Réalisation d'application

Rapport itération III

Jeu D'aventure

Zuul-bad

Auteurs
Groupe 5
LOKO Loïc
GHOUIBI Ghassen
BOUCHICHA Abdelrahim

Table des matières

I Présentation

Dans le cadre du cours "Réalisation d'application", nous avons eu la tâche de réaliser l'itération 1 du jeu d'aventure intitulé "Zuul".
Zuul-bad est une version de world-of-zuuul c'est un jeu d'aventure qui était développer en 1970 par Will Crowther et étendu par Don Woods.Le jeu original est connu sous le nom Colossal Cave Adventure.
Le principe du jeu c'est trouvé un chemin dans un système compliqué de cave autrement dit une labyrinthe avec des portes et le joueur doit essayer de trouver des mots secrets et des trésors cachés et autres.Toute en visant de recolter le maximum de point possible. Dans ce rapport on se base sur Zuul-bad c'est une implémentation avec des mauvais design de classe.
Le but de cette itération est de découvrir certains facteurs qui influencent la conception d'une classe.
Dans cette itération, en répondant aux différentes questions, nous avons découvert des principes importants dans la conception d'une classe. Ce rapport contient donc les différentes réponses aux questions.

II Itération I

II.1 Exercice 7.1

