem qui prendra la String du nom de l'Item en paramètre et qui supprimera l'item de la Hashmap d'Items
ajout de nouveaux mots-clef take et drop dans GameEngine et

CommandWords
ajouts des fonctions de take et drop qui vont transférer l'item supprimé dans la room/Player et qui le transférera à l'autre entité (player/room actuelle)

Exercice 7.31.1
Création d'une nouvelle classe ItemList regroupant la gestion des Items des Rooms et Players
Transfert des méthodes de gestion et des Hashmap vers les itemList associés à chaque inventaire
Modification légère de GameEngine pour rediriger les méthodes vers l'itemList correspondante

ItemList.java

```
import java.util.HashMap;
import java.util.Map;
import java.util.Collection;



public class ItemList
{
  // instance variables - replace the example below with your own
  private Map<String, Item> aItems;


  /**
   * default constructor for objects of class ItemList
   */
  public ItemList()
  {
     // initialise instance variables
     aItems = new HashMap<String,Item>();
  }


  public boolean isEmpty(){
     return this.aItems.isEmpty();
  }

  public Collection<Item> getValues(){
```

```
        return this.aItems.values();
    }

    public void setItem(final Item pItem){
        this.aItems.put(pItem.getName(),pItem);
    }



    public Object[] getItems(){
        return this.aItems.values().toArray();
    }

    public Item takeItem(final String vItemString){
        return this.aItems.remove(vItemString);
    }
} // ItemList
```

Exercice 7.32

Ajout d'un poids maximum pour un plyer
Ajout de 2 attributs dans Player: aPoids et aPoidsMax
Ce dernier est pour le moment initialisé à 1000 (en grammes)
Les getters et setters pour ces 2 attributs sont implémentés
Ajout d'un if imbriqué dans game Engine qui repose l'objet et affiche un message si le poids ajouté de l'objet est supérieur au poids max
ajout dans take et drop de l'ajustement du poids de l'inventaire une fois l'objet pris ou reposé

Exercice 7.33

Implémentation d'un inventaire pour le joueur
Déplacement de l'inventaire d'une Room dans la classe ItemList
Utilisation de cette nouvelle méthode dans la classe player pour afficher l'inventaire du joueur
ajout d'un nouveau CommandWord dans CommandWord et dans GAmeEngine
Ajout de la fonction inventaire qui cherche la nouvelle méthode de Player qui vérifie si le joueur a des objets, affiche leur description et affiche enfin le poids total de l'inventaire

Exercice Hors-projet

Ajout d'une Interface simple d'utilisation
Changement du layout de l'UserInterface et basculement sur un boxLayout pour l'interface globale et un CardLayout pour les boutons
Ces derniers s'affichent maintenant sur une ligne et permettent d'en ajouter plus et avoir une plus grande flexibilité

Ajout d'une classe CustomPanel héritant de JPanel pour les images permettant de faire plus de choses à la place du Jpanel
L'image de fond ainsi que les différents Sprites sont faits à l'aide de l'override de la méthode PaintComponent
Cette idée est tirée de https://www.abeautifulsite.net/java-game-programming-for-beginners
Le panel est redimensionné en même temps que la fenêtre et tous les éléments gardent la même place et la même taille relative
La methode scale pour redimensionner à la volée est inspirée par https://stackoverflow.com/questions/11959758/java-maintaining-aspect-ratio-of-jpanel-background-image
La taille préférée du composant est elle aussi redéfinie à la volée à l'aide du override du paintComponent

Ajout d'un principe de Sprites
Les sprites sont des images s'affichent sur l'interface principale (le JPanel customisé) et représentant graphiquement des objets, des personnages
Ajout d'une nouvelle classe Sprite qui sera à terme associée à chaque élément Player et Item
Cette classe est dérivée de la classe cercle du TP2-3 pour les méthodes de déplacement
Elle a pour attributs : un nom, une image qui s'affiche sur l'interface des positions x et y en pourcentage de l'interface (x=0 en haut x=100 invisible en dessous du dessin; y=0 à gauche,y=100 invisible à droite) et une hauteur en pourcentage de la hauteur calculée à partir des dimensions de l'image de fond ainsi qu'un attribut estVisible
Le Sprite est voué à être affiché par dessus le fond si son attribut estVisible est à vrai
Les sprites s'affichent les uns par dessus les autres et sont stockés dans une Hashmap dans le Panel personnalisé
Une fonction est prévu pour réinitialiser cette Map (en cas de changement de Room par exemple)
Les Sprites peuvent se déplacer sur le dessin (immédiatement ou lentement via la fonction Thread.sleep qui arrête pendant 50 ms entre chaque mouvement

