Rapport projet IPO :

Titre : Alerte au pied de l’arc en ciel !

lien pour la page web : https://perso.esiee.fr/~hermannm/

I. A. Auteur :

Manon HERMANN

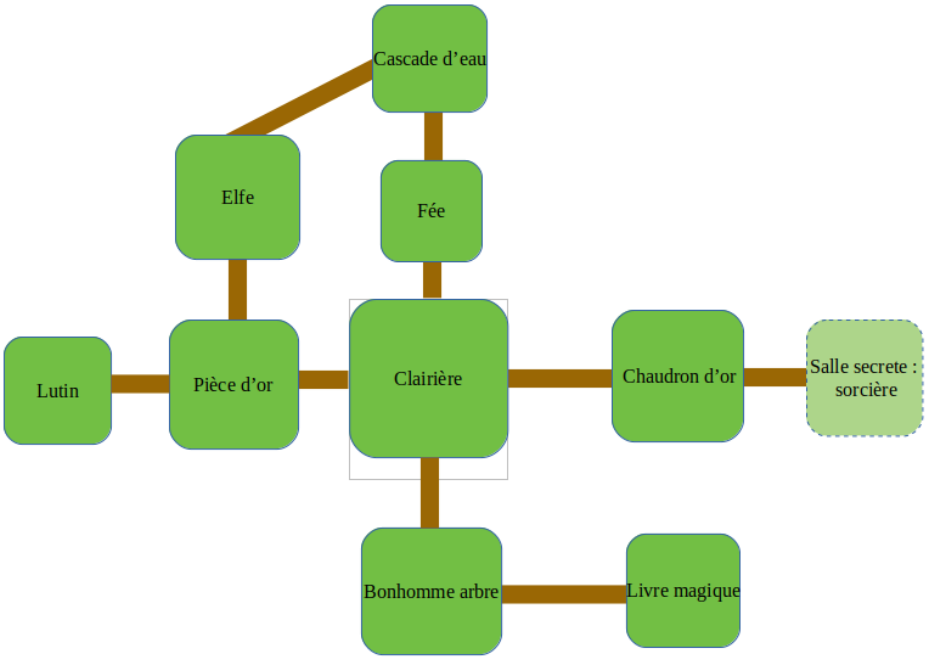
B. Thème :

Dans une forêt magique, un farfadet doit retrouver les pièces volées au pied de l’arc en ciel

C. Résumé du scénario (complet) :

Dans une foret, un farfadet est appelé d'urgence au pied d'un arc en ciel car des pièces d'or du chaudron ont été volées ! Il devra alors traverser la forêt pour tenter de retrouver 5 pièces d'or perdu. Au cours de son chemin il rencontrera des personnages qui l'aideront en échange d'un autre objet... Mais lorsqu'il aura enfin trouvé toutes les pièces il se rendra compte qu'il lui manque la clé pour refermé le chaudron : quelqu'un l'a volé ! Il faut retrouver cette personne.

D. Plan complet :



E. Scénario détaillé :

Vous incarnez un farfadet qui doit retrouver les pièces d'or qui ont disparus dans le chaudron au pied d'un arc en ciel. Elles sont dispersées dans la forêt. La 1ère vous est donnée par la fée après un court dialogue. La 2sd vous est donnée par l'elfe une fois que vous aurez retrouver son livre magique placer dans une autre salle. La 3ème est posé par terre dans une salle, il suffit de la ramasser. La 4ème vous est donnée par le lutin une fois que vous lui serez aller chercher deux champignons disponible dans une autre salle. La 5ème vous est donnée par l'arbre vivant une fois lui être chercher un verre d'eau trouvable dans une autre salle.
Une fois ces 5 pièces récoltées il faudra retourner au chaudron d'or pour les déposées. C'est à ce moment que vous vous rendez compte que la clé pour fermer ce chaudron a aussi disparu. Une salle se débloque à gauche du chaudron. C'est dans cette salle que se trouve une sorcière détenant la clé. Elle vous posera une question, si la réponse est juste alors vous gagner le jeu, sinon vous perdez.

