Rapport version intermédiaire
Zuul

Ghezli Ramy B3/A3P

Aelyndor : l'épée des âmes

I.A) Auteur

Ghezli Ramy.

I.B) Thème

Dans les ruines d'un ancien royaume, un jeune héritier doit rassembler trois gemmes légendaires pour reforger l'Épée des Âmes.

I.C) Résumé du scénario

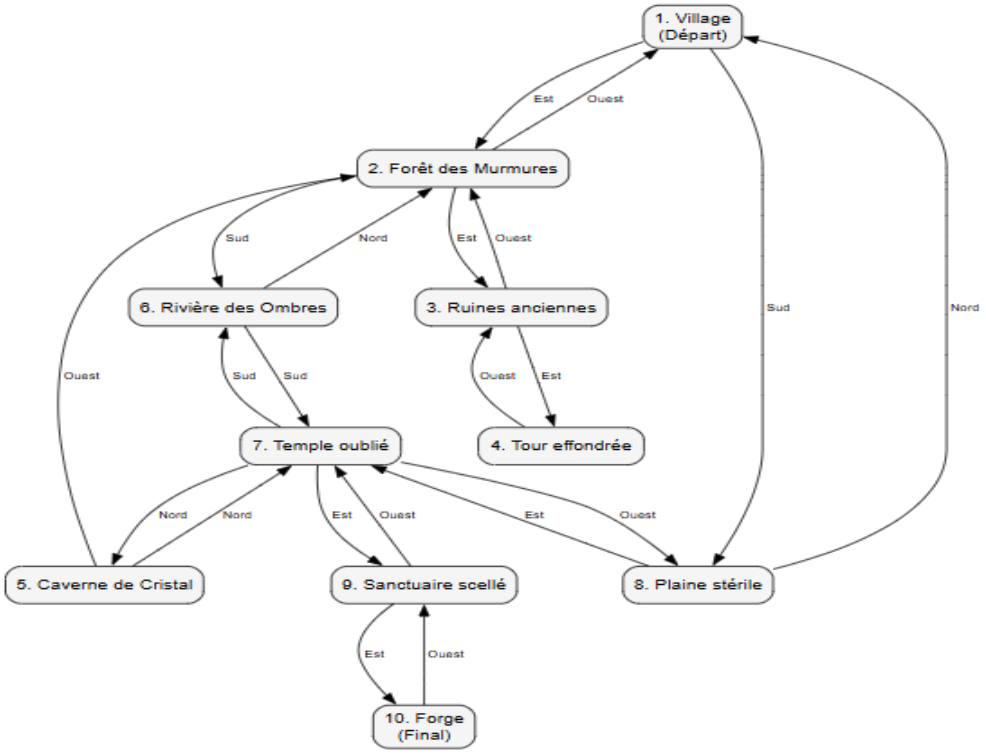
L'histoire se déroule dans le royaume déchu d'Aelyndor qui était autrefois prospère grâce à l'Épée des Âmes, une arme magique alimentée par trois gemmes légendaires, le royaume a sombré dans le chaos lorsque l'épée fut brisée et les gemmes dispersées.

Le joueur incarne un jeune villageois qui ignore qu'il est le descendant de la lignée royale sa quête commence modestement : il part chercher un moyen de sauver son village , il sera alors guidé par un vieil érudit, il explore les ruines du monde, découvrant peu à peu la vérité sur son passé et sur la gemme "morte" qu'il porte depuis toujours.

Son objectif est de retrouver les trois gemmes légendaires disséminées à travers les ruines, la tour, et les cavernes d'Ælyndor. Chaque gemme retrouvée lui révèle une partie de son héritage. La quête atteint son apogée lorsque le joueur, ayant rassemblé les trois gemmes atteint la Forge des Âmes. Là, il doit reforger l'Épée des Âmes, restaurant ainsi l'équilibre du royaume et s'affirmant comme l'héritier légitime.

I.D) Plan

plan complet:



Plan Réduit:

1. Village du héros
2. Forêt des Murmures
3. Ruines anciennes
4. Sanctuaire scellé
5. Forge des Âmes

I.E) Scénario détaillé

Mon scénario n'est pas encore très détaillé, le résumé dans la partie I.C) est une idée de ce vers quoi je veux aller mais des détails peuvent encore être ajoutés/enlevés, idem pour les lieux .

