Rapport A3P (Java A3P 2025/2026 G2)

Le retour de Yamata no Orochi

ヤマタノオロチの帰還

I. Informations sur le jeu

a) Auteur du jeu

Christophe TARATIBU

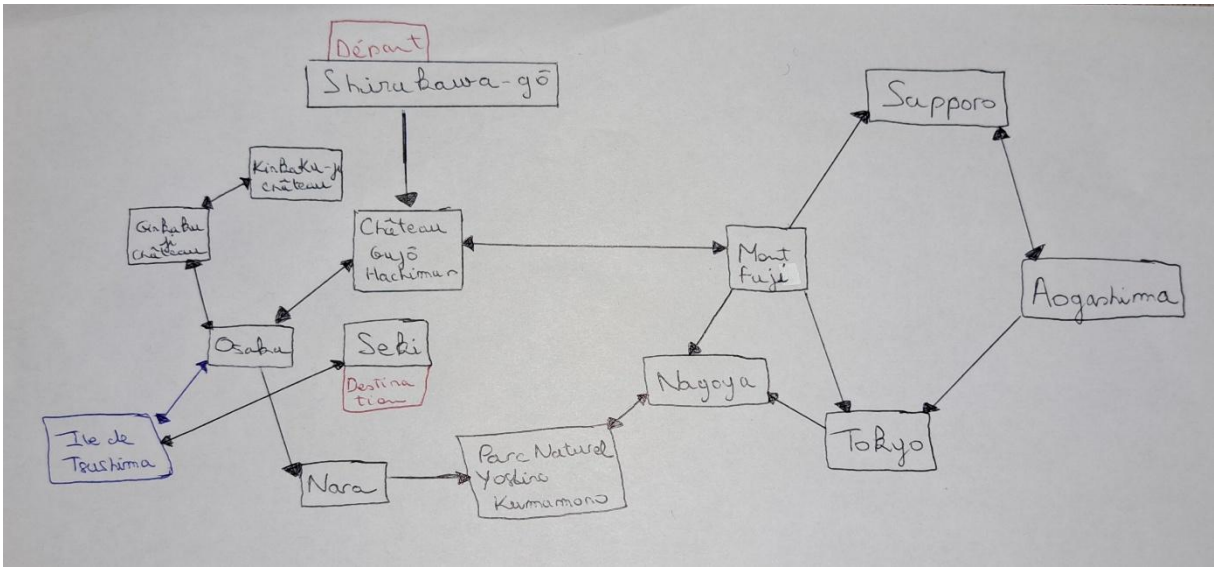
b) Thème (Phrase thème validée)

Dans le Japon contemporain, le forgeron doit forger un sabre légendaire qui pourfend le mal.

c) Résumé du scénario (complet)

Le Japon fait face à une crise sans précédent ! Yamata no Orochi le dragon maléfique à 8 têtes à fait son grand retour, ressuscité par des monstres maléfiques appelés Yôkais. Tetsuma Hasegawa, forgeron humain choisi par la déesse du soleil Amaterasu, a pour mission de parcourir le Japon afin de trouver les 9 matériaux qui permettra à Susanoo dieu des tempêtes et frère de la déesse de terrasser le monstre avec le sabre que Tetsuma aura forgé à Seki, le village des forgerons. Néanmoins quelques monstres puissants seront là pour compliquer sa tâche.

d) Plan du jeu (Villes réelles du Japon)



e) Scénario complet du jeu

Quand le jeu débute, le joueur se retrouve dans la ville de Sirakawa-gô ville ou le héros est originaire. Il entend alors la voix céleste d'Amaterasu lui disant que le monstre Yamata no Orochi est revenu et qu'il a pour mission de trouver 9 matériaux pour forger la lame de Totsuka, sabre qui permettra de tuer la créature. Ces matériaux sont dispersés à travers tout le Japon et le héros doit tous les trouver et créer la lame dans la ville de Seki, villages des forgerons où de nombreux sabres ont été forgées. Pour bien mener sa mission, le personnage doit passer par le château Gujô Hachiman qui a été