F. Détails des lieux, items, personnages :

Chaque lieu contient un personnage avec lequel le joueur peut interagir ou un objet qu'il peut ramasser.
Liste des personnages : Fée, Lutin, Homme Arbre, Elfe, Sorcière
Liste des items : 5 pièces d'or, la clé du chaudron, 2 champignons, 1 verre d'eau, 1 livre magique
Liste des lieux : - « Chaudron d'or » où le joueur commence le jeu, c'est ici qu'il devrait rapporté les 5 pièces d'or ;
- « Sorcières » c'est la salle secrète du jeu qui se déverrouille lorsque le joueur à fini de rapporter toutes le pièces au chaudron ;
- « Clairière » on peut y trouver des champignons ;
- « Fée » on peut parler avec une fée qui nous donne une pièce d'or ;
- « Cascade d'eau » on peut y récolter de l'eau ;
- « Elfe » on peut parler avec un elfe qui nous demande de retrouver son livre magique contre une pièce d'or ;
- « Pièce d'or » c'est l'endroit ou on peut ramasser une pièce d'or tombé a terre ;
- « Lutin » on peut parler avec un lutin qui nous demande de lui trouver 2 champignons en échange d'une pièce d'or ;
- « Bonhomme arbre » on peut parler avec un arbre vivant qui manque d'eau, pour vous remercier de l'eau que vous lui donnerez il vous offrira une pièce d'or ;
- « Livre magique » c'est l'endroit où vous pouvez ramasser le livre magique de l'elfe ;

G. Situations gagnantes et perdantes :

Il est possible de gagner lorsque les 5 pièces d'or sont récupérées et ramenées au chaudron d'or (dans le temps imparti) et que le joueur saura répondre juste a l'énigme
Il est possible de perdre si le joueur ne récupère pas toute les pièces dans le temps imparti, ou s'il n'a pas le temps de les ramener au chaudron d'or, ou s'il ne répond pas correctement a l'énigme

H. Éventuellement énigmes, mini-jeux, combats :

Une énigme sera posé par la sorcière pour qu'elle veuille bien vous rendre la clé du chaudron d'or : « Combien de personnages avez-vous rencontrer dans ce jeu ? »
La bonne réponse est « 4 ».

II. Réponses aux exercices :

Exercice 7,0 (optionnel) : Pour la création de la page web du projet, j'ai créer un dossier public_html dans lequel il y a un document .html. Pour faire un site web avec une belle mise en forme, j'ai choisi de prendre un template sur internet (cf référence dans les sources). Ayant quelque base en langage HTML et CSS, j'ai pu modifié certains paramètres pour qu'il soit à l'image de ce que je voulais représenter.

Exercice 7,4 : Les 5 classes des TP 3,1 et 3,2 ont été copier dans un nouveau projet BlueJ. Les test relative aux différentes classes ont disparus, et celles-ci sont maintenant à la racine du projet.

Exercice 7,5 : Dans cet exercice on souhaite évité la duplication de code notamment pour les méthodes printWelcome et goRoom (classe Game) qui contiennent chacune un bout de code permettant d'afficher les sorties valides de la pièce où se trouve le joueur. Pour cela, on crée dans la classe Game une méthode printLocationInfo qui reprend exactement ce que faisant ces dernières méthodes en double. Puis dans printWelcome et goRoom on supprime cette duplication pour appeler cette nouvelle méthode.

Exercice 7,6 : Le jeu fonctionne correctement, mais il y a du couplage de code notamment pour définir les sorties des pièces courantes. Nous allons donc créer un accesseur *getExit* dans *Room* qui permettra d'obtenir directement ces informations. Donc il faut remplacer les morceaux de code en appelant l'accesseur comme par exemple <nextRoom = currentRoom.getExit(« East »);>. Ainsi de grosse partie de code peuvent être remplacée par une seule ligne de code.

Exercice 7,7: Par la suite nous créerons une méthode *getExitString* dans *Room* qui affichera les sorties possibles de la pièces courantes. Il faudra donc l'appeler dans *printLocationInfo* de la classe *Game* pour qu'à chaque entré dans une nouvelle pièce le joueur sache quelles sorties sont possible.

