A3P 2016/2017 G8

Quentin GARRIDO
E1 Groupe 8
2016-2017

Sommaire :

I)Présentation globale du projet: 2
Auteur: 2
Phrase thème: 2
Scénario: 2
Plan du jeu: 4
Plan Simplifié : 5
Détail des lieux, objets, personnages 5
Situations Gagnantes et perdantes: 6

II) Réponses aux exercices 6
7.0: 6
7.1.1 6
7.2 6
7.2.1 6
7.3 7
7.3.1 7
7.3.2 7
7.4 7
7.5 7
7.6 7
7.7 8
7.7.1 8
7.8 8
7.8.1 9
7.9 9
7.10 9
7.10.1 10
7.10.2 10
7.11 10
7.14 10
7.15 11
7.16 12
7.17 12
7.18.1 13
7.18.3 13
7.18.4 13
7.18.5 13
7.18.6 14
7.18.8 14

I)Présentation globale du projet:

Auteur:

Quentin Garrido, étudiant en première année à l'ESIEE PARIS
Site internet : http://perso.esiee.fr/~garridoq

Phrase thème:

Dans une antre médiévale fantastique Luneth doit vaincre le Bahamut.

Scénario:

Luneth, chevalier Dragon de la lumière du village d'Ir, doit aller pourfendre Bahamut afin de récupérer son sang pour guérir le Roi du mal dont il souffre.

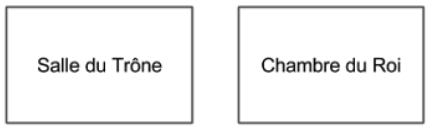
Bahamut se trouvant dans son antre en haut du Mont Amarth, Luneth devra trouver son antre et ensuite se frayer un chemin à l'intérieur afin de récupérer Ragnarok, la lance forgée un millénaire auparavant mais qui fut volée par le seigneur du mal Melkar et cachée avec le dragon lui même qui permettra à Luneth de vaincre le dragon.

Mais le chemin sera semé d'embûches et d'énigme que Luneth devra résoudre afin d'accéder au Bahamut. Heureusement pour lui, grâce à sa formation donnée par Firio, et l'armure de chevalier dragon dans le meilleur mithril du royaume, il est prêt à parer à toute éventualité afin de triompher de sa quête d'une importance vitale pour le peuple.

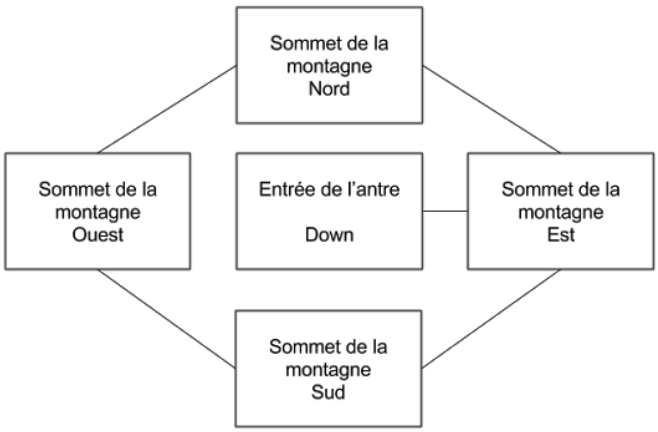
Grâce au savoir des anciens Sheik résidant dans la montagne avant que le Bahamut ne les chasse il y a 500 ans, il existerait 4 mécanismes divins permettant d'affaiblir toute créature dans la montagne, à condition de posséder un fragment d'elle.

Plan du jeu:

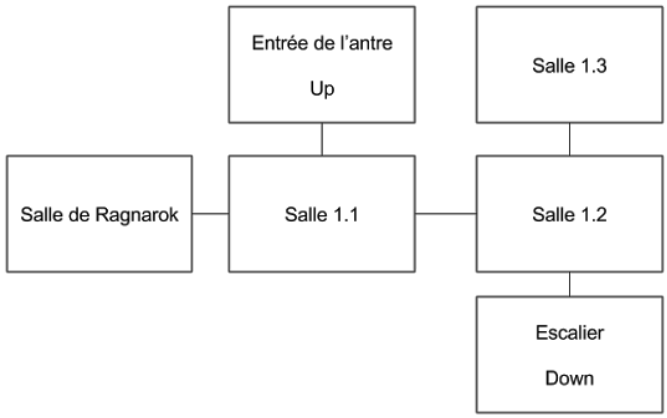
Chateau :



