OBJECTIF PIRATE

Projet Zuul

Table des matières

Rapport :

Objectif Pirate + https://perso.esiee.fr/~brennera (Non fonctionnel, le php n'est pas interprété)

(A3P(AL) 2022/2023 G2)

I.A) Auteur

Brenner Augustin, Groupe 2

I.B) Thème (phrase-thème validée)

Dans les Caraïbes, un jeune homme doit faire ses preuves pour devenir un pirate.

I.C) Résumé du scénario (complet)

L'histoire se déroule au cours du XVIIe siècle. Un jeune homme nommé Guybrush d'une vingtaine d'années vivant dans les Caraïbes rêve depuis sa naissance de devenir un pirate, il décidât donc de se lancer dans cette quête après son vingtième anniversaire. Cependant, son caractère naïf et son physique faible l'empêcheront d'atteindre son objectif facilement.

L'histoire débute sur une île peuplée de divers individus atypiques que le joueur rencontrera et avec lequel il échangera. Ces protagonistes l'assisteront dans sa quête en marchandant avec lui.

I.D) Plan (complet, avec indication de la partie "réduit" si exercice 7.3.3)

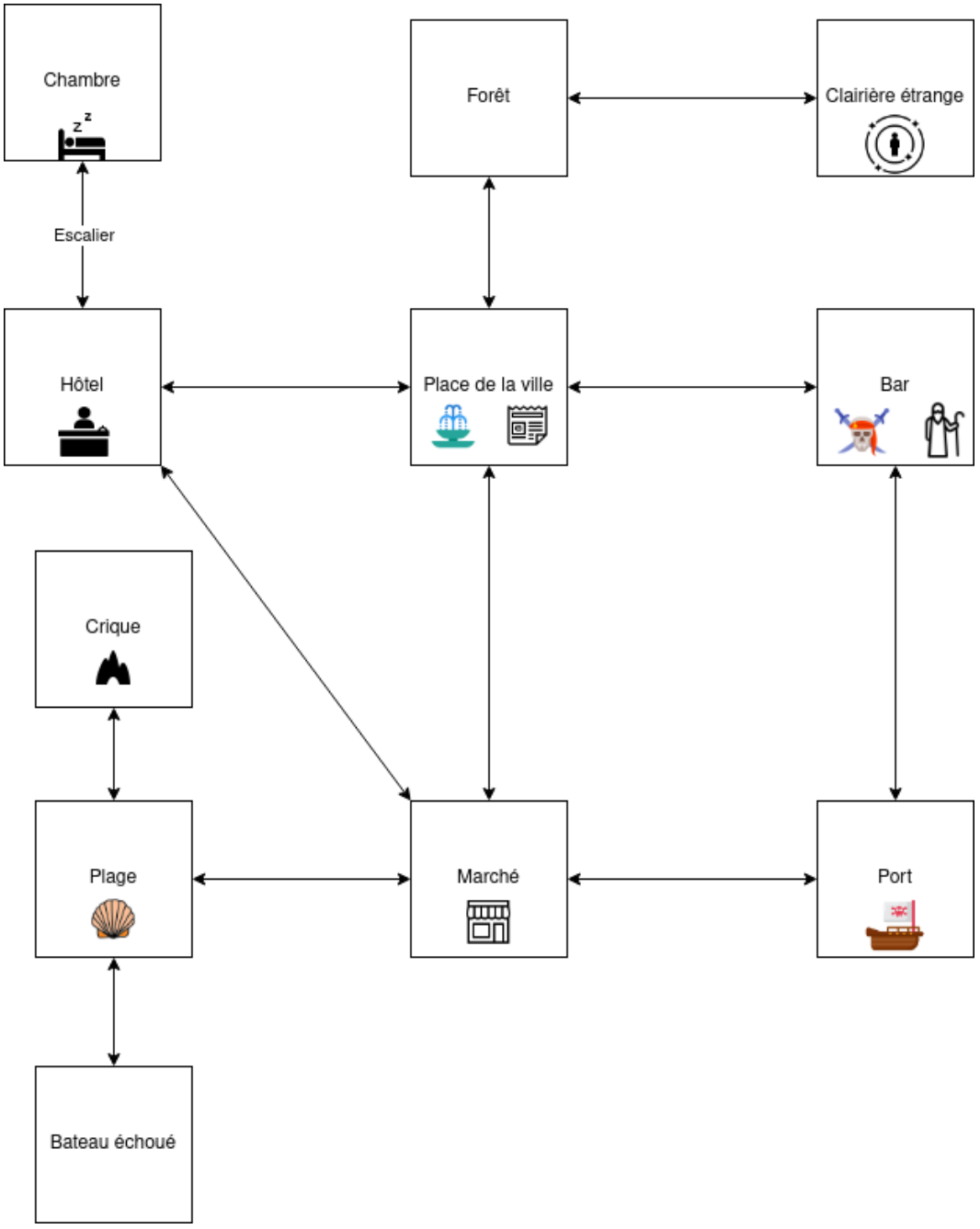

Illustration utilisé pour la carte : icones8.fr

I.E) Scénario détaillé (complet, avec indication de la partie "réduit" si exercice 7.3.3)

Guybrush commence son aventure alors qu'il se promenait dans la place publique de la ville où il ramasse un journal dans lequel il lit un article au sujet d'un groupe féroce de pirate des Caraïbes, ce groupe se réunissait souvent au bar de son village. Le jeune homme y vit donc une occasion d'aller rencontrer ceux qu'il admirait, il se rendit immédiatement au bar pour avoir la chance de rencontrer ses idoles. Une fois arriver au bar, il vit les pirates et tentât de les approcher, cependant dès qu'il commençait à s'approcher de leurs tables, un silence s'installa, Guybrush commença donc à se présenter, mais les pirates commencent à rire et à se moquer de lui en lui disant qu'il n'était pas près et qu'il n'avait pas les capacités pour prétendre devenir un vrai pirate, qu'il ne survivrait pas trois jours. Triste, Guybrush quitta le bar la larme à l'œil. Mais il ne souhaitait pas abandonner son rêve. À la sortie du bar, il rencontra un vieil homme qui avait assisté à la scène, celui-ci lui dit qu'il fit partie d'un des groupes de pirates