Rapport de projet de HIGELIN Maxime :

A La recherche de « LostIsland »

(A3P(AL) 2023/2024 G3)

I.A) Auteurs :

HIGELIN Maxime

I.B) Thème (phrase-thème validée) :

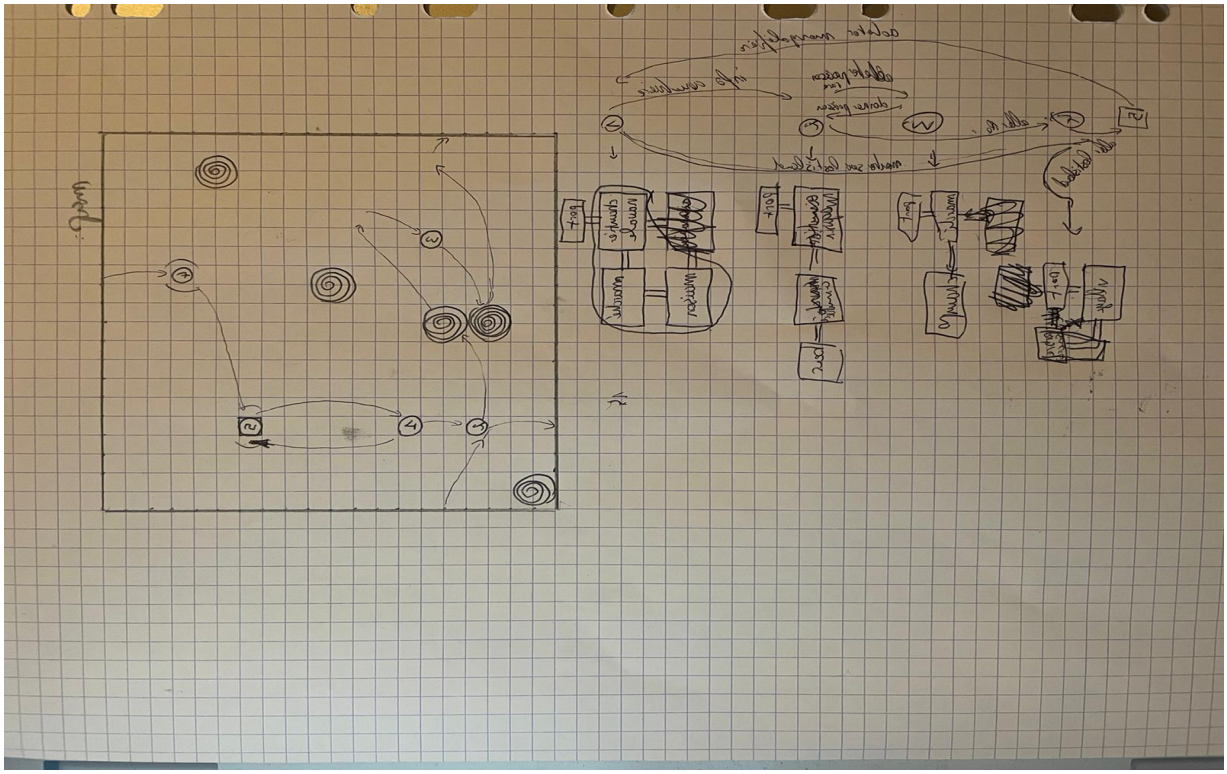
Un jeune marin doit retrouver « LostIsland ».

I.C) Résumée du scénario :

Le jeu se déroule dans un monde constituer de 5 iles au milieu de l'océan. On Débute le jeu sur l'ile de départ ou l'on apprend que la légende sur l'ile perdue (LostIsland) n'est en fait pas une légende. Cette ile qui regorge de ressource miraculeuse et rare est bien réel. Le jeune garçon que nous incarnons décide donc de prendre le large pour retrouver cette ile. Il va devoir accomplir plusieurs taches et quêtes afin de récupérer des information qui lui permettrons de retrouver l'ile perdu.

On le comprend bien, l'objectif est donc de retrouver cette ile.

I.D)Plan :



I.F)

1. Lieux :

Il existe 5 iles , celles-ci sont toutes accessible en bateau, lorsqu'on arrive sur une ile, on se retrouve au « port » de celle-ci. Chaque il a donc un port.

