Rapport du jeu d’aventure

A. Titre
B. Auteur

Ce rapport a éte réalisé par PEREIRA CARREIRA Samuel.

C. Thème

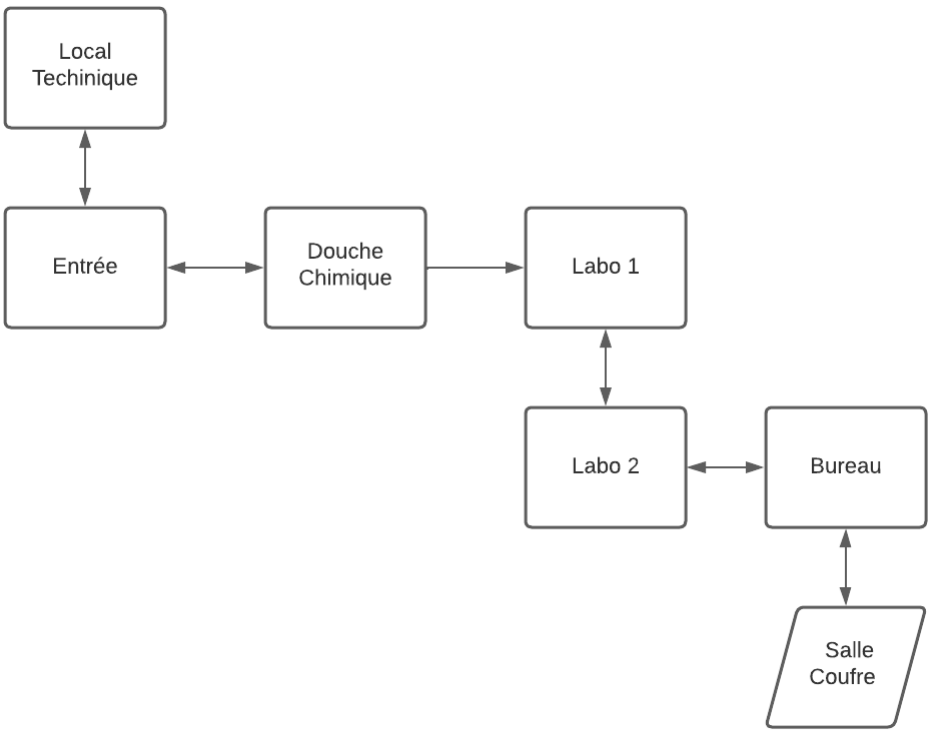
Dans un laboratoire, un chercheur fraichement diplômé doit créer un vaccin pour sauver son fils à temps.

D. Résumé du scénario

Vous êtes à l’entrée du laboratoire, vous allez voir le directeur du laboratoire dans son bureau pour avoir son autorisation. Il vous la donne à condtion que vous obtenez la seringue dans un délai imparti. Dans le cas du non respect du temps, vous perdez. Sinon, le directeur vous autorise à récuperer dans la salle sécurisée un échantillon du virus. En possession du virus, il ne vous reste plus qu’à créer la formule du vaccin et assembler le tout pour gagner.

E. Plan

Le plan du jeu d’aventure est le suivant:



F. Scénario détaillé

Vous êtes dans l'entrée du laboratoire, vous allez à l'accueil récuperer le protocole imprimé à l'avance contenant les étapes nécessaires à la production du vaccin. D'après le protocole, vous devez demander l'autorisation du directeur de recherche pour la création du vaccin. Pour cela, vous obtenez de l'acceuil la position du directeur se trouvant dans le bureau. Après entretien avec le directeur, son approbation n'est possible qu'à condition d'être en possession de la seringue type Z dans un délai imparti. Pour acquerir cette seringue, le directeur vous redirige vers Michel, responsable materiel du laboratoire, que vous devez trouver. Michel explique que pour la conception particuliére de cette seringue, il faut récuperer dans le local technique un verre de haute qualité et métal d'une precision chirurgical que vous assemblerez dans le labo 1. (mini jeu 1) La seringue étant créée, vous pouvez retourner voir le directeur pour passer à l'étape suivante. Pour passer l'étape suivante le directeur vous autorise à prendre un échantillon du virus contenu dans la salle sécurisée à condition de passer toutes les sécurités (mini jeu 2). Une fois à l'interieur de la salle sécurisée, pour sortir le virus de son emplacement, il vous faudra trouver un code secret qui est caché dans la salle.(mini jeu 3) Une fois en possession du virus, vous vous dirigerez vers le labo 2 pour créer la formule chimique du vaccin. (mini jeu 4). Cette étape étant réalisée, vous irez dans le labo 1 pour assembler la seringue et le produit et ainsi obtenir le vaccin.

G. Détail des lieux, items, personnages

a. Personnages

i. Personnage incarné par le joueur

Albert, un chercheur fraichement diplômé dans le biomédical, cherche à sauver son fils atteint d'un virus mortel.

ii. Autres personnages

Bernard : responsable de l'acceuil.
Louis : Directeur du laboratoire.
Joachim : responsable du matériel.

H. Situations gagnantes et perdantes

Pour terminer le jeu, il faut le fils d'Albert, pour cela il faut suivre tout le processus de la création du vaccin dans le temps imparti. Si cette limite de temps n'est pas respectée, le fils d'Albert meurt, ainsi, le joueur finit perdant.