les plus féroces des Caraïbes qui avait volé des milliers de trésors, celui-ci était venu s'exiler sur cette île pour fuir ses ennemis. Il lui dit qu'en échange de services, il était prêt à le former et le rendre plus féroce, mais pour cela le jeune homme va devoir se surpasser. Le personnage va donc tout mettre en œuvre pour réaliser des missions et résoudre des énigmes dans le but d'atteindre son objectif.

I.F) Détail des lieux, items, personnages

Personnages :

- Guybrush
- Groupes de pirates
- Ancien pirate
- Marchand
- Armateur
- Réceptionniste de l'hôtel

Lieux :

- Place du village :
    - Description : Point central du village.
    - Items :
        - Journal au début : article pirate.
- Hôtel :
    - Description : Un endroit permettant de se reposer.
- Hôtel 1ère étage :
    - Description : accès à une chambre.
    - Action :
        - Le joueur peut aller dormir.
- Bar :
    - Description : QG des pirates.
- Forêt :
    - Description : Lieu étrange et peu rassurant.
- Clairière étrange :
    - Description : Petite clairière au milieu de la forêt avec une grande pierre au milieu.
    - Action :
        - À chaque fois qu'une personne s'approche trop près de la pierre, celui-ci s'endort rapidement sur l'herbe et se réveille une heure plus tard dans un autre endroit qui n'est jamais le même.
- Marché :
    - Description : Lieu où il est possible d'acheter divers objets.
    - Action :
        - Le joueur peut y acheter divers équipements.
- Port :
    - Description : Lieu où d'immenses bateaux appartenant aux plus grands pirates sont amarrés, certains contiennent des trésors précieux.
    - Action :
        - Le joueur peut y rencontrer l'armateur.
- Plage :
    - Description : Grande plage de sables blancs.
- Crique :
    - Description : Difficile d'accès et dangereuse.
- Bateau échoué :

- Description : Un grand bateau de pirate écrasé contre un immense rocher.

I.G) Situations gagnantes et perdantes

Situation gagnante :

- Guybrush est connu dans tous les Caraïbes comme un pirate féroce, il est redouté de tous.

Situations perdantes :

- Guybrush est mort.
- Guybrush est humilié et tout le monde se moque de lui, il a échoué dans sa quête et tout le monde le voie comme un incapable.