Ajout de nouvelles images ainsi que l'icône du jeu qui s'affichera dans la barre de tâche par exemple

Implémentation de l'interface graphique avancées

L'utilisateur peut cliquer pour se déplacer dans chaque tableau et naviguer entre les tableaux et cela déplace le personnage
Les objets sont représentés par leur image propre ou à défaut par l'image de base

La zone des messages a été remplacée par un JTextPanel pour pouvoir le personnaliser
Le texte a maintenant une police propre chargée depuis un fichier .ttf dans l'archive

Exercice 7.34
Ajout d'un cookie magique qui augmente par un facteur de 5 le poids que l'utilisateur peut transporter
Ajout d'un nouveau CommandWord use qui utilise une nouvelle commande use() de GameEngine
La commande use vérifie alors si il s'agit bien d'un cookie et que le cookie est dans la salle
Si c'est le cas il affiche un message et multiplie par 5 le poids maximal
Ce fonctionnement a été retenu au vu des prochaines utilisations possible d'un cas "use" sur les Items

GameEngine.java

```
public void use(final String pItemString){
    if (pItemString.equals("cookie")){
        Item vItem = this.aPlayer.getCurrentRoom().getItems().takeItem(pItemString);
        if ( vItem != null){
            this.aGUI.println("Vous decidez de manger le cookie");
                this.aGUI.println("Cela ne vous fait rien mais vous trouvez derriere le cookie un exosquelette");
            this.aGUI.println("Vous pouvez maintenant porter bien plus d'objets sur vous");
            this.aPlayer.setPoidsMax(this.aPlayer.getPoidsMax()*5);
        }
        else{
            this.aGUI.println("Vous ne trouvez pas l'objet que vous voulez utiliser dans la piece");
            this.aGUI.println("Essayez de le poser si il est dans votre inventaire");
        }
        this.aGUI.setSprites();
    }
```

Exercice 7.34.1
Mise à jour des cas de test pour l'ajout de use cookie

Exercice 7.35
Mise en place du contenu du projet zuul-with-enums
Ajout d'une classe CommandWord au format enums
Modification de la classe CommandWords pour prendre en compte les changements
Modification de GameEngine, Parser et Command pour envoyer et recevoir un commandWord à la place d'une String pour le 1er mot de chaque commande

Exercice 7.35.1
Transformation de la suite de if de processCommand de Game engine en un switch case plus clair

Exercice 7.41
Ajout de la nouvelle conception de zuul-with-enums v2
Permet d'associer à l'enum commandWord un mot que l'on va pouvoir retrouver facilement

Exercice 7.42
Ajout d'une limite de temps (en temps réel)
Ajout dans GameEngine d'un Timer fourni par java Swing
Création d'une nouvelle classe TimerElement héritant de TimerTask et dont la méthode run() est appelée à chaque seconde par le Timer
Cette méthode passe alors le temps restant à GameEngine et UserInterface qui se charge de l'afficher dans un JLabel au dessus des boutons
Le temps est affiché en minutes et secondes

GameEngine.java

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



   public void createTimer(){
      UserInterface vGUI = this.aGUI;
```

```
    final Timer timer = new Timer();
    TimerElement vTT = new TimerElement();
    vTT.pass(this, timer);
    timer.scheduleAtFixedRate(vTT, 0, 1000);


}
public void updateTime(final int pTime){
    this.aGUI.updateTimeGUI(pTime);
}


public int getTime(){
    return this.aTime;
}
```

TimerElement.java

```
import java.util.Timer;
import java.util.TimerTask;


public class TimerElement extends TimerTask
{
  private int aTime;
  private GameEngine aGE;
  private Timer aTimer;

  public TimerElement(){
      this.aTime = 150;