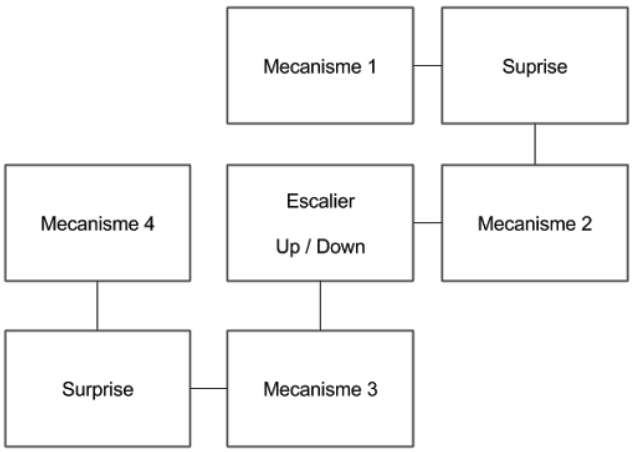
Montagne :



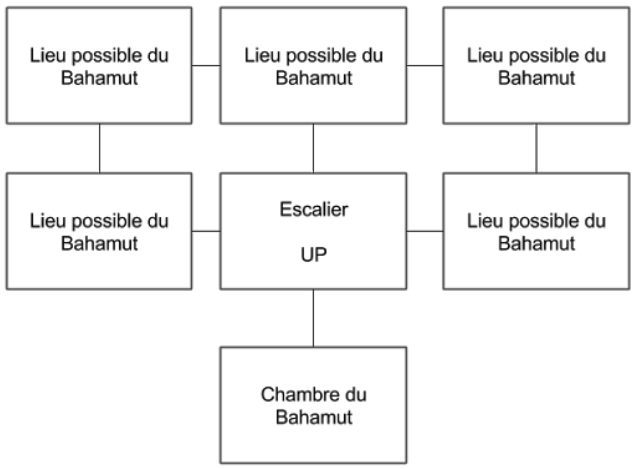
Etage 2:



Etage 1 :

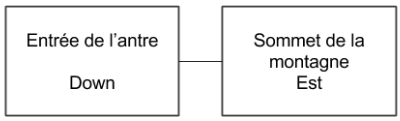


Etage 3:

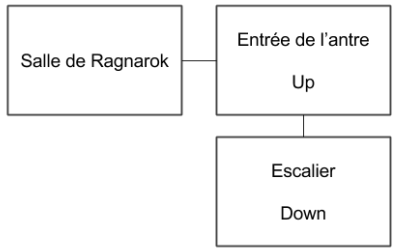


Plan Simplifié :

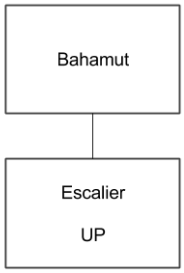
Montagne :



Etage 1:



Etage 2:



Détail des lieux, objets, personnages

Lieux: La salle du trône est le lieu de début du jeu
La chambre du Roi est le lieu de fin du jeu
Les salles Mecanisme contiennent les 4 mécanismes à activer
La salle libellée Ragnarok contient Ragnarok
Les 5 salles Lieu possible du Bahamut sont les salles ou le Bahamut pourra apparaître aléatoirement
Les Salles surprises contiendront des événements aléatoires
Le reste des salles possède des noms assez explicites, ou avec des événement non définis

Objets: Ragnarok, reste non définit

Personnages: Luneth (incarné par le joueur)

Situations Gagnantes et perdantes:

Le joueur gagne si il termine le donjon en donnant le sang du bahamut au Roi dans un temps imparti.
Il perd si il meurt dans un combat, ou lors de tout autre évènement, ainsi que si il est trop lent.

II) Réponses aux exercices

7.0:

Ma page internet a été créée à l'aide de Bootstrap et est accessible par tout le monde de part ses permissions et sa localisation dans mon dossier /public_html

Lien : http://perso.esiee.fr/~garridoq

7.1.1

Le thème choisit est le suivant : *" Afin de sauver le Roi d'un terrible mal, Luneth doit aller chercher du sang du Bahamut au mont Amarth"*

7.2

Les TPs 3.1 et 3.2 ont été terminés et vérifiés par les professeurs en cours.