*Exercice 7,8 :*Concept de la HashMap pour simplifier les déplacements directionnels et réduire les erreurs de codage. Il faut tout d'abord importer (grâce à la java doc) la HashMap. Il faut ensuite l'initialiser avec les attributs ainsi que dans le constructeur. Mais cette création de HashMap entraîne une modification dans d'autres méthodes de *Game* et *Room*. Ainsi le *setExit* créé précédemment peut-être résumé en une ligne grâce aux fonctionnalités de la HashMap <exits.put(pDirection, pRoom);>.

Exercice 7,8,1 : Ajout d'un déplacement vertical dans le jeu. Dans mon cas j'ai choisi la possibilité d'accéder à la pièce *Lutin* en aller vers le haut.

Exercice 7,9 : Ici il va falloir redéfinir les pièces (les instanciés de nouveau) car la HashMap facilite leur définition. Il faudra aussi modifié *getExitString* en se servant des « key » de la HashMap. Ainsi on utilisera la méthode *keySet* pour aller chercher les sorties associées à la pièce courante.

*Exercice 7,10 (optionnel) : L*a méthode getExitString de la classe Room a été crée pour éviter la duplication de code dans la classe Game. Elle renvoie les sorties possibles de la pièces où le joueur se trouve. Elle fonctionne grâce aux propriétés de la HashMap, c'est-à-dire qu'on va lui envoyer la position du joueur (la pièce associée) pour quelle renvoie la clé associée permettant ainsi d'aller voir à cet emplacement. On y trouvera les sorties reliées.

Exercice 7,10,1 : Mise a jour de la javadoc pour chaque classe et méthode de celle-ci. On y indique a quoi sert la classe/méthode, puis pour chaque méthode quel(s) paramètre(s) i faut lui assigné ainsi que ce quelle retourne

Exercice 7,10,2 : Génération et visualisation de la javadoc grâce a BlueJ, elle comprend bien tout ce qui a été renseigné.

Exercice 7,11: Nous allons maintenant anticiper les prochains TP en considérant que les pièces auront des items et/ou personnes. Il faudra donc le dire dans la description (lorsqu'on arrive dans la pièce), pour cela on créé une méthode *getLongDescription* dans la classe *Room* qui ajoute ces détails. On va donc appeler cette méthode dans la méthode *printLocationInfo* de la classe *Game.*

Exercice 7,12 (optionnel) : x
Exercice 7,13 (optionnel) : x

Exercice 7,14 : On veut créer une nouvelle fonctionnalité dans le jeu, l'acteur de regarder. Pour cela il faut tout d'abord mettre à jour le tableau constant de *CommandWords* qui répertorie toutes les commandes valides du jeu. Ensuite, il faut logiquement la créer dans *Game*, elle renverra la méthode *getLongDescription*. Et la rajouter dans le répertoire des commandes de la méthode *processCommand*.

Exercice 7,15 : De même, on veut faire une commande pour manger. Il faut procédé de même que l'exercice précédent mais avec une nouvelle action *eat*.

*Exercice 7,16 :*On remarque *printHelp* n'est pas à jour vis a vis des nouvelles commandes possibles, on pourra les rajouter à la main mais il faudrait le faire à chaque fois, et cela induit plusieurs erreurs possible. On va donc créer une méthode *showAll* dans la classe *CommandWord* qui affiche les commandes valides (à partir du tableau constant). Or cette méthode n'est pas directement accessible depuis la classe *Game*, il faut donc créer une méthode intermédiaire *showCommands* dans la classe *Parser*.

*Exercice 7,17 (optionnel) :*Pour ajouter une nouvelle commande, il faut forcement changer la classe *Game*. En effet, il faut la définir dans cette classe et ajouter sa référence dans la méthode *processCommand*.