transformé en lieu infini par un yokaï mystérieux, la fille à la guitare qui posera une question et suivant la réponse du héros, il sera téléporté à un endroit contenant un matériau ou non. A Osaka, pourra visiter la ville et manger. Sur l'île de Tsushima, il fera la rencontre du dieu des mers Ryûjin qui lui racontera un peu de mythologie du pays et lui donnera comme matériau ses écailles aquatiques puis le héros devra aller dans les châteaux de Kinkaku-ji et Ginkaku-ji pour trouver respectivement les poudres d'or et d'argent qui ont été subtilisés par deux yokaïs. Il devra trouver avoir la bénédiction du temple de Tokyo sinon il se fera tuer instantanément (c'est un humain après tout). Il pourra aller Nara ou il rencontra la déesse cerf (Shika no Megami) qui après avoir combattu les monstres qui attaquaient son sanctuaire fait don au héros de d'un de ses bois, nécessaire à la création de la lame. A l'est, le forgeron, se trouvera dans le parc national Yoshino ou il rencontrera les esprits de la forêt qui lui permettront de couper les bambous à condition de leur faire une offrande de nourriture (La demande sera aléatoire). A Nagoya, le héros trouvera le Habaki (un collier en cuivre) nécessaire pour l'objet tranchant. Au mont Fuji le joueur sera accueilli par les esprits de la montagne et la récupération des pierres pourra se faire en trouvant la taille de l'ancien volcan dans le temps imparti. Si la hauteur n'est pas trouvée, les esprits banniront le joueur et il aura automatiquement perdu. S'il réussit il pourra récupérer du bois de magnolia à Sapporo et se reposer un peu (il y un sanctuaire de la déesse ici). Il ira à Aogashima récupérer les cendres de l'île mais il devra combattre le grand kappa qui lui posera des devinettes dans un temps imparti avec pour risque de noyer l'île tout entière. Ensuite, il pourra aller à Tokyo recevoir la bénédiction et une amulette qui permettra d'aller dans les châteaux à côté d'Osaka. Il aura fini sa quête et devra trouver son chemin pour aller à Seki forger enfin son sabre.

f) Détail des lieux, items et personnages

Les salles de ce jeu représentent les villes et lieux du Japon. Une description générale des lieux est faite dans le code du jeu. Les objets du jeu ont aussi leur propre description.
Voici une description des lieux, personnages et objets du jeu :

- Shirakawa-go : C'est le village de départ de notre héros. Il entendra alors le message d'Amaterasu qui précisera sa mission. Il aura avec lui sa pioche et hache qui lui permettra de récupérer des matériaux nécessaires

- Château Gûjo Hachiman : C’est un château qui a été rendu magique par la fille à la guitare qui expliquera le pouvoir du lieu. Cela désorientera le héros. Néanmoins c’est le lieu qui dirigera la quête de ce dernier.

- Osaka : La grande ville du sud-ouest du Japon.

- Île de Tsushima : Île situé entre la Corée du Sud et le Japon, lieu de villégiature du dieu de l’eau et des océans Ryûjin, le joueur pourra récupérer des écailles aquatiques.

- Kinkaku-ji : Château pris en possession par la démone blanche qui contient la poudre dorée. Le joueur doit être béni par le temple de Tokyo pour pouvoir récupérer ce matériau.

- Ginkaku-ji : Château pris en possession par le démon noir qui contient la poudre argentée. Le joueur doit être béni par le temple de Tokyo pour pouvoir récupérer ce matériau.

- Nara : Ville du Japon connu pour son parc de cervidés, le joueur rencontrera la déesse cerf blessée qui lui racontera l’attaque de son sanctuaire et lui donnera un de ses bois.

- Parc national Yoshino : Parc connu pour ses forêts des cerisiers (Sakuras). Le joueur trouvera les esprits du parc et pourra récolter des bambous.

- Nagoya : Une des grandes villes du Japon où le joueur pourra trouver le habaki, collier de cuivre nécessaire à la création de l’arme.

- Mt Fuji : Mont montagneux qui fait la célébrité du Japon, le héros rencontrera les esprits de la montagne et pourra récolter la roche qui en est issu.

- Sapporo : Grande ville du Japon, le joueur pourra récolter du magnolia, matériau important pour la confection de l’arme.