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

- Enigmes :
  - …
  - …
  - …
- Combats :
  - …
  - …
  - …

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

Todo List :

- ☐ Rendre le site fonctionnel
- ☐ Définir les énigmes
- ☐ Définir les combats

II.) Réponses aux exercices

Exercice 7.5 :

J'ai créé la procédure "printLocation()" dans la classe "Game" qui permet d'afficher la localisation ainsi que les sorties de la pièce où se trouve le joueur, on peut ensuite appelé cette procédure dans la méthode "printWelcome()" pour que le joueur sache où il se situe et puisse savoir où il peut aller. J'ai aussi appelé cette méthode dans la fonction "goRoom()" pour qu'après chaque déplacement le joueur connaisse la room dans laquelle il est et les sorties. Cela permet de remplacer l'affichage des sorties dans les méthodes "printWelcome()" et "goRoom()."

```
/**
* Procédure qui affiche les informations de la localisation
*/
private void printLocationInfo(){
        System.out.println("You are " + aCurrentRoom.getDescription());

        System.out.println("Exits : ");
        if(aCurrentRoom.aNorthExit != null){
            System.out.println("north ");
        }
        if(aCurrentRoom.aSouthExit != null){
            System.out.println("south ");
        }
        if(aCurrentRoom.aEastExit != null){
            System.out.println("east ");
        }
        if(aCurrentRoom.aWestExit != null){
            System.out.println("west ");
        }
        if(aCurrentRoom.aNorthWestExit != null){
            System.out.println("north-west ");
        }
        if(aCurrentRoom.aSouthEastExit != null){
            System.out.println("south-east ");
    }
```

Exercice 7.6 :

J'ai créé l'accesseur "getExit()" dans la classe "Room" qui renvoie les sorties situé dans la direction indiquée ou "null" s'il n'y en a pas. La création de cet accesseur permet de simplifier le code ainsi que de pouvoir ajouter plus facilement des directions sans avoir à modifier plusieurs parties du code. Dans la classe "Game", cela m'a permis de définir la salle suivante en fonction de la direction indiquée par le joueur plus facilement. En effet je n'ai plus qu'as appelé l'accesseur "getExit()" et exécuté différente instruction en fonction de la direction fournie par le joueur.

Dans la classe "Room" :

```
    /**
     * Renvoie les valeurs des différentes sorties en fonction de la direction indiquée. La
valeur retournée est de type "Room"
     * @param -> String : direction
     * @return -> Room : sortie dans la direction indiquée par le joueur
     */
    public Room getExit(final String pDirection){
        if (pDirection.equals("north")){
            return aNorthExit;
        }

        if (pDirection.equals("south")){
            return aSouthExit;
        }

        if (pDirection.equals("east")){
            return aEastExit;
        }

        if (pDirection.equals("west")){
            return aWestExit;
        }
        if (pDirection.equals("north-west")){
            return aNorthWestExit;
        }
        if (pDirection.equals("south-east")){
            return aSouthEastExit;
        }
        return null;
    }
```

Dans la classe "Game" :

```java
/**
* Procédure qui permet un changement de pièce
* @param -> pInstruction (Command)
*/
private void goRoom(final Command pInstruction)
{
    // Verification que le joueur à bien indiquer une direction
    if (!pInstruction.hasSecondWord()){
        System.out.println("Go where ?");
        return;
    }

    Room vNextRoom = null; // définition de la prochaine pièce
    String vDirection = pInstruction.getSecondWord(); // la direction indiqué par
le joueur
    boolean vUnknowDirection = true; // permet de savoir si le joueur a donné une
direction valide
    String[] vTabDirection = {"north", "south", "east", "west", "up", "down",
"north-west", "south-east"}; // tableau des directions possibles

    // Verification de la commande entré par le joueur
    for (int i = 0; i < vTabDirection.length; i++){
        if (vDirection.equals(vTabDirection[i])){
            vNextRoom = aCurrentRoom.getExit(vDirection);
            vUnknowDirection = false;
        }
    }

    // Si la commande ne correspond à aucune direction
    if(vUnknowDirection == true){
        System.out.println("Unknown direction!");
        return;
    }
    // Sinon si il n'y a pas de piece dans la direction indiqué par le joueur
    else if(vNextRoom == null){
        System.out.println("There is no door !");
        return;
    }

    this.aCurrentRoom = vNextRoom; // définie la prochaine piece

    printLocationInfo(); // affiche la description de la piece
}
```