*Exercice 7,18 :*Pour mieux organisé l'implantation du code et rétablir la conception de base de certaines méthode. Nous allons modifié la méthode *showAll* en *getCommandWord* pour quelle renvoie une String et ne s'occupe pas de la renvoyer. Ainsi ça sera *printHelp* de *Game* qui se chargera de l'afficher. Il faut donc en conséquence modifié la méthode intermédiaire de la classe *Parser* en la faisant renvoyer une String.

Exercice 7,18,1 : Ici, il faut comparer notre projet actuel avec celui de *zuul-better*. Ils correspondent bien et aucun exercices n'a été sauté ou mal fais.

Exercice 7,18,2 (optionnel) : x

Exercice 7,18,3 : Des photos 800x600 ont été trouvé pour chaque pièce du jeu incluant ou non des personnages.

Exercice 7,18,4: Le titre du jeu sera «Alerte au pied de l'arc en ciel».

Exercice 7,18,5 : Création d'une HashMap contenant toutes les Room. Il a donc fallu la créer dans le constructeur de la classe GameEngine et les initialiser dans la méthode *createRoom()* en faisant pour toutes les salles : `aRoomsHM.put(vChauron, « caldron'room ») ;`

Exercice 7,18,6 : etude du Zuul-with-images, les principales changement ont été sur l ‘emplacement de certaines méthodes, notament entre *Game* et *GameEngine.*

Exercice 7,18,7 (optionnel) : x

*Exercice 7,18,8 : l’a*jout d’un bouton s’opère par différentes étapes : création du bouton help dans la méthode *createGUI()* ; indication de sa position dans la fenêtre de jeu *vPannel.add(this.aButton, BorderLayout.EAST)*; ; ajout de l’action Listener pour les boutons ; dans la méthode *actionPerformed()* écrire ce que le bouton doit effectuer.

Exercice 7,19 (optionnel) : x
Exercice 7,19,1 (optionnel) : x

Exercice 7,19,2 : Toutes les images relative au jeu ont été déplacé dans un dossier appelé *Images*

Exercice 7,20 : Création d’une nouvelle classe *Item* avec deux attributs correspondant à une description et à un poids. Modification de la méthode *getLongDescription()* pour qu’elle renvoie aussi les informations relatives à l’objet dans la salle courante. Ajout d’un attribut Item a la classe *Room*. Modification de l’initialisation des room pour qu’elle ai un Item, mettre null si aucun item est dans cette salle.

Exercice 7,21 : Ajout d’une méthode *getItemDescritpion()* qui revois la description de l’item ainsi que son poids. La description d'un Item doit être produite par l’Item lui-même. La description longue d'une Room devant indiquer l'Item qui s'y trouve éventuellement, la classe Room devra être modifiée. Ces descriptions doivent être affichées à chaque fois que nécessaire par le GameEngine

Exercice 7,21,1 (optionnel) : x

Exercice 7,22 : création d’une HashMap pour pouvoir stoker plusieurs objets. Et création de la méthode *addItem()* qui place l’item dans la room adapté. Suppression du 3eme attribut de *Room*, modifications des méthodes en conséquences.

Exercice 7,22,1 (optionnel) : x

Exercice 7,22,2 : des objets sont ajouté au jeu, dont obligatoirement un dans la pièce initiale et une pièce contenant deux objets :

Objet :	**Lieu :**
Piece d’or 1	Foret
Piece d’or 2	Donné par la fée
Piece d’or 3	Donné par l’elfe
Piece d’or 4	Donné par le lutin
Piece d’or 5	Donné par l’arbre vivant
Eau	Cascade
Livre magique	Foret (lieu éloigné)
Champignons	Clairière
Clé	Donné par la sorcière
Chaudron magique	Chaudron

Fleur	Foret (lieu de l'elfe)

Exercice 7,23 : Création de la commande *back*. Pour cela, il faut d'abord la rajouter dans le tableau des commandes valides, puis la créer dans la classe *Room* en ajoutant un attribut *aCurrentRoom* qui prendra la valeur de la pièce où nous étions.