- Île d’Aogashima : Îlot de terre perdu au beau milieu de l’océan Pacifique. Le joueur devra récolter des cendres de ce lieu mais devra battre avant le grand kappa, un yokaï des eaux qui mettra le héros à l’épreuve.

- Tokyo : Ville qui fait la renommée de ce pays et c’est la capitale. Le joueur rencontrera la déesse Amaterasu dans son sanctuaire qui le bénira et lui offrira une amulette, symbole de sa bénédiction.

- Seki : Village des forgerons ou le joueur pourra enfin créer l'arme avec les matériaux trouvés. Le joueur aura gagné et le jeu pourra s'achever.

g) Situations gagnantes et perdantes :

Pour les situations gagnantes, le joueur gagnera dès qu'il aura trouvé tous les matériaux nécessaires à la forge de la lame, il perdra à coup sur s'il se rend dans les châteaux possédés par les démons sans protection ou s'il ne réussit pas les diverses épreuves qu'il rencontrera dans sa quête.

h) Eventuellement énigmes, mini-jeux, combats, etc.:

Quelques énigmes et conditions vont êtres implémentés dans ce jeu pour ajouter un peu de réflexion et de difficulté.

i) Commentaires (ce qui manque, reste à faire, ...) :

Le jeu à une bonne configuration des salles mais manque encore de partie graphique qui affiche les différents lieux explorés par le joueur.

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

1. Exercice 7.5

En ce qui concerne cet exercice, il y a la présence dans la classe Game, de lignes de codes dupliquées qui sont présentes dans les procédures `goRooms(Command pCmd)` et `void welcome()` dont l'une permet de changer de pièce et l'autre qui affiche la fenêtre de bienvenue.
Voici les lignes en question :

```
System.out.println(" You are " +
this.aCurrentRoom.getDescription());

System.out.println("Possible exits");
if (this.aCurrentRoom.aEastExit != null) {
```

```
            System.out.println("- East");
      }
      if (this.aCurrentRoom.aWestExit != null) {
            System.out.println("- West");
      }
      if(this.aCurrentRoom.aNorthExit != null){
            System.out.println("- North");
      }
      if (this.aCurrentRoom.aNorthExit != null) {
            System.out.println("- North");
      }
      if (this.aCurrentRoom.aSouthExit != null) {
            System.out.println("- South");
      }
```

Ces lignes ci-dessus permettent affichage de la salle courante et L'affichage des sorties possibles de cette dernière matérialisée par les points cardinaux (Nord, Sud, Est et Ouest en anglais). Ainsi pour éviter la duplication de code, la création d'une procédure est nécessaire. C'est là qu'intervient la procédure privée `printLocationInfo()` qui effectue ce travail.

```
      private void printLocationInfo(){

            System.out.println(" You are " +
            this.aCurrentRoom.getDescription());
            System.out.println("Possible exits");
            if (this.aCurrentRoom.aEastExit != null) {
                  System.out.println("- East");
            }
            if (this.aCurrentRoom.aWestExit != null) {
                  System.out.println("- West");
            }
            if(this.aCurrentRoom.aNorthExit != null){
                  System.out.println("- North");
            }
            if (this.aCurrentRoom.aNorthExit != null) {
                  System.out.println("- North");
            }
            if (this.aCurrentRoom.aSouthExit != null) {
                  System.out.println("- South");
            }

}
```

Les procédures qui contenaient du code dupliqué contiennent dorénavant la ligne de code `this.printLocationInfo();` qui permet l'exécution de la nouvelle procédure.

2. Exercice 7.6

Le code du jeu est fonctionnel mais les attributs de la classe Room sont publics et cela ne respecte pas les conventions du langage Java.

```
public Room aNorthExit;
public Room aEastExit;
public Room aSouthExit;
public Room aWestExit;
```

Attributs publics dans la classe Room

Ainsi la classe Game avait accès aux attributs de la classe Room.