7.2.1

Scanner va nous permettre de réucpérer l'entrée *System.in* (ici le clavier) et de la stocker dans une variable afin de traiter cette entrée, ce qui est indispensable pour notre programme qui demandera à l'utilisateur d'entrer des commandes.

7.3

Voir scénario écrit précédemment

7.3.1

Fait car le rapport est écrit avec les informations demandées

7.3.2

Voir plan plus haut

7.4

Les classes ont été mises à la racine du projet en se débarrassant du package v1.

7.5

Afin d'éviter toute duplication de code et pour réduire le coupling nous avons implémenté la méthode printLocationInfo séparément pour la réutiliser dans d'autres méthodes la requérant.

7.6

Nous avons modifié la classe Room comme dans le livre, afin de créer une fonction getExit prenant en paramètre la direction afin de savoir quelle pièce est dans la direction donnée, afin de pouvoir mettre les attributs de sortie en private et non public comme avant.
Nous avons donc aussi modifié la classe Game afin d'utiliser cette méthode (accesseur) car les attributs sont désormais private.

7.7

Création d'une méthode dans la classe Room:

```
public String getExitString()
{
    String vExitString = new String("The exits are :");
    if(this.aNorthExit != null)
        vExitString += " north";
    if(this.aSouthExit != null)
        vExitString += " south";
    if(this.aEastExit != null)
        vExitString += " east";
    if(this.aWestExit != null)
        vExitString += " west";

    return vExitString;

}
```

Pour afficher les sorties on remplace dans game le bloc de conditions de sorties par un appel de la méthode getExitString (eg : this.aCurrentRoom.getExitString();)

7.7.1

Fait, voir la partie 2 du rapport jusqu'à cet endroit.

7.8

Modification dan la classe Room:

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

public void setExits(final Room pN,final Room pS,final Room pE,final Room pW)
{
    if(pN != null)
      this.aExits.put("north", pN);
    if(pS != null)
        this.aExits.put("south", pS);
    if(pE != null)
       this.aExits.put("east", pE);
```

```
    if(pW != null)
       this.aExits.put(“west”, pW);
}

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

7.8.1

Voir plan du jeu pour se rendre compte des différents étages, induisant des déplacement verticaux.

7.9

Modification de la méthode getExitString de Room en :

```
public String getExitString()
{
    String vReturnString = "Les sorties sont:";
    Set<String> vKeys = this.aExits.keySet();
    for(String vExit : vKeys)
    {
        vReturnString += " " + vExit;
    }

    return vReturnString;

}
```

keySet() retourne un objet Set contenant toutes les keys (clefs) de l’objet HashMap

7.10

Tout d’abord nous créons une chaîne de caractères *vReturnString* avec le texte ne dépendant pas des sorties.
Ensuite nous récupérons les clefs de *aExits* grâce à *keySet.*

Nous allons ensuite parcourir les éléments du Set retourné, correspondant au sorties de la pièce courante, puis les ajouter une par une à *vReturnString*, enfin nous retournerons *vReturnString*.

7.10.1

Voir l'onglet javadoc sur le site

7.10.2

Voir l'onglet javadoc sur le site

7.11

Ajout de la méthode suivante dans la classe Room:

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

Et utilisation de cette dernière dans la classe Game pour afficher directement toute la description, sans la créer dans la classe Game, pour réduire le coupling.

7.14

Ajout de *"look"* dans les commandes valides de CommandWords.

Création de la méthode suivante dans Game :

```
private void look()
{
    this.printLocation();
}
```

Et ajout de la condition d'exécution de la commande dans processCommands (code montré avec l'exercice suivant car code similaire).

7.15

Ajout de *"eat"* dans les commandes valides de CommandWords.