  }

  public void pass(final GameEngine pE , final Timer pT){
      this.aGE = pE;
      this.aTimer =pT;

  }


  @Override
  public void run() {
```

```
        this.aTime -= 1;

        this.aGE.updateTime(this.aTime);


        if (this.aTime == 0){
            this.aTimer.cancel();
            this.aGE.gameOver();
        }
    }
}
```

Exercice 7.43

Le joueur ne peut passer la porte d'entrée principale du manoir que dans un seul sens
Si il essaye de franchir la porte dans l'autre sens, le jeu lui indiquera qu'il n'y a pas de porte
Si il essaye par la commande BACK, on lui dira qu'il est coincé dans le manoir et qu'il ne peut faire demi tour

GameEngine.java

```
public void back(){
    if (! this.aPlayer.getHistory().empty()) {
        Room vRoomPre = this.aPlayer.getHistory().pop();
        if (vRoomPre.getDescription().equals("Entrée du manoir")){
            this.aGUI.print("Vous ne pouvez pas revenir plus loin ");
            this.aGUI.println("car la porte s'est refermée quand vous êtes entré");
        }
        else{
            goToRoom(vRoomPre);
        }
    }
    else this.aGUI.println("Vous ne pouvez pas revenir plus loin");
}
```

Exercice 7.44

Ajout d'un "beamer" au jeu
Il s'utilise par la commande use introduite précédemment

On introduit aussi le principe d'un attribut d'état des Items par défaut un int à 0 ainsi qu'un Objet Java générique en tant qu'attribut de variable d'état.

Le programme va si on utilise un objet ayant un nom beamer va vérifier la variable d'état de l'objet.
Si elle est à 0, il va dire qu'il charge le beamer, mettre la Room actuelle dans la variable d'état et mettre l'état de l'objet à 1
Si elle est à 1 (beamer chargé), on va chercher la room dans la variable d'état de l'objet , la caster en room, se déplacer dans cette room, remettre à 0 la commande back pour éviter les bugs et mettre l'etat du beamer à 0 (non chargé)

Le beamer s'utilise quand il est au sol mais on peut le prendre et le joueur se téléporte avec le beamer au sol dans la room chargée précédemment
J'ai choisi cette approche de variable d'état et d'état afin de conserver une modularité de mon jeu et éviter la multiplication des mots clefs (charge, beam) pour l'utilisateur et l'interface
Cela me permet de garder la même logique d'utilisation des objets tout au long du jeu

GameEngine.java

```
public void use(final String pItemString){
      if (pItemString.equals("cookie")){
    [...]
}



      if (pItemString.equals("beamer")){
         Item vItem = this.aPlayer.getCurrentRoom().getItems().takeItem(pItemString);
         if ( vItem != null){
            if (vItem.getState() == 0){
               this.aGUI.println("Ce beamer est maintenant chargé");
               this.aGUI.println("Si vous l'utilisez encore il vous ramènera ");
               this.aGUI.print("instantanément dans la pièce actuelle");
               this.aGUI.println("\n CET OBJET EST SECRET DEFENSE");
               vItem.setState(1);
               vItem.setStateVar(this.aPlayer.getCurrentRoom());
            }
            else if (vItem.getState() == 1){
               this.aGUI.println("BEAMER ACTIVÉ");
               this.aGUI.println("\n TÉLÉPORTATION EN COURS");
               Room vRoom = (Room)vItem.getStateVar();
               goToRoom(vRoom);
```

```
            this.aPlayer.resetHistory();
            vItem.setState(0);
        }

        this.aPlayer.getCurrentRoom().getItems().setItem(vItem);
```

Exercice 7.46

Ajout d'une TransporterRoom

On ajoute un nouveau Type de Room, une TransporterRoom (classe à part héritant de Room)

Cette Room va téléporter le joueur qui rentre dedans dans une autre salle au hasard parmi un ensemble fourni dans le constructeur

La méthode getExit de cette classe à été Overwritten pour que si on aille vers l'ouest, la sortie renvoyée soit une salle au hasard

Le choix aléatoire est fait par la classe java.util.Random et sa méthode nextInt() qui renvoie un nombre au hasard entre 0 inclus et le nombre maximal décidé exclus

Ce nombre aléatoire n est ensuite utilisé pour prendre la n-ième valeur de l'Array des salles disponibles

C'est ce que la méthode goRoom() renvoie alors

TransporterRoom.java