Cependant les sorties étaient énumérées et ceci devenait vite répétitif en cas de multiplication de nouvelles sorties. Afin de compenser cela, on utilise l'encapsulation qui octroie une meilleure protection des attributs en en remplaçant le mot clé java `public` par un autre mot clé `private` . Ceci évite une modification anarchique des données et/ou d'invariants d'une classe qui compromettrait tout le code final. Une méthode publique de la classe Room `Room getExit(String vDirection)` qui fait office de getter permet de récupérer les sorties possibles d'une salle en indiquant un point cardinal.

```
public Room getExit(String vDirection){
	switch (vDirection) {
		case "north":
			return this.aNorthExit;
		case "south":
			return this.aSouthExit;
		case "east":
			return this.aEastExit;
		case "west":
			return this.aWestExit;
		default:
			return null;
	}
}
```

Par conséquent la procédure privée `private void goRooms(Command pCmd)` se retrouve aussi modifiée car les attributs de la classe Rooms sont maintenant privés.

```
private void goRooms(Command pCmd){
	if(!pCmd.hasSecondWord()){
		System.out.println("Go where ?");
	}
	Room vNextRoom = null;
	String vDirection = pCmd.getSecondWord();
	System.out.println(vDirection);
	switch (vDirection) {
		case "north":
		vNextRoom = this.aCurrentRoom.getExit(vDirection);
		break;

		case "south":
		vNextRoom = this.aCurrentRoom.getExit(vDirection);
		break;

		case "east":
		vNextRoom = this.aCurrentRoom.getExit(vDirection);
		break;
		case "west":
		vNextRoom = this.aCurrentRoom.getExit(vDirection);
		break;
		default:
		System.out.println("Unknown direction");
		break;
	}
	if(vNextRoom == null){
	System.out.println("There's no door");
	} else {
		this.aCurrentRoom = vNextRoom;
		this.printLocationInfo();
	}
}
```

3. Exercice 7.7

La classe Room a pour mission la création des salles de pour le jeu de zuul. On utilisera la méthode publique `public String getExitString()` qui servira de stockage pour les sorties possibles d'une salle. Donc la classe principale du jeu, Game pourra récupérer la « liste » de salles qui est contenue.

```
public String getExitString(){
```

```
            String vStrExits = "";
            if(this.getExit("south") != null){
                  vStrExits += "South ";
            }
            if(this.getExit("north") != null){
                  vStrExits += "North";
            }
            if(this.getExit("east") != null){
                  vStrExits += "East";
            }
            if(this.getExit("west")!= null){
                  vStrExits += "West";
            }
            return vStrExits;
      }
```

4. Exercice 7.8

La nouvelle classe HashMap va nous permettre de stocker les salles avec leurs sorties respectives. On pourra même ajouter des directions verticales comme bas et haut. L'affectation des sorties avec les salles attribuées sera plus simple avec la méthode `put` propre à la classe choisie. Cela va faciliter l'affectation des sorties possibles des salles sans passer par l'ancienne méthode qui consistait à attribuer les sorties d'une salle par 4 suivant les points cardinaux.

`import java.util.HashMap;` -> permet d'invoquer la classe avec les méthodes qui vont avec
`private HashMap<String, Room> aExits;` -> Sorties possibles d'une pièce représentés par une table de hachage.
Voici une comparaison entre l'ancien code et le nouveau code la procédure `public void setExits`:

```
// Ancien code
   public void setExits(final Room pNorthExit,final Room pEastExit,
final Room pSouthExit, final Room pWestExit){
      if(pNorthExit != null){
      this.aExits.put("north", pNorthExit);
  }
  if(pEastExit != null){
      this.aExits.put("east", pEastExit);
  }
  if(pSouthExit != null){
      this.aExits.put("south", pSouthExit);
```

```
    }
    if(pWestExit != null){
        this.aExits.put("west", pWestExit);
    }
}

// Nouveau code

public void setExit(String pDirection, Room pNeighbor){

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

On se retrouve ainsi avec du code plus simple et concis. Pour la définition des sorties d'une salle :

```
vOsaka.setExits(vGinkaku,vGujoHachi,vNara,vTsushi); // Salle qui
représente la ville d'Osaka

vOsaka.setExit("north", vGinkaku);
vOsaka.setExit("east", vGujoHachi);
vOsaka.setExit("south", vNara);
vOsaka.setExit("west", vTsushi);