Ile 1 : c'est l'ile de départ, on y retrouve la maison du jeune héros, un marché et un chantier naval.

Ile 2 : Cette ile est essentiellement un immense cimetière . on y retrouve aussi un bar et un magasin.

Ile 3 : Cette ile se compose d'un marché et d'un champs.

Ile 4 : cette ile abrite un village et un endroit avec un immense arbre

Ile 5 : C'est « LostIsland », L'ile que l'on cherche. Elle se situe dans le ciel est n'est accessible que en montgolfière.

2.PNJ :

Il y a une fille, qu'on rencontre au cimetière de l'ile 2. Un barman qu'on retrouve au bar de l'ile 2. Il y a la chef du village sur l'ile 4. Sur chaque marché et magasins on retrouves des PNJ vendant leurs objets.

3. ITEMS :

Des cartes, qui donne notre positions.

Des Boussoles, qui pointe vers des iles.

Une Longue-vue, qui permet d'examiner les lieux alentours.

De la nourriture, qui peut être échanger ou utilisée.

Des pièces, qui peuvent être échangée.

Des armes, pour se défendre des attaques pirates.

I.G) Situations gagnantes et perdantes

1) Gagnant.

Si le Joueur retrouve L'ile perdue.

2) Perdant.

Si le joueur meurt de faim.

Si il se fait attaquer dans la mer et qu'il n'a pas d'armes.

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

Enigme permettant d'aider le joueur a savoir que l'ile perdue se trouve dans le ciel :

Sur un boussole il est gravé : « il faut savoir lever les yeux pour atteindre nos rêves »

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

Au niveau du code, il reste tout a faire. Aucune implémentation propre au jeu n'a encore été faite.

II. Exercice 7.5 a 7.16 :

7.5

Je créer une methode « printLocationInfo » :

```
private static void printLocationInfo(Room pRoom)
    {
        System.out.println(pRoom.getDescription());
        System.out.print("Exits: ");
        System.out.println(pRoom.getExitString());
    }
```

On créer une procédure pour éviter la duplication de code. En effet, si un jour on rajoute des sorties, nous aurons maintenant besoins de changer seulement le code dans la procédure et non chaque méthodes qui affiche les infos du lieu.

7.6

1. Oui, il n'est pas utile de vérifier que chaque paramètre est « null » car dans le cas où un paramètre serait « null », on affecterait pas la valeur « null » a la sortie. Or la sortie par default est « null ». On a Donc pas besoins de vérifier que le paramètre qu'on veut affecter à la sortie n'est pas « null ».

2. Modification dans la classe « Game » :

```
53  -
54  -       switch(vDirection) {
55  -           case "North":
56  -               vNextRoom = this.aCurrentRoom.aNorthExit;
57  -               break;
58  -           case "South":
59  -               vNextRoom = this.aCurrentRoom.aSouthExit;
60  -               break;
61  -           case "East":
62  -               vNextRoom = this.aCurrentRoom.aEastExit;
63  -               break;
64  -           case "West":
65  -               vNextRoom = this.aCurrentRoom.aWestExit;
66  -               break;
67  -           default:
68  -               System.out.println("Unknown direction !");
69  -               return;
    55  +
    56  +       if (!commandWords.isDirection(vDirection))
    57  +       {
    58  +           System.out.println("Unknown direction !");
    59  +           return;
    60  +       }
    61  +       else
    62  +       {
    63  +           vNextRoom = aCurrentRoom.getExit(vDirection);
70  64          }
```

Modification dans la classe « Room », ajouts d'une méthode « getExit » :

```
    28  +
    29  +   public Room getExit(final String pDirection)
    30  +   {
    31  +       if(pDirection.equals("North"))
    32  +       {
    33  +           return aNorthExit;
    34  +       }
    35  +       if(pDirection.equals("South"))
    36  +       {
    37  +           return aSouthExit;
    38  +       }
    39  +       if(pDirection.equals("East"))
    40  +       {
    41  +           return aEastExit;
    42  +       }
    43  +       if(pDirection.equals("West"))
    44  +       {
    45  +           return aWestExit;
    46  +       }
    47  +       return null;
    48  +   }
```