```
import java.util.Random;


public class TransporterRoom extends Room
{
   private Room aPseudoRandom;
   private Room[] aAvailableRooms;
```

```
public TransporterRoom (final String pDesc, final String pImg, final Room[] pARooms)
{
    super(pDesc,pImg);
    this.aPseudoRandom = null;
    this.aAvailableRooms = pARooms;
}


@Override
/**
 * accesseur de la direction
 * @param pDirection la direction
 * @return la room qui fait face à cette direction
 */
public Room getExit(final String pDirection){
    if (pDirection.equals("west")){
        return findARandomRoom();
    }
    else{
        return null;
    }
}

@Override
public String getExitString(){
    return "une salle au hasard dans la maison à portée de l'appareil";
}


public Room findARandomRoom(){
    if (aPseudoRandom == null){
        Random vRandomObject = new Random();
        int vI = vRandomObject.nextInt(this.aAvailableRooms.length); //DERNIER NON INCLUS
        return aAvailableRooms[vI];
    }
    else{
        return this.aPseudoRandom;
    }
}


public void setPseudoRandom(final Room pR){
    this.aPseudoRandom = pR;
```

```
  }


}
```

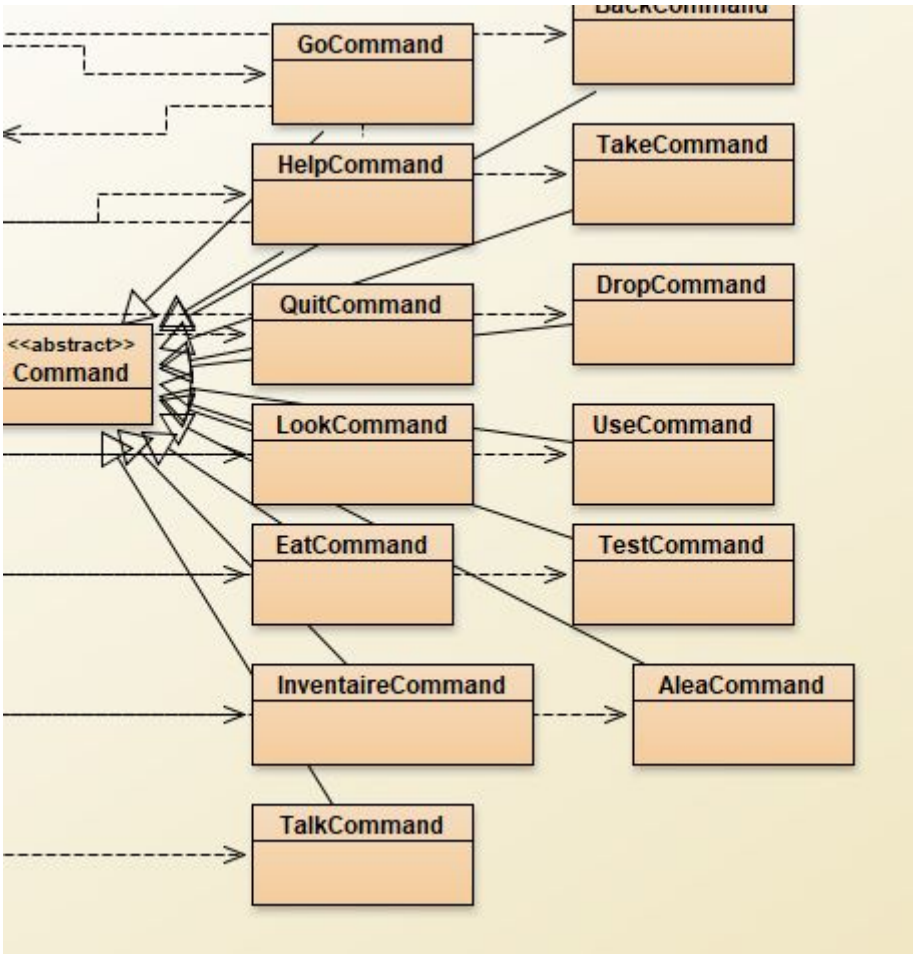
Exercice 7.46.1

On implémente un attribut aPseudoRandom tel que s'il est différent de null (sa valeur par défaut) il court-circuite le raisonnement précédent et c'est ce que goRoom renvoie
Ajout d'un commandWord alea dans CommandWord et dans gameEngine qui appelle une nouvelle méthode alea() qui prend le second mot de la commande
Ce second mot est le nom de la salle tel que défini dans la HashMap aRooms et dans createRooms()
la méthode va chercher s'il existe une salle de ce nom dans la hashmap et si oui la passer à la méthode setPseudoRandom de la TransporterRoom créée précédemment
La TransporterRoom ne cherchera donc aucun mécanisme aléatoire pour sa sortie mais redirigera vers la salle indiquée
Un message indique lors de l'utilisation de la méthode si la salle n'a pas été trouvée

Exercice 7.47

Changement de la logique de la gestion des commandes
On remplace la longue séquence de ifs de GameEngine par des sous classes de Command