// Nouvelle représentation des sorties de Salles
```

5. Exercice 7.8.1

Question au prof (Peut être déplacement vertical Mt Fuji)

6. Exercice 7.9 / 7.10

La classe Set du langage java permet de stocker les éléments choisis dans un « ensemble ». Ils sont uniques dans cette structure de donnée et aucun doublon n'est autorisé. La structure de donnée keySet permet de stocker les clés de la Hashmap (en l'occurrence aExits. On utilisera la méthode keySet() qui permettra de récupérer les clés de la Hashmap. Une boucle For Each sera utilisée pour itérer sur le Set (qui est une liste de String unique de sorties possible d'une salle) et concaténer avec la variable `vExitString` qui contiendra les sorties possibles d'une salle.

Voici le code qui répond à l'exercice demandé :

```
public String getExitString(){
        String vExitString = "Exits:";
        for(String vDirection : this.aExits.keySet()){
```

```
            vExitString += " " + vDirection;
        }
        return vExitString;
    }
```

7. Exercice 7.10.1 / 7.10.2

Javadoc faite.

8. Exercice 7.11

Dans cet exercice, le couplage a été réduit afin de faciliter l'ajout de nouvelles descriptions pour la salle. Avec l'ancienne procédure, la description complète des salles et des sorties était gérée par deux autres procédures de la classe Room. Une unique procédure affiche dorénavant la description d'une salle en concaténant la description principale et les sorties possibles :

```
public String getLongDescription(){
        return "You're " + this.aDescription + getExitString();
}
 //Dans la classe Room

 private void printLocationInfo(){

        System.out.println(this.aCurrentRoom.getLongDescription());
 }
 //Dans la classe Game
```

9. Exercice 7.14/7.15

Pour limiter le découplage, il est essentiel de localiser les modifications : une classe doit pouvoir être modifiée sans impacter les autres. Les champs publics peuvent introduire divers niveaux de couplage, dont le plus problématique est le couplage implicite, lorsque qu'une classe dépend indirectement des données internes d'une autre. Ce risque apparaît, par exemple, lors de l'ajout d'une nouvelle commande au jeu, en plus de quit, help et go. La commande look, destinée à réafficher la description de la pièce actuelle et ses sorties, doit être implémentée de manière à éviter

tout couplage implicite. Dans la classe CommandWords, la commande look a été introduite dans le tableau statique des commandes principales et une procédure look, à été codée et réaffiche la description de la salle où se situe le joueur au cas où il serait perdu. La nouvelle commande eat est aussi créée en tant qu'utilitaire et le code est implémenté de la même manière. Voici le code modifié :

Dans la classe CommandWords :

```
public CommandWords()

{
this.aValidCommands = new String[5];
this.aValidCommands[0] = "go";
this.aValidCommands[1] = "help";
this.aValidCommands[2] = "quit";
this.aValidCommands[3] = "look";
this.aValidCommands[4] = "eat";



}
```

Et dans la classe Game :

```
private void look(){

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

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


private boolean processCommand(final Command pCmd) {

      if (pCmd.isUnknown()) {
            System.out.println("I don't know what you mean...");
            return false;
      }
      String vCommandWord = pCmd.getCommandWord();
      if ("help".equals(vCommandWord)) {
            printHelp();
            return false;
      } else if ("go".equals(vCommandWord)) {
            goRooms(pCmd);
            return false;
            } else if ("quit".equals(vCommandWord)) {
```

```
                return quit(pCmd);

            } else if("look".equals(vCommandWord)){
                this.look();
                return false;

            } else if ("eat".equals(vCommandWord)){
                this.eat();
                return false;
            } else {
                    System.out.println("Erreur du programmeur
: commande non reconnue !");
                return true;
            }
        }
```

10. Exercice 7.16

Après l'ajout des commandes eat et look, la méthode printHelp() se retrouve alors incomplète. Comme solution la procédure showAll() dans la classe CommandWords() va faire une boucle for-each sur la liste des commandes et énumérer les commandes possibles en les affichant en sortie.

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

III. Déclaration anti-plagiat

Le scénario crée est s'inspire beaucoup de la mythologie japonaise et du manga Kimetsu no Yaiba (Les Rôdeurs de la nuit). L'idée du forgeron se base réellement sur de vrais qui ont crée des armes qui ont marqué l'histoire du pays. Les exercices faits ont pour base le Zuul-Bad et les exercices du chapitre 7 du livre de Java.