Exercice 7.7 :

Pour rendre le code plus clair et éviter la duplication de code, j'ai créé un accesseur dans la classe "Room" nommée "getExitString()" celui-ci permet de renvoyer sous forme d'une chaîne de caractère les différentes directions possibles pour sortir de la pièce ou le joueur se situe. J'appelle donc cette fonction dans la méthode

“printLocationInfo()” de la classe “Game” afin de remplacer le code précédent qui était plus complexe. L’intérêt de mettre l’accesseur dans la classe “Room” est de regrouper les méthodes qui agiront sur les différentes salles. De plus, il est logique de demander à la classe “Game” d’afficher les informations de la classe “Room” car cette classe sera le moteur du jeu.

Accesseur “getExitString()” dans la classe “Room” :

```
    /**
     * Cet accesseur renvoie les sorties de la salle dans laquelle le joueur est présent
sous forme de String
     * @return -> String : liste des différentes directions permettant de sortir de la
pièce
     */
    public String getExitString(){
        String vExit = "Exits : ";
        if(this.getExit("north") != null){
            vExit += "north ";
        }

        if(this.getExit("south") != null){
            vExit += "south ";
        }

        if(this.getExit("east") != null){
            vExit += "east ";
        }

        if(this.getExit("west") != null){
            vExit += "west ";
        }
        if(this.getExit("north-west") != null){
            vExit += "north-west ";
        }
        if(this.getExit("south-east") != null){
            vExit += "south-east ";
        }
        return vExit;
    }
```

Méthode “printLocationInfo()” dans la classe “Game” :

```
    /**
     * Procédure qui affiche les informations de la localisation
     */
    private void printLocationInfo(){
        System.out.println("You are " + aCurrentRoom.getDescription());
        System.out.println(this.aCurrentRoom.getExitString());
    }
```

Exercice 7.8 :

Pour cet exercice, j'ai d'abord dû importer deux librairies dans la classe "Room" :

```
import java.util.HashMap;
import java.util.Set;
```

Celles-ci vont nous permettre de définir de nouvelle direction, ce qui permettra au joueur de monter, descendre ou d'aller vers le nord-ouest par exemple.

Tout d'abord j'ai remplacé les six attributs qui définissent les directions des sorties :

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

par une HashMap :

```
private HashMap<String, Room> aExits;
```

Comme les attributs ont été modifié, il faut maintenant modifier le constructeur naturel en ajoutant l'instruction permettant de créer une nouvelle HashMap:

```
/**
 * Constructeur naturel
 * @param pDescription -> La description de la pièce
 */
public Room(final String pDescription){
	this.aDescription = pDescription;
	aExits = new HashMap<String, Room>();
}
```

La fonction "setExit()" peut être modifiée pour définir les sorties, en effet nous pouvons utiliser une HashMap, cela permet de ne pas avoir à réaliser différent test sur les directions pour voir si celle-ci sont "null" ou non puisqu'une HashMap le fait automatiquement.

```
/**
* Définition des sorties de chaque piece. Chaque direction conduit à une autre pièces ou
est "null"
```

```
* @param pDirection -> La direction de la sortie
* @param pNeighbor -> La piece dans la direction indiqué
*/
public void setExits(final String pDirection, final Room pNeighbor){
	aExits.put(pDirection, pNeighbor);
}
```