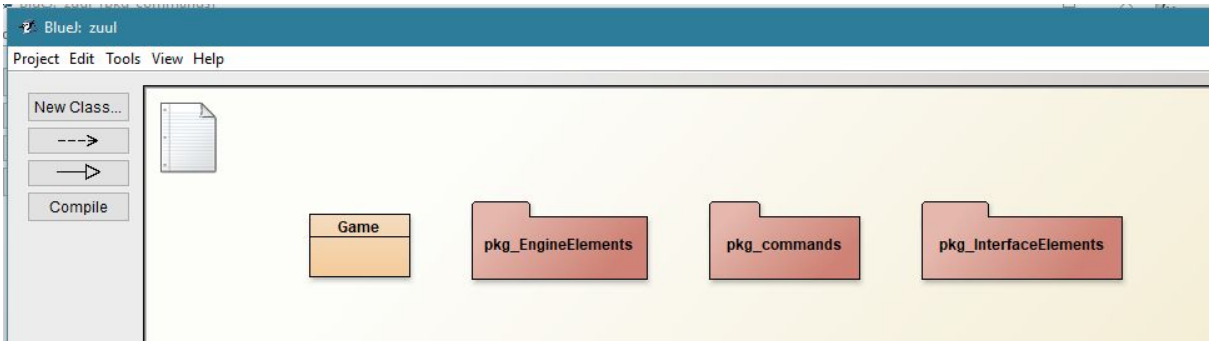
La fonction interpretCommand de Parser va maintenant associer le premier mot de la commande à un des CommandWords qui sera lui même associé à une nouvelle commande (GoCommand, HelpCommand, etc ...)
interpretCommand va alors utiliser la fonction execute() de la nouvelle Command
Dans cette command est stockée les instructions spécifique au type de la commande précédemment stockés dans GameEngine
Le fonctionnement du programme est le même mais la visibilité et facilité de relecture sont grandement accrus.



Exercice 7.47.1
Organisation en structure de packages / sous dossiers
Il y 3 nouveaux packages :

- *pkg_EngineElements (elements du moteur structurel du jeu)*
- *pkg_InterfaceElements (éléments de l'interface graphique)*
- *pkg_commands (regroupe la gestion des commandes)*

La visibilité du projet en est grandement accrue



Exercice 7.48

Ajout d'un concept de personnage non joueur (PNJ)
Le PNJ (PNJElement) est un héritage d'un Item qui peut parler
Il a en plus des attributs des items un tableau de String contenant les différents dialogues qu'il peut avoir
Il se place dans une Room à l'aide de la nouvelle ItemList aCharacters de chaque Room.
(on les met à part des Items afin d'éviter de pouvoir prendre une personne par exemple)

La longDescription fournie par la Room indique elle aussi maintenant les personnages présents dans un message à part
Ajout d'un CommandWord et d'une Command Talk afin de pouvoir parler avec eux
cette commande appelle la commande talk du personnage correspondant

Exercice 7.49

Ajout d'un pattern de déplacement du PNJ défini à l'initialisation dudit PNJ
A chaque fois que l'on parle au PNJ, on vérifie si la salle de destination n'est pas nulle
Si ce n'est pas le cas, cela veut dire que le PNJ doit se déplacer dans cette salle à la fin de sa phrase
Il appelle alors la methode movePNJRoom du GameEngine afin que le PNJ soit déplacé

GameEngine.java

`String[] vDialC = new String[] {"bonjour et bienvenue dans mon chateau",`
`"Vous pouvez me suivre dans la bibliotheque",`
`"Vous pouvez rester dans votre chambre le temps que l'orage cesse"};`