Exercice 7,24 (explication optionnel) : x
Exercice7,25 (explication optionnel) : x

Exercice 7,26 : Pour faire la fonction back() de façon plus optimale, nous allons utiliser une Stack pour stocker les différentes salles qu'aura parcourus le joueur. Les methodes utilent a connaître pour une Stack sont : push(element) qui empile element sur la Stack
pop() qui retourne l'élément du dessus de la Stack ET le supprime de la Stack
empty() qui permet de savoir si la Stack est vide
peek() qui retourne l'élément du dessus de la Stack sans le supprimer
Nous avons donc créé la méthode suivante :

```
private void back(){
     this.aCurrentRoom =this.aRoomStack.pop();
     this.aGui.println(this.aCurrentRoom.getLongDescription());
     if(this.aCurrentRoom.getImageName() != null)
          this.aGui.showImage(this.aCurrentRoom.getImageName());
}
```

Et il faudra bien evidement rajouter la piece dans laquelle nous etions. Pour cela il faut dans la methode *goRoom()* rajouter une ligne avant de changer de salle pour stoker la valeur dans la Stack :

```
this.aRoomStack.push(this.aCurrentRoom);
```

*Exercice 7,26,1 :*Generation des deux javadocs grace aux deux commandes :

```
javadoc -d userdoc -author -version *.java
javadoc -d progdoc -author -version -private -linksource *.java
```

Exercice 7,27 : Faire un document texte pour tester certaines commandes

Exercice 7,28 : Creer le document test avec quelques commandes pour essayer les fonctionnalité du jeu (comme look(), back(), eat(), help(), …)

Exercice 7,28,1 : Creer cette nouvelle commande *Test()* qui permettra de tester le document texte ecrit apres la commande :

```
private void test (final String pNom) {
     Scanner vS;
     try {
          String vNom =pNom;
          if (! vNom.endsWith(".txt")) vNom += ".txt";
          InputStream vIPS = getClass().getResourceAsStream(vNom);
          vS = new Scanner (vIPS);
          while ( vS.hasNextLine() ) {
                String vLine = vS.nextLine();
                this.interpretCommand(vLine);
          }
    }
```

```
        catch (final Exception pErr) {
            this.aGui.println ("Erreur : " + pErr.getMessage() );
        }
    }
```

Exercice 7,28,2 : Création de deux fichiers : une version longue (pour visiter toute les salles) et une version nous permettant de gagner au jeu

Exercice 7,28,3 : x

Exercice 7,29 : Nous allons maintenant créer une nouvelle classe *Player* qui permettra de gérer le joueur, et par soucis de cohérence faire en sorte que les actions telles que *look()* ou *eat()* soient faite par un joueur. Il faudra aussi transferer tout les méthodes de *GameEngine* contenant *aCurrentRoom* dans la nouvelle classe *Player.* Il faudra tout de même faire attention à ce que *GameEngine* garde le contrôle sur la système d'affichage.

Exercice 7,30 : Nous allons maintenant faire deux méthodes *take()* et *drop()* qui permettrons de prendre et poser un item. Pour cela il faut créer un attribut *Item* qui sera l'item porter (ou non) par le joueur initialiser à null. Dans la méthode *take(),* qui comportera un paramètre, on modifiera la valeur de notre attribut en y plaçant le nom de l'item pris. Pour la méthode *drop(),* qui comportera un paramètre, on remettra l'attribut à null tandis qu'on ajoutera l'item à la salle courante.

Exercice 7,31 : Pour que cela soit plus intéressant, on va faire en sorte que le joueur puisse porter plusieurs items. Pour cela il suffit de faire une HashMap des differents Items que porte le joueur. Les méthodes *take()* et *drop()* sont à modifier en conséquences.