Comme nous venons de modifier cette méthode et que celle-ci est utilisée dans la classe “Game” pour la procédure “createRoom()” nous devons aussi modifier cette méthode. La modification de “setExit()” nous permet de ne plus être obligé de spécifier qu’une sortie est “null” quand la pièce ne contient pas de sortie dans une direction. Nous obtenons donc cela :

```
/**
* Procédure de création de Room
*/
private void createRooms(){
	Room vPlace = new Room("In the public place");
	Room vForest = new Room("In the forest");
	Room vClearing = new Room("In the strange clearing");
	Room vBar = new Room("In the bar");
	Room vHotel = new Room("In the hotel");
	Room vMarket = new Room("In the market");
	Room vPort = new Room("In the port");
	Room vCove = new Room("On the cove");
	Room vAgroundBoat = new Room("In the aground boat");
	Room vBeach = new Room("On the beach");

	// Sortie de la place
	vPlace.setExits("north", vForest);
	vPlace.setExits("south", vMarket);
	vPlace.setExits("east", vBar);
	vPlace.setExits("west", vHotel);

	// Sortie de la foret
	vForest.setExits("south", vPlace);
	vForest.setExits("east", vClearing);

	// Sortie de la clairière
	vClearing.setExits("west", vForest);

	// Sortie du bar
	vBar.setExits("west", vPlace);
	vBar.setExits("south", vPort);

	// Sortie de l'hotel
	vHotel.setExits("east", vPlace);
	vHotel.setExits("south-east", vMarket);
```

```
        // Sortie du marché
        vMarket.setExits("north", vPlace);
        vMarket.setExits("east", vPort);
        vMarket.setExits("west", vBeach);
        vMarket.setExits("north-west", vHotel);

        // Sortie du port
        vPort.setExits("west", vMarket);

        // Sortie de la crique
        vCove.setExits("south", vBeach);

        // Sortie du bateau échoué
        vAgroundBoat.setExits("north", vBeach);

        // Sortie de la plage
        vBeach.setExits("north", vCove);
        vBeach.setExits("south", vAgroundBoat);
        vBeach.setExits("east", vMarket);

        this.aCurrentRoom = vPlace;
}
```

Il est aussi nécessaire de modifier la méthode “getExit()” dans la classe “Room” pour que celle-ci puisse continuer de fonctionner suite au modification apporté au attributs :

```
/**
* Renvoie les valeurs des différentes sorties en fonction de la direction indiquée. La
valeur retourné est de type "Room"
* @param pDirection -> String : direction
* @return -> Room : sortie dans la direction indiqué par le joueur
*/
public Room getExit(final String pDirection){
        return aExits.get(pDirection);
}
```

Exercice 7.8.1 :

Dans mon scénario, j'ai rajouté une chambre située au première étage de l'hôtel. J'ai donc dû rajouter la possibilité au joueur de monter et de descendre.

Premièrement dans la classe "Game", dans la méthode "createRooms()" j'ai rajouté la chambre. J'ai aussi rajouté une sortie dans la direction "up" depuis l'hôtel qui permet d'accéder à la chambre, ainsi que la sortie dans la direction "down" de la chambre vers l'hôtel. J'ai donc obtenue ceci :

```
/**
* Procédure de création de Room
*/
private void createRooms(){
        ...
        ...
        ...
        Room vBedroom = new Room("A classic bedroom with a comfortable bed");

        ...
        ...
        ...


        // Sortie de l'hotel
        vHotel.setExits("south-east", vMarket);
        vHotel.setExits("east", vPlace);
        vHotel.setExits("up", vBedroom);

        // Sortie de la chambre
        vBedroom.setExits("down", vHotel);

        ...
        ...
        ...

        this.aCurrentRoom = vPlace;
}
```

Ensuite, j'ai rajouté la possibilité au joueur de pouvoir aller dans les directions "up" et "down". J'ai donc modifié la méthode "goRoom()" dans la classe "Game" :

```java
/**
* Procédure qui permet un changement de pièce
* @param -> pInstruction (Command)
*/
private void goRoom(final Command pInstruction) {
	// Verification que le joueur à bien indiquer une direction
	if (!pInstruction.hasSecondWord()){
		System.out.println("Go where ?");
		return;
	}

	Room vNextRoom = null; // définition de la prochaine piece
	String vDirection = pInstruction.getSecondWord(); // la direction indiqué par le
joueur
	boolean vUnknowDirection = true; // permet de savoir si le joueur a donné une
direction valide
	String[] vTabDirection = {"north", "south", "east", "west", "up", "down",
"north-west", "south-east"};
 // tableau des directions possibles

	// Verification de la commande entré par le joueur
	for (int i = 0; i < vTabDirection.length; i++){
		if (vDirection.equals(vTabDirection[i])){
		    vNextRoom = aCurrentRoom.getExit(vDirection);
		    vUnknowDirection = false;
		}
	}

	// Si la commande ne correspond à aucune direction
	if(vUnknowDirection == true){
		System.out.println("Unknown direction!");
		return;
	}
	// Sinon si il n'y a pas de piece dans la direction indiqué par le joueur
	else if(vNextRoom == null){
		System.out.println("There is no door !");
		return;
	}

	this.aCurrentRoom = vNextRoom; // définie la prochaine piece

	printLocationInfo(); // affiche la description de la piece
}
```

Exercice 7.9 :

En supprimant les attributs qui correspondent aux directions des sorties, la fonction “getExitString()” dans la classe “Room” ne pouvait plus renvoyer les sorties car celle-ci reposait sur les anciens attributs. Il a donc fallu l'adapter, pour

cela on à déclarer un ensemble qui regroupe toutes les clés de la HashMap que l'on parcourt avec une boucle for each afin d'obtenir les éléments associés au clés, puis on les ajoute successivement à une chaîne de caractère qui pourra être renvoyé à la fin.