3. Pour résoudre les problèmes liés à « Unknown direction » :

- J'ai ajouté un fonction booléen « isDirection » dans la classe « CommandWords » qui retourne vrai si le paramètre est une direction valide et faux si ça ne l'est pas :

```
 private final String[] aValidDirections =
{"North","South","East","West","Up","Down"};


public boolean isDirection(final String pDirection)
    {
        for(int i=0;i<aValidDirections.length;i++)
        {
            if(aValidDirections[i].equals(pDirection))
            {
                return true;
            }
        }

        return false;
    }
```

- J'ai ajouté une condition dans la méthode « goRoom » de la classe « Game », qui affiche « Unknown direction » si la méthode « isDirection » retourne faux :
-

```
55  +
56  +       if (!commandWords.isDirection(vDirection))
57  +       {
58  +           System.out.println("Unknown direction !");
59  +           return;
60  +       }
61  +       else
62  +       {
63  +           vNextRoom = aCurrentRoom.getExit(vDirection);
70  64          }
```

-

7.7

1./2. Définition de la methode « getExitString » dans la classe « Room » :

```
private HashMap<String, Room> exits;
    private String[] aExitDirections = {};

public String getExitString()
    {
        ArrayList<String> directions = new ArrayList<String>();
        for (String direction: exits.keySet())
        {
            directions.add(direction);
        }
        return String.join(" ",  directions.toArray(this.aExitDirections));
    }
```

Utilisation de la methode « getExitString » dans la methode « printLocationInfo » de la classe « Game » :

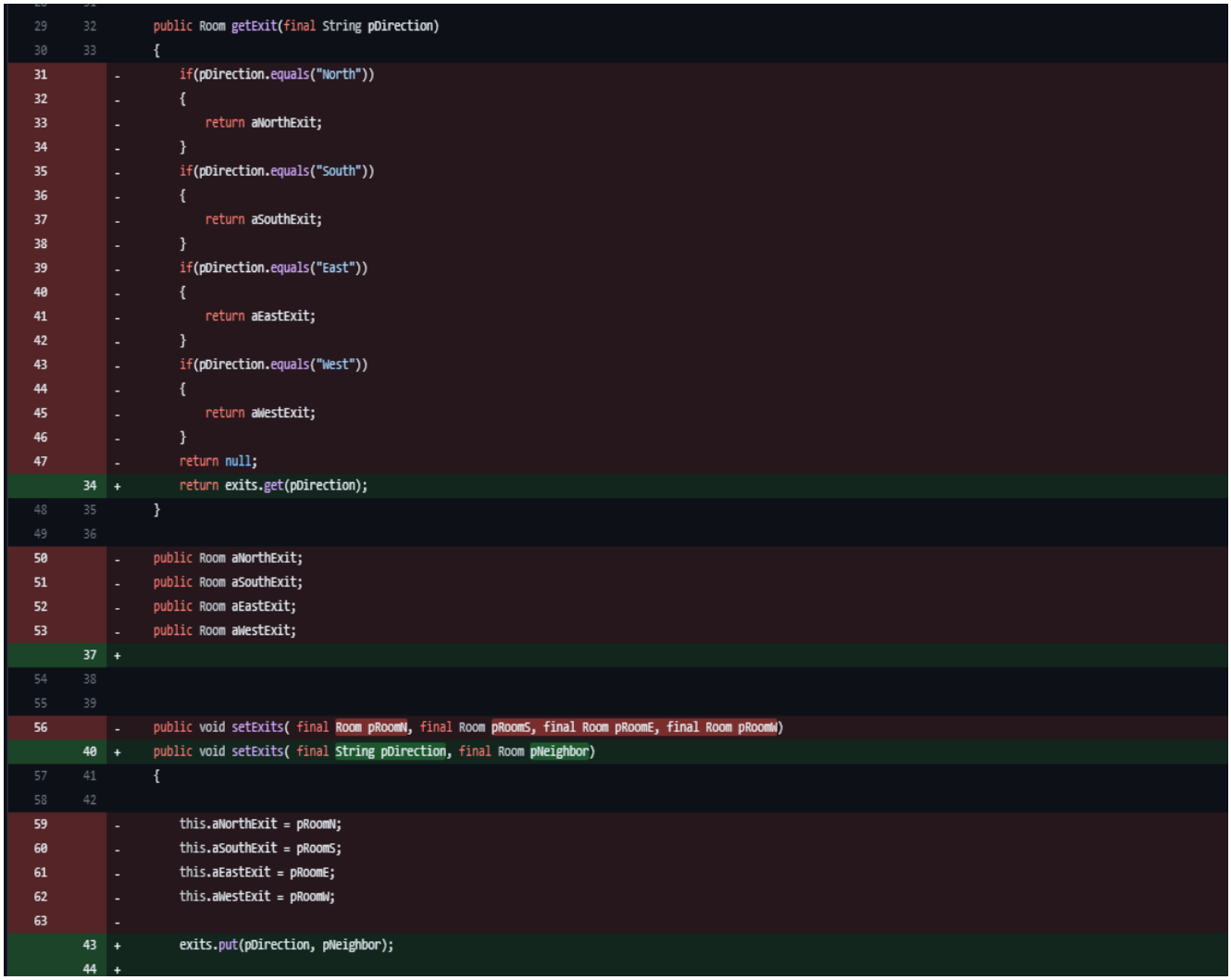
```
private static void printLocationInfo(Room pRoom)
    {
        System.out.println(pRoom.getDescription());
        System.out.print("Exits: ");
        System.out.println(pRoom.getExitString());
    }
```