I. Énigmes, mini-jeux, combats

J. Commentaires

K. Réponses aux exercices

a. L'objet Scanner (exercice 7.2.1)

Nous avons utilisé un objet Scanner dans la classe Parser du jeu d'aventure afin de lire au clavier les commandes de l'utilisateur.

```
this.aReader         = new Scanner( System.in );
```

Nous avons également utilisé un autre objet Scanner dans la même classe pour analyser une chaîne de caractères mot par mot. Dans le projet, nous avons tout d'abord créé un nouvel objet scanner en lui passant la chaîne de caractères en question.

```
Scanner vTokenizer = new Scanner( vInputLine );
```

Ensuite, nous devons vérifier s'il existe un prochain mot à analyser à l'aide de la méthode *hasNext().* Si oui, nous le récupérons avec la suivante instruction :

```
vTokenizer.hasNext()
```

Par la suite, nous vérifions s'il existe un second mot, si oui, nous le récupérons à l'aide de l'instruction précédente. Lorsque nous vérifions s'il existe un mot, la méthode ignore le reste de la ligne tapée par l'utilisateur.

b. La méthode *printlocationInfo* (exercice 7.5)

Les méthodes *printWelcome* et *goRoom* de la classe *game* affichent l'information de la situation actuelle. Afin d'éviter la duplication de code au sein du projet, la procédure *printlocationInfo* a été implémentée dans la classe game. Elle affiche la description de la situation courante et les sorties possibles.

c. La méthode *getExit* (exercice 7.6)

La procédure *setExits* du livre teste si chaque paramètre est égal à *null* afin de vérifier si la pièce existe. Par exemple, le code ci-dessous vérifie s'il y a une pièce appelée north . Si oui, on définit une sortie nord.

```
if(north != null) {
    northExit = north;
}
```

Capture d'écran de la page 211 du livre

L'accesseur *getExit* que nous avons implémenté dans la classe *Room* sert à réduire le couplage entre classes. Nous pouvons ainsi modifier la manière dont nous accédons aux attributs de la classe Room.

d. La fonction *getExitString (exercice 7.7)*

…

e. HashMap (exercice 7.8)

Une HashMap est une manière de stocker des éléments du même type. Au contraire d'un tableau, l'HashMap est extensible, c'est-à-dire, que la taille peut être modifiable et l'accès aux éléments de la HashMap peut se réaliser via à des clés de n'importe quel type.
Tout d'abord, nous avons déclaré une HashMap dans la classe *Room*.

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

Ensuite, tout objet du type *Room* doit créer une nouvelle hashMap, nous avons ainsi modifier le constructeur de la classe *Room* en ajoutant l'expression suivante:

```
this.aExits = new HashMap<String, Room>();
```

Puis, la fonction *getExit* de la classe Room a été également mise à jour avec la HashMap. Cependant, la procédure *setExits* était remplacée par une procédure appelée *setExit*:

```
public void setExit(final String pDirection, final Room pNeighbor )
{
    aExits.put(pDirection, pNeighbor);
}
```

Nous devons maintenant modifier la procédure *createRooms* dans la classe *Game*.

On remplace

```
vEntree.setExits(vLocalTecnique,null,null,vDoucheChimique);
```

par

```
vEntree.setExit("North",vLocalTecnique);
vEntree.setExit("East",vDoucheChimique);
```

f. Déplacement vertical (exercice 7.8.1)

Nous avons ajouté deux déplacements: up et down. Nous avons fait deux modifications. La première sur la méthode *createRooms.*

```
vBureau.setExit("Down", vSalleCoufre);
vSalleCoufre.setExit("Up", vBureau);
```

La seconde sur la méthode *goRoom.*

```
else if(vDirection.equals("Down"))
{
    vNextRoom = this.aCurrentRoom.getExit("Down");
}
else if(vDirection.equals("Up"))
{
    vNextRoom = this.aCurrentRoom.getExit("Up");
}
```

g. keySet (exercice 7.9)

Comme, nous avons introduit une HashMap dans la classe *Room*, la fonction getExitString n'est plus valable. Nous devons donc utiliser la méthode keySet pour itérer les éléments de la HashMap. Les modifications sont les suivantes:

```
public String getExitString()
{
    String vExits = "";

    for ( String vS : this.aExits.keySet() )
        vExits= vExits + " " + vS ;
    return vExits.toString();

}
```

h. getLongDescription (exercice 7.11)

Nous procédons à ajouter une méthode dans la classe *Room*, avec l'objectif de réduire le couplage entre la classe *Room* et *Game.*

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

Dans la classe *Game*, nous écrivons

```
System.out.println(this.aCurrentRoom.getLongDescription());
```

i. look (exercice 7.14)

Nous avons ajouté une nouvelle commande appelée “look”. Elle sert à avoir la description détaillée de la pièce. Il faut donc modifier la méthode *processCommand,* en vérifiant si la commandWord est égale à “look”.

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

j. eat (exercice 7.15) (le nombre de commandes disponibles ne sont pas encore définies.)

k. showAll, showCommands (exercice 7.16)

La méthode *printHelp* doit être modifiée à chaque fois qu’une nouvelle commande est introduite, cela pose un problème fondamentale au niveau du code, c'est pourquoi qu’une nouvelle méthode est introduite appelée *showAll* dans la classe *Command*.

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

Nous devons également ajouter une méthode dans la classe *Parser* appelée *showCommands.*

```
public void showCommands()
{
    aValidCommands.showAll();
}
```

Pour finir, nous ajoutons ces deux lignes de code dans la méthode *printHelp.*

```
System.out.println(vHelp);
aParser.showCommands();
```

L. Mode d’emploi
M. Déclaration anti-plagiat