I.F) Détail des lieux, items, personnages

Lieux

1. **Village du héros** : Lieu initial du joueur , s'y trouve l'Érudit
2. **Forêt des Murmures** : Zone d'exploration. Contient un Chasseur
3. **Ruines anciennes** : Lieu de quête principal, contient le Garde spectral et la première gemme, sortie vers la Forêt et la Tour.
4. **Tour effondrée** :s'y trouve l'esprit du chevalier.
5. **Caverne de Cristal** : Contient la deuxième gemme.
6. **Rivière des Ombres** : Sens unique on y trouve un Marchand
7. **Temple oublié** : Point central de la carte on y trouve le Prêtre et la troisième et derniere gemme
8. **Plaine stérile** : Zone d'exploration secondaire , contient un individus Errant, et un objet.
9. **Sanctuaire scellé** : verrouillé, il faut les 3 gemmes pour y accéder
10. **Forge des Âmes** :Contient le socle de l'épée brisée c'est ici que se termine le jeu

Objets

1. **Gemme familiale** : Objet présent dans l'inventaire dès le départ de l'aventure, nécessaire pour la quête.
2. **Gemme légendaire 1** : À trouver dans les Ruines anciennes.
3. **Gemme légendaire 2** : À trouver dans la Caverne de Cristal.
4. **Gemme légendaire 3** : À trouver dans le Temple oublié.
5. **Herbes médicinales** : Objet consommable.
6. **Objet quelconque** : objet à Trouver dans la zone secondaire d'exploration (reste à définir)
7. **Petit coffre** : Objet à trouver à la Rivière

Personnages (PNJ)

1. **Érudit (Village)** : Donne la quête de départ au personnage et met en place le contexte de l'histoire.
2. **Chasseur (Forêt)** : Donne des informations sur l'histoire de l'Aelydor et l'origine du chaos qui règne.
3. **Garde spectral (Ruines)** : Protège la première gemme on s'y confrontera via des combat/enigmes.
4. **Esprit chevalier (Tour)** : Révèle l'histoire et le rôle de l'épée des âmes.
5. **Marchand (Rivière)** : Permet de tester les commandes take par exemple ou de future commande..
6. **Prêtre ancien (Temple)** : Révélation de l'héritage royale et de la responsabilité porté par le joueur.
7. **Errant (Plaine)** : PNJ optionnel donnant des informations d'ambiance.

I.G) Situations gagnantes et perdantes

Le joueur gagne si et seulement si 3 condition sont réuni:

1. Le joueur se trouve dans le lieu final
2. Il y a les 3 gemmes dans l 'inventaire du joueur .
3. Le joueur utilise l'action appropriée pour reforger l'Épée des âmes sur son socle.

I.H) Éventuellement énigmes, mini-jeux, combats, etc

Les énigmes, les combats, les mini-jeux restent pour le moment à définir , je n'ai pas réellement d'idée en tête, je vais pour ça effectuer des recherches et m'inspirer de ce qui existe dans certains jeux vidéo.

II. Réponses aux exercices (à partir de l'exercice 7.5 inclus)

7.5) le code qui affiche les informations du lieu est dupliqué dans printWelcome et goRoom , et au fur et à mesure des tp et de la lecture du livre j'ai pu comprendre que c'est une mauvaise pratique car cela complique la maintenance du code.

```
private void printLocationInfo(){
    System.out.println("Vous etes " +this.aCurrentRoom.getDescription());
    System.out.println("Sorties: ");
    if (this.aCurrentRoom.aNorthExit != null)
        {
            System.out.println("North");
        }
    if (this.aCurrentRoom.aSouthExit != null)
        {
            System.out.println("South");
        }
    if (this.aCurrentRoom.aEastExit != null)
        {
            System.out.println("East");
        }
    if (this.aCurrentRoom.aWestExit != null)
        {
            System.out.println("West");
        }
}
```

Pour corriger ça j'ai créé une nouvelle méthode privé printLocationInfo() dans la classe game , ce qui permet maintenant à printWelcome et goRoom de juste appeler cette méthode pour afficher la sorties du lieu.

7.6) les attributs de sortie de la classe Room comme par exemple aNorthExit étaient public ce qui viole le principe de l'encapsulation.

Pour corriger ça j'ai passé en privé les attributs sorties dans Room et défini un accesseur , getExit.

J'ai alors ensuite utilisé cet accesseur dans Game dans la méthode goRoom . ce qui permet d'enlever toute la

```
private void goRoom(final Command pSecondWord)
{
    //a)
    if (!pSecondWord.hasSecondWord())
    {System.out.println("Go where");
        return;
    }

    //b)
    String vDirection = pSecondWord.getSecondWord();
    Room vNextRoom = this.aCurrentRoom.getExit(vDirection);
    //c)

    if (vNextRoom == null)
    {
        System.out.println("there is no door !");
        return;

    }
    else
    {
        this.aCurrentRoom = vNextRoom;
        this.printLocationInfo();

    }

}
```

series de if pour la remplacer par: **this.aCurrentRoom.getExit(vDirection)** .