3. Oui il est logique que de demander à Room de produire ces informations car celles-ci sont stockes chez elle en non dans la classe Game. Il est donc aussi logique que ce soit la classe Game qui affiche les informations et non a Room car les informations produites par Room pourraient servir à autre chose que « juste » les afficher ». C'est donc Game qui choisi ce qu'elle fait de ses information. Dans notre cas, elle les affiches.

7.8

Implémentation des changements décrits dans la partie du livre :

Dans la classe « Room » :



```
29  32      public Room getExit(final String pDirection)
30  33      {
31      -       if(pDirection.equals("North"))
32      -       {
33      -           return aNorthExit;
34      -       }
35      -       if(pDirection.equals("South"))
36      -       {
37      -           return aSouthExit;
38      -       }
39      -       if(pDirection.equals("East"))
40      -       {
41      -           return aEastExit;
42      -       }
43      -       if(pDirection.equals("West"))
44      -       {
45      -           return aWestExit;
46      -       }
47      -       return null;
        34  +       return exits.get(pDirection);
48  35      }
49  36
50      -   public Room aNorthExit;
51      -   public Room aSouthExit;
52      -   public Room aEastExit;
53      -   public Room aWestExit;
        37  +
54  38
55  39
56      -   public void setExits( final Room pRoomN, final Room pRoomS, final Room pRoomE, final Room pRoomW)
        40  +   public void setExits( final String pDirection, final Room pNeighbor)
57  41      {
58  42
59      -       this.aNorthExit = pRoomN;
60      -       this.aSouthExit = pRoomS;
61      -       this.aEastExit = pRoomE;
62      -       this.aWestExit = pRoomW;
63      -
        43  +       exits.put(pDirection, pNeighbor);
        44  +
```

Dans la classe « Game » :

```
37  -       vPub.setExits( null, null, vOutside, null);
38  -       vOutside.setExits( null, vLab, vTheatre, vPub);
39  -       vTheatre.setExits( null, null, null, vOutside);
40  -       vLab.setExits( vOutside, null, vOffice, null);
41  -       vOffice.setExits( null, null, null, vLab);
37  +       vPub.setExits( "East", vOutside);
38  +
39  +       vOutside.setExits( "West", vPub);
40  +       vOutside.setExits( "East", vTheatre);
41  +       vOutside.setExits( "South", vLab);
42  +
43  +       vTheatre.setExits( "West", vOutside);
44  +
45  +       vLab.setExits( "North", vOutside);
46  +       vLab.setExits( "East", vOffice);
47  +
48  +       vOffice.setExits( "West", vLab);
```

La méthode « getExit » est donc maintenant plus précisément « qu'un accesseur » (d'où le « get » dans son nom) permettant d'accéder à l'attributs privé « exits ». On pourrait donc supprimer la méthode getExit et rendre l'attribut « exits » public mais pour éviter les problèmes de couplage, on préfère laisser les attributs privée et créer des accesseurs.