```
      Room[] vCPattern = new Room[] { null, vBibliotheque, null};
       PNJElement vPNJComte = new PNJElement("comte", "le comte du chateau", vDialC,
vCPattern, this);
vPNJComte.setRoom(vHall);

  [...]

public void movePNJRoom(final PNJElement pPNJ ,final Room pR){
     ItemList vIL = pPNJ.getRoom().getCharacters();

     vIL.takeItem("comte");
      System.out.println(vIL.getItemArray().toString());
      pR.getCharacters().setItem(pPNJ);
      this.aGUI.setSprites();
  }
```

PNJElement.java

```
package pkg_EngineElements;



public class PNJElement extends Item
{
   // instance variables - replace the example below with your own
   private String[] aDialogues;
   private int aDialInt;
   private Room[] aDepPattern;



   private GameEngine aGE;
   private ItemList aItems;
   private Room aRoom;



   /**
    * Constructor for objects of class Character
    */
      public PNJElement(final String pName, final String pDescription, final String[]
pDialogues, final Room[] pDepPattern, final GameEngine pGE)
   {
      super(9999,pName, pDescription);
```

```
        this.aDialogues = pDialogues;
        this.aDepPattern = pDepPattern;
        this.aGE = pGE;
        this.aDialInt =0;
    }


    /**
     *
     */
    public String talk()
    {
        if (this.aDialInt <= this.aDialogues.length)
        {
            this.aDialInt ++ ;
        }


        if (this.aDepPattern[this.aDialInt-1] != null){
            Room vR = this.aDepPattern[this.aDialInt-1];
            this.aGE.movePNJRoom(this,vR);
        }


        return this.aDialogues[this.aDialInt -1];
    }


    public void setRoom(final Room pRoom){
        this.aRoom = pRoom;
    }


    public Room getRoom(){
        return this.aRoom;
    }


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

Exercice 7.53
Ajout de la méthode main

Exercice 7.58
Mise à jour du .jar et de certaines parties du programme pour lui permettre d'être lancé depuis le jar

III. Mode d'emploi

Le jeu se lance grâce au .jar ou à la méthode statique de la classe Game (à la racine)

Le beamer et le cookie peuvent être trouvés dans la salle en haut à gauche de la carte (le laboratoire) et s'utilisent tous les deux avec la commande "use".

La commande test s'utilise avec "test test-complet" ou "test test-opti" pour le chemin rapide

Le but du jeu est d'arriver en haut à gauche de la carte.

Le jeu peut se jouer tel que décrit dans les différents exercices du projet ou par un système "point and click" en fonction de la position des clics de souris

IV) Déclaration anti-plagiat

Aucune partie de code autre que la bas *Zuul-Bad* n'a été plagiée.
Tout le code est original ou suggéré par les exercices sauf les parties suivantes :

Dans la classe UserInterface

Pour l'utilisation d'une police personnalisée :

Merci à 3ph3r du forum StackOverflow :
https://stackoverflow.com/questions/24800886/how-to-import-a-custom-java-awt-font-from-a-font-family-with-multiple-ttf-files

Dans la classe UserInterface

Pour l'utilisation de l'override du PaintComponent d'un JPanel

Merci à Cory LaViska du site aBeautifulSite.net
https://www.abeautifulsite.net/java-game-programming-for-beginners

Pour faire en sorte que l'image de fond soit redimensionnée dynamiquement

Merci à MadProgrammer du forum StackOverflow :
https://stackoverflow.com/questions/11959758/java-maintaining-aspect-ratio-of-jpanel-background-image

Pour retrouver la position d'un clic de souris sur un panel et plus généralement pour implémenter la gestion d'un clic de souris:

Merci à David Kroukamp du forum StackOverflow :
https://stackoverflow.com/questions/12396066/how-to-get-location-of-a-mouse-click-relative-to-a-swing-window