Ce qui nous donnes :

```
/**
* Cette fonction renvoie les sorties de la salle dans laquel le joueur est présent sous
forme de String
* @return -> String : liste des différentes directions permettant de sortir de la pièce
*/
public String getExitString(){
	String vExitString = "Exits : ";
	Set<String> vKeys = this.aExits.keySet();
	for (String vExit : vKeys){
		vExitString += " " + vExit;
	}
	return vExitString;
}
```

Exercice 7.10 :

La fonction "getExitString()" va renvoyer les différentes sorties auxquelles le joueur a accès en fonction de la salle où il se situe. Pour cela, on crée tout d'abord un ensemble qui va contenir les clés des différentes sorties. En suite à l'aide d'une boucle for each ont peut parcourir l'ensemble des sorties associé aux clés de la HashMap, et ainsi les ajoutés au fur et à mesure à une chaîne de caractère que nous renverrons ensuite.

Exercice 7.10.1 :

Fait

Exercice 7.10.2 :

Fait

Exercice 7.11 :

Nous allons maintenant créer une nouvelle méthode dans la classe “Room”, celle-ci devra donner une description complète de la pièce dans laquelle se situe le joueur, comme par exemple les items, les montres ou bien les autres personnages présents dans la salle.

On crée donc la méthode “getLongDescription()” dans la classe “Room” :

```
/**
* Retourne une description complete de cette pièce, de la forme :
*     You are in the kitchen.
*     Exits : north west
* @return -> String : description de la pièce
*/
public String getLongDescription(){
   return "You are " + this.aDescription + ".\n" + this.getExitString();
}
```

Désormais on peut donc modifié la procédure “printLocationInfo()” dans la classe “Game” pour y intégrer la nouvelle méthode “getlongDescription()” et ainsi la simplifié :

```
/**
* Procédure qui affiche les informations de la localisation
*/
private void printLocationInfo(){
   System.out.println("You are " + this.aCurrentRoom.getLongDescription());
}
```

Exercice 7.14 :

Afin de simplifier le code dans le future et ajouter de nouvelle commande plus facilement, j’ai attribué au tableau de chaînes de caractères “aValidCommands” (l'attribut de la classe “CommandWords”) l’ensemble des commandes valides qui peuvent être saisies par le joueur.

Nous allons rajouter la commande “look” qui permettra au joueur d'accéder à la description complète de la salle dans laquelle il se situe. Pour cela nous

commençons par ajouter “look” à la liste des command valid dans le tableau static “aValidCommand” dans la classe “CommandWords()” :

```
// un tableau constant qui contient toutes les commandes valides
private static final String aValidCommands[] = {
	"go", "quit", "help", "look"
};
```

Ensuite, il faut ajouter une procédure “look()” dans la classe “Game” qui vas permettre d’afficher la description :

```
/**
* Procédure qui permet d'afficher la description de la pièce
*/
private void look(){
System.out.println(this.aCurrentRoom.getLongDescription());
}
```

Enfin, il nous reste à faire en sorte que cette procédure soit appelée dès que l’utilisateur utilise la commande “look”. Pour cela, nous devons modifier la fonction “processCommand()” dans la classe “Game” :

```
/**
* Procédure qui permet de traiter les commandes
* @param pInstruction (Command) -> Commande entré par le joueur
* @return -> boolean (true si la commande est "quit" et que le joueur souhaite quitter le
jeu)
*/
private boolean processCommand(final Command pInstruction){
	if(pInstruction.isUnknown()){
		System.out.println("I don't know what you mean...");
		return false;
	}else if(pInstruction.getCommandWord().equals("quit")){
		return quit(pInstruction);
	}else if (pInstruction.getCommandWord().equals("help")){
		printHelp();
		return false;
	}else if(pInstruction.getCommandWord().equals("go")){
		goRoom(pInstruction);
		return false;
	} else if (pInstruction.getCommandWord().equals("look")) {
		if (pInstruction.hasSecondWord()){
			System.out.println("I don't know how to look at something in
particular yet");
		}else {
			look();
```

```
            }
            return false;
      }else{
            System.out.println("I don't know what you mean...");
            return false;
      }
}
```

Désormais si le joueur exécute la commande “look” suivie d’aucun mot, la description de la pièce sera affichée.