7.8.1

Ajouts d'un lieu en hauteur :

J'ai ajouté « up » et « down » dans la liste des directions valide :

```
private final String[] aValidDirections =
{"North","South","East","West","Up","Down"};
```

J'ai défini les sorties de deux « Room » avec « up » et « down » :

```
vLab.setExits( "Down", vOffice);

vOffice.setExits( "Up", vLab);
```

7.9

La méthode « keyset » est une méthode ne prenant aucun paramètre et qui renvoi un tableau avec toutes les clés de la « hashmap ».

7.10.2

Oui, la JavaDoc de la classe Game est beaucoup plus courte que celle de Room car les méthodes de la classe Game sont pour la plus part privées, elles n'apparaissent donc pas dans la JavaDoc. En revanche, les méthodes de la classe Room sont publics car elles sont utilisée dans la classe Game, elles apparaissent donc dans la JavaDoc.

7.11

J'ai donc rajouter la méthode « getLongDescription » dans la classe Room puis je l'ai utilisé dans la méthode « printCurrentRoomLocationInfo » de la classe Game, qui se contente d'afficher ce que retourne « getLongDescription » :

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

7.14

Pour ajouter une commande « look » qui affiche les infos de la Room actuelle et qui renvoi un message si la command en précéder d'une autre commande j'ai :

- Ajouter le mot « look » au command valide dans la classe « CommandWords » :

```
-   private final String[] aValidCommands =
    {"go","quit","help","look","eat"};
```

- Ajouter une méthode « look » dans la classe « Game » :

```
-     private void look(final Command pLookCommand)
-       {
-           if (pLookCommand.hasSecondWord())
-           {
-               System.out.println("I don't know how to look at something
    in particular yet.");
-           }
-           else
-           {
-               this.printCurrentRoomLocationInfo();
-           }
-
-       }
```

- Ajouter à la méthode « processCommand » de la classe « Game » une ligne exécutant la methode « look » si la command rentrée est « look » :

```
-   }else if(pCommand.getCommandWord().equals("look")){
-               look(pCommand);
-               return false;
-
```

7.15

Pour ajouter une commande « eat » qui affiche un message une fois exécutée, j'ai :

- Ajouter le mot « eat » au command valide dans la classe « CommandWords » :

```
-   private final String[] aValidCommands =
    {"go","quit","help","look","eat"};
```

- Ajouter une méthode « eat » dans la classe « Game » :

```
-    private void eat()
-       {
-           System.out.println("You have eaten now and you are not hungry
    any more.");
-       }
```

- Ajouter à la méthode « processCommand » de la classe « Game » une ligne exécutant la methode « eat » si la command rentrée est « eat » :

```
-   }else if(pCommand.getCommandWord().equals("eat")){
-               eat();
-               return false;
-
```

7.16

Pour ajouter une méthode « ShowAll » qui affiche toute les commandes disponible un fois exécutée, j'ai :

- Ajouter une méthode « showAll » à la classe « CommandWords » qui retourne un string de l'attribut « aValidCommand » contenant toute les commandes reconnues :

```
-    public void showAll()
-       {
-           for(String command : aValidCommands)
-           {
-               System.out.print(command + " ");
-           }
-           System.out.println();
-       }
```

- Ajouter un accesseur de la commande « showAll » dans la classe « Parser » car la classe « Parser » connait la classe « CommandWords » et la classe « Game » connait la classe « Parser » mais la classe « Game » ne connait pas la classe « CommandWords » donc pour éviter le couplage, on ajouter un accesseur dans la classe « Parser » :

```
-   public void showCommands()
-       {
-           commandWords.showAll();
-       }
```

- Remplacer l'affichage d'un string avec les commandes par l'appelle de la méthode « showAll », qui les affiches toutes :

```
-    private void printHelp()
-       {
-           System.out.println("You are lost. You are alone.");
-           System.out.println("You wander around at the university.");
-           System.out.println(" ");
-           System.out.println("Your command words are:");
-           aParser.showCommands();
-       }
-
```

III. Mode d'emploi (si nécessaire, instructions d'installation ou pour démarrer le jeu)

Créer une nouvelle instance de la classe Game et le Jeu démarre.

IV. Déclaration obligatoire anti-plagiat

Je certifie ne pas avoir plagié.

HIGELIN Maxime