Exercice 7,31,1 : Création d'une nouvelle classe *ItemList* pour gérer plus facilement la liste d'items que porte le joueur. Nous utiliserons une HashMap pour lier les items à leur nom. Il faut créer les accesseurs pour retrouver les clés de la HashMap. On decide aussi le créer une methode *getItemString()* pour avoir a liste des items portés par le joueur. Ainsi que deux méthodes *addItem()* et *removeItem()* qui servira lorsque le joueur appelera la méthode *take()* et/ou *drop(),* qu'il faut aussi changer ne conséquences.

```
public class ItemList
{
    public HashMap <String, Item> aInventoryHM;

    public ItemList () {
        this.aInventoryHM = new HashMap <String, Item> ();
    }

    public void removeItem (final String pName, final Item pItem){
        this.aInventoryHM.remove(pName, pItem);
    }
    public void addItem (final String pName, final Item pItem) {
        this.aInventoryHM.put(pName, pItem);
    }
}
```

et dans la classe *Player* :

```
public void takeItem (final Command pCommand) {
     String vDescr = pCommand.getSecondWord();
     Item vItem = this.aCurrentRoom.aItemHM.get(vDescr);
     if (vItem == null)
         this.aGui.println ("this object do not exist");
     else if (this.aPoidsMax < this.aInventory.getTotalWeight() +
               vItem.getWeight())
          this.aGui.println ("your inventory is too heavy");
          else {
            this.aInventory.addItem(vDescr, vItem);
            this.aCurrentRoom.aItemHM.remove(vDescr, vItem);
            this.aGui.println (this.aInventory.getItemsString());
          }
}

public void dropItem (final Command pCommand) {
     String vDescr = pCommand.getSecondWord();
     Item vItem = this.aInventory.aInventoryHM.get(vDescr);
     if (vItem == null)
         this.aGui.println ("can't drop this object");
     else {
         this.aInventory.removeItem(vDescr, vItem);
         this.aCurrentRoom.aItemHM.put(vDescr, vItem);
     }
     this.aGui.println (this.aInventory.getItemsString());
}
```

Exercice 7,32 : Nous va maintenant incorporer le fait qu'un item peut avoir un poids. Pour cela il suffi de rajouter un attribut *aWeight* dans la classe *Item* et de faire un accesseurs pour avoir celui-ci. On rajoutera juste pour que cela soit plus pratique, une méthode *getItemInformation()* permettant obtenir le nom, la description et le poids de l'item.

Exercice 7,33 : Création de l'inventaire. Il faut donc ajouter une commande *inventory* qui renvera la liste des objets et de leur descriptions lorsqu'elle sera ecrire. On fait donc une méthode *getItemString()* dans *ItemList* pour obtenir tout les items portés, on fait de même pour avoir le poids total.

```
public String getItemsString(){
        if (this.aInventoryHM.isEmpty()) {
            return "inventory is empty !";
        }
        StringBuilder vList = new StringBuilder("inventory : ");
        Set <String> keys = this.aInventoryHM.keySet();
        for(String vItem : keys){
            vList.append(" " + vItem);
        }
        return vList.toString();
```

Puis une méthode *inventory()* reprenant ces deux methodes.

Exercice 7,34 : On souhaite faire un item « magique » qu'on appellera *magicCookie*. Comme pour les autres items, on le créé et le place dans une salle. La chose particulière à faire est de modifier la fonction *eat()* dans *Player* pour que nous puissons le manger. Ainsi la méthode devient :

```
public void eat (final String pCommand){
     Item vItem = this.aInventory.aInventoryHM.get(pCommand);
     if (pCommand.equals("magicCookie") &&
     this.aInventory.aInventoryHM.containsValue(vItem))
         this.aPoidsMax = this.aPoidsMax*2;
         this.aGui.println ("You eat the Magic Cookie, you
increases inventory'weight");
         this.aInventory.removeItem(pCommand, vItem);
     }
     else this.aGui.println ("You can't eat that");
}
```

Il faudra donc aussi changer la manière dont *eat()* est appelé dans *interpretCommand()* de *GameEngine* pour que eat prenne la String du deuxieme mot tapé.

Exercice 7,34,1 : Mise à jour des fichiers test en incorporant les nouvelles méthodes que peut effectuer le Player.