Exercice 7.15 :

Pour ajouter la commande “eat” le procédé est le même que pour la commande “look”.

Il faut d’abord l’ajouter à la liste des commandes valides :

```
// un tableau constant qui contient toutes les commandes valides
private static final String aValidCommands[] = {
      "go", "quit", "help", "look", "eat"
};
```

Ensuite, il faut ajouter une procédure dans la classe “Game” qui réalise l’action :

```
/**
* Procédure qui permet au personnage de manger
*/
private void eat(){
      System.out.println("You have eaten now and you are not hungry any more.");
}
```

Enfin, il faut exécuter cette fonction quand le joueur écrit la commande “eat”, pour cela on va modifier la fonction “processcommand()” :

```
/**
* Procédure qui permet de traiter les commandes
* @param pInstruction (Command) -> Commande entré par le joueur
* @return -> boolean (true si la commande est "quit" et que le joueur souhaite
quitter le jeu)
*/
private boolean processCommand(final Command pInstruction){
```

```
    if(pInstruction.isUnknown()){
        System.out.println("I don't know what you mean...");
        return false;
    }else if(pInstruction.getCommandWord().equals("quit")){
        return quit(pInstruction);
    }else if (pInstruction.getCommandWord().equals("help")){
        printHelp();
        return false;
    }else if(pInstruction.getCommandWord().equals("go")){
        goRoom(pInstruction);
        return false;
    } else if (pInstruction.getCommandWord().equals("look")) {
        if (pInstruction.hasSecondWord()){
             System.out.println("I don't know how to look at something in particular
yet");
        }else {
            look();
        }
        return false;
    }else if (pInstruction.getCommandWord().equals("eat")) {
        if (pInstruction.hasSecondWord()){
            System.out.println("What do you want to eat ?");
        }else {
            eat();
        }
        return false;
    }else{
        System.out.println("I don't know what you mean...");
        return false;
    }
}
```

Exercice 7.16 :

Actuellement, chaque nouvelle commande doit être ajouter manuellement à la procédure “printHelp()” qui liste les commandes que le joueur peut exécuter, pour rendre cela plus simple et automatique on peut tout d’abord créer une méthode qui vas lister l’ensemble des commandes disponibles :

```
/**
* Print all valid commands to System.out.
*/
public void showAll(){
   for(String command : this.aValidCommands) {
       System.out.print(command + "  ");
   }
   System.out.println();
}
```

Pour ne pas relié directement la classe “CommandWords” à la classe “Game” on vas créer une nouvelle méthode dans la classe “Parser” qui est relié à la classe “Game” cette méthode appelles juste la procédure “showAll()” de la classe “CommandWords” :

```
/**
* Print out a list of valid command words.
*/
public void showCommands(){
   this.aValidCommands.showAll();
}
```

Enfin, il nous reste plus qu’à appeler cette méthode dans la procédure “printHelp()” de la classe “Game” :

```
/**
* Procédure qui affiche le message d'aide
*/
private void printHelp(){
    System.out.println("You are lost. You are alone. \n You wander around the
island. \n \n Your command words are: ");
   this.aParser.showCommands();
}
```

Exercice 7.17 :

Pour ajouter une nouvelle commande, il suffit de l'ajouter dans le tableau de String "aValidCommands" dans la classe "CommandWords". A partir de ce moment-là le joueur peut entrer la commande sans que le programme renvoie "I don't know what you mean...". Cependant la commande ne réalise aucune action, en effet dans la classe "Game" il faut créer une procédure qui réalise l'action voulue, et exécuter cette procédure quand l'utilisateur rentre la commande. Cela se fait dans la fonction "process command()".

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

IV. Déclaration obligatoire anti-plagiat (*)

Je certifie ne pas avoir effectué de plagiat ou de copier/coller dans mes codes, et d'avoir respecté les droits d'auteur dans mes travaux.

V, VI, etc... : tout ce que vous voulez en plus