Création de la méthode suivante dans Game :

```
private void eat()
{
    System.out.println("Vous avez mangé et n'avez plus faim désormais.");
}
```

Et modification de processCommands (ici avec look et eat ajoutés) :

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

        if(pC.getCommandWord() == null)
         {
        System.out.println("I don't understand");
        return false;

        }
        else if(pC.getCommandWord().equals("go"))
        {
        this.goRoom(pC);
        return false;
        }
        else if(pC.getCommandWord().equals("help"))
        {
        this.printHelp();
        return false;
        }
        else if(pC.getCommandWord().equals("look"))
        {
        this.look();
        return false;
        }
        else if(pC.getCommandWord().equals("eat"))
        {
        this.eat();
```

```
        return false;
        }
        else if(pC.getCommandWord().equals("quit"))
        return this.quit(pC);
        else
        return false;
}
```

7.16

Ajout de la méthode suivante dans la classe CommandWords pour afficher toutes les commandes lorsque l'on utilise la commande help :

```
        public void showAll()
        {
        for(String vCommand : sValidCommands)
        System.out.print( vCommand +" ");

         System.out.println();

        }
```

Ajout dans la classe Parser d'une procédure appelant showAll() afin d'éviter de créer du coupling entre Game et CommandWords, ainsi on n'utilise que des liens déjà existants.

7.17

Dans cet exercice, nous avons modifié la methode showAll() de la classe CommandWords en :

```
        public String getCommandList()
        {
        String vString = new String("");
        for(String vCommand : sValidCommands)
        vString += vCommand + " ";
        return vString;

        }
```

Nous avons juste dû changer après dans la classe PArser la manière d'afficher les commandes, en n'appelant plus directement showAll (renommée en getCommandList) mais en l'affichant.

7.18.1

Rien n’a eu à être modifié.

7.18.3

Les images sont toutes intégrées dans la version simplifiée du jeu et sont présentes sur la version intermédiaire.

7.18.4

Titre du jeu décidé : Bahamut’s Lair.

7.18.5

Création d’une array list :

Dans le constructeur:

```
this.aRoomList = new ArrayList<Room>(7);
```

Dans createRooms():

```
        aRoomList.add(vSommetEst);
         aRoomList.add(vEntreeH);
         aRoomList.add(vEntreeB);
         aRoomList.add(vRagnarok);
         aRoomList.add(vEscalierH);
         aRoomList.add(vEscalierB);
         aRoomList.add(vBahamut);


         this.aCurrentRoom = aRoomList.get(0);
```

7.18.6

Tout a été mis en place en respectant la structure de zuul-with-images et aucune erreur n'est présente lors de la compilations.
Les principaux changements ont été l'emplacement de méthodes dans certaines classes

7.18.8

Créations de 10 boutons pour gérer les actions et déplacements:

```
this.aButtonHarakiri = new JButton("harakiri");
this.aButtonEat = new JButton("eat");
this.aButtonLook = new JButton("look");
this.aButtonAttack = new JButton("attack");

this.aButtonNorth = new JButton("go nord");
this.aButtonNorth.setPreferredSize( new Dimension(100,66));
this.aButtonSouth = new JButton("go sud");
this.aButtonEast = new JButton("go est");
this.aButtonWest = new JButton("go ouest");
this.aButtonUp = new JButton("go haut");
this.aButtonDown = new JButton("go bas");
```

Et d'une ArrayList les contenant

```
this.aButtonList = new ArrayList<JButton>(10);

this.aButtonList.add(this.aButtonHarakiri);
this.aButtonList.add(this.aButtonEat);
this.aButtonList.add(this.aButtonLook);
this.aButtonList.add(this.aButtonAttack);
this.aButtonList.add(this.aButtonNorth);
this.aButtonList.add(this.aButtonSouth);
this.aButtonList.add(this.aButtonEast);
this.aButtonList.add(this.aButtonWest);
this.aButtonList.add(this.aButtonUp);
this.aButtonList.add(this.aButtonDown);
```

Ainsi création de listener simplifiée par cette ArrayList, permettant d'avoir uniquement :

```
for(JButton vB : this.aButtonList)
    vB.addActionListener( this);
```

Et dans l'interprétation des commandes:

```
if(pE.getSource().getClass() == this.aButtonHarakiri.getClass())
        this.aEngine.interpretCommand(pE.getActionCommand());
else
        this.processCommand();
```

Ensuite nous avons empêché l’utilisation de boutons quand le joueur est mort , encore une fois simplifié par l’arrayList créée auparavant :

```
for(JButton vB : this.aButtonList)
        vB.setEnabled(pOnOff);
```