Exercice 7,34,2 : Voir exercice 7,26,1 pour générer les javadocs

Exercice 7,35 – Exercice 7,41,2 (optionnel) : x

Exercice 7,42 : On ajoute une limite de temps. On ajoute un attribut de nombre de pas limite à ne pas franchir (ici 30). Et dans la méthode *goRoom()* tout à la fin, verifier si le temps n'est pas à zéro, et sinon enlever 1 à ce timer :

```
if (aTimer == 0) {
     this.aGui.println("!! time is over !!");
     this.endGame();
}else {
     this.aTimer -= 1;
     this.aGui.println("You have " + this.aTimer + "moves even");
}
```

Exercice 7,42,1 (optionnel) : x
Exercice 7,42,2 : Nous ne ferons pas cette implantation graphique

Exercice 7,43 + Exercice 7,45 (optionnel) : Incorporation d'une trap door et d'une lock door grâce à une nouvelle classe *Door* qui comportera trois atributs boolean : *aTrap, aLock, aGoodDir*. Il faudra faire le constructeurs et les accesseurs get et set. Dans la méthode *createRoom()*, il faudra créé les trapDoor et lockDoor : attention pour une trapDoor il faut faire « deux doors », une de A vers B et une autre de B vers A, d'où l'intérêt de rajouter la bonne direction. Il faudra ensuite relier les salles existantes avec leur door grâce aux setters.
Attention il faudra aussi modifier *goRoom()* et *back()* de facon à ce que nous puissions pas aller dans un sens interdit.

Exercice 7,44 : Je n'ai pas réussi à faire cet exercice.

Exercice 7,45,1 : Fichier de test mis à jour.

Exercice 7,45,2 : Voir exercice 7,26,1 pour générer les javadocs

Exercice 7,46 – Exercice 7,47,2 : non traiter

Exercice 7,28 : Création d'une nouvelle classe *Character* qui a un nom, une Room et un Item. Dans la classe *Room*, faire une HashMap comme pour les doors et les sorties. Création des personnages et d'une fonction *talk()* qui permettra de parler aux personnages.

Remarques : J'ai décidé de sauter des exercices et de passer directement à la création des PNJ pour que mon jeu ai un sens, et que l'on puisses y sortir victorieux.

III. Mode d’emploi (instructions, comment démarrer le jeu, …) :

Pour démarrer le jeu, il faut créer un objet jeu en faisant un clic droit sur la classe *Game*, puis lancer la méthode *Play()*. Une fenêtre du jeu va alors s’ouvrir.
Les commandes de base pour jouer a ce projet Zuul sont :
- *go* + la direction (en anglais) pour se déplacer
- *help* pour obtenir les commandes valides
- *quit* pour quitter le jeu (ne sauvegarde pas la progression du jeu en cours)
- *look* pour avoir la description de la pièce (objet, personnages et sorties)
- *eat* pour manger
- *back* pour revenir dan sla salle recedente
- *test* pour executer un fichier texte
- *take* pour prendre un item dans une salle
- *drop* pour poser un item dans une salle
- *inventory* pour avoir la liste des items que porte le joueur
- *talk* pour parler aux personnages

IV. Sources :

template du site internet : https://html5up.net/

Lieux

chaudron d'or : http://mythologie13.m.y.pic.centerblog.net/o/b8badfff.jpg
clairiere + champignon : http://image.noelshack.com/fichiers/2013/23/1370780095-clairiere-arbre-magique.jpg
piece d'or : http://royaume-andoras.net/images/histoire/regions/foret_pelethor/foret.jpg
cascade d'eau : https://www.detoursenfrance.fr/sites/art-de-vivre/files/styles/large/public/det-cascade-planches-pres-arbois.jpg?itok=ppv3150d

Objets

objet magique : https://i.pinimg.com/474x/6c/43/ae/6c43ae32fa3c8eec78bba70f1b4b05d1--fantasy-landscape-fantasy-art.jpg

Persos

fee : http://immaginidivenezia.i.m.pic.centerblog.net/349b949a.jpg
lutin : https://i2.wp.com/peuple-feerique.com/wp-content/uploads/2017/04/Broceliande-tome01.jpg?resize=800%2C445
petit bonhomme bois :
http://img.over-blog-kiwi.com/1/01/19/89/20160219/ob_7fcb0c_1412189655-14450-1.jpg
elfe :
voleur (sorciere) : https://jooinn.com/images/forest-witch-1.jpg