7.7) Pour finaliser l'encapsulation commencé à l'exercice 7.6 j'ai apporté la même modification a la méthode c .

```
private void printLocationInfo(){
    System.out.println("Vous etes " +this.aCurrentRoom.getDescription());
    System.out.println(this.aCurrentRoom.getExitsString());
}
```

Pour cela dans Room il fallait d'abord implémenter getExitsString .
ce qui permet de simplifier la méthode printLocationInfo comme montré ci dessus , en appelant que **this.aCurrentRoom.getString()**

```
public String getExitsString()
{
    String vExits = "Sorties: ";
    if( this.aNorthExit!=null){
        vExits= vExits + " North";
    }
    if( this.aSouthExit!=null){
        vExits= vExits + " South";
    }
    if( this.aEastExit!=null){
        vExits= vExits + " East";
    }
    if( this.aWestExit!=null){
        vExits= vExits + " West";
    }
    return vExits;

}
```

7.8)
Dans 7.6 et 7.7 j'ai amélioré l'encapsulation mais la classe Room utilisais 4 attributs privé ce qui limite à 4 directions , ainsi si je dois en ajouter une nouvelle , je devrais apporter des modification dans Game et ajouter plusieurs "if".

Pour remédier à ça les 4 attribut privé on été remplacé par une seul attribut :
private HashMap<String, Room> aExits;
et j'ai appliqué le reste des modifications montrées dans le chapitre 7.8.

Quand il fallait appliquer la modification pour getExitsString , j'ai du utiliser le lien "Comparaison Tableau/HashMap, pour voir comment afficher l'ensemble des clé , j'ai alors copié cette syntaxe:

```
Set<TypeDesClés> ensembleDesClés = nomHashMap.keySet();
for ( TypeDesValeurs variablePourUneValeur : ensembleDesClés )
System.out.println( variablePourUneValeur );
```

mais ça ne marchait pas car BlueJ n'as pas reconnu "Set" j'ai alors importé Set comme suggéré par le message d'erreur , le code était ensuite fonctionnel.

Maintenant on peut gérer n'importe quelle direction dans Room sans aucune modification supplémentaire.

7.9)
J'avais déjà apporté cette modification sur getExitsString comme demandé a la fin de la question 7.8.
toutefois j'ai apporté une modification à la classe game car il utilisait l'ancien système pour définir les sorties.
désormais pour définir les sorties d'un lieu je fais par exemple:
vVillage.setExit("south", vForet);
pour dire que le village a une sortie au sud et qu'elle mène vers la forêt.

7.10)
La Classe game contient moins de méthode que Room dans la Javadoc car la Javadoc ne prend en compte que les méthode public hors beaucoup de méthode de Gamme sont privée.

7.11)
J'ai modifié comme demandé dans le chapitre la classe Room en y ajoutant une nouvelle méthode getLongDescription() qui permet d'afficher la description du lieux ainsi que les sorties disponibles , ce qui se faisait avant depuis la classe Game grâce à la méthode printLocationInfo().

Grâce à ça je respecte bien le principe du "Responsibility-driven design" ainsi si plus tard je suis amené à ajouter des objet ou des personnages je n'aurais qu'à effectuer la modification dans getLongDescription().

7.14)
Pour cet exercice j'ai dû implémenter la commande "look" , pour cela j'ai d'abord dû modifier la classe CommandWord en ajoutant "look" au tableau aValidCommand et en modifiant la taille du tableau de 3 à 4.

```
public CommandWords()
{
    this.aValidCommands = new String[4];
    this.aValidCommands[0] = "go";
    this.aValidCommands[1] = "help";
    this.aValidCommands[2] = "quit";
    this.aValidCommands[3] = "look";
} // CommandWords()
```

Mais cette modification n'est pas suffisante car le jeu ne reconnait pas la commande , pour y remédier j'ai aussi modifier la classe Game en y ajoutant:

1. une méthode "look" qui affiche la description du lieux courant
2. un else if en plus dans la méthode processCommand() qui permet d'exécuter "look"
3. et j'ai mis a jours printHelp() pour aussi afficher "look"

7.15)
Je reproduit exactement le même processus pour la commande "eat" qui va afficher : "vous avez mangé , vous êtes désormais rassasié"

7.16)
l'exercice 7.12 a montré le problème du couplage implicite notamment lors de l'ajout de la commande "look" que j'ai dû ajouter manuellement.

Pour corriger ce problème j’ai donné la tâche d'afficher les commandes à une classe qui les connaît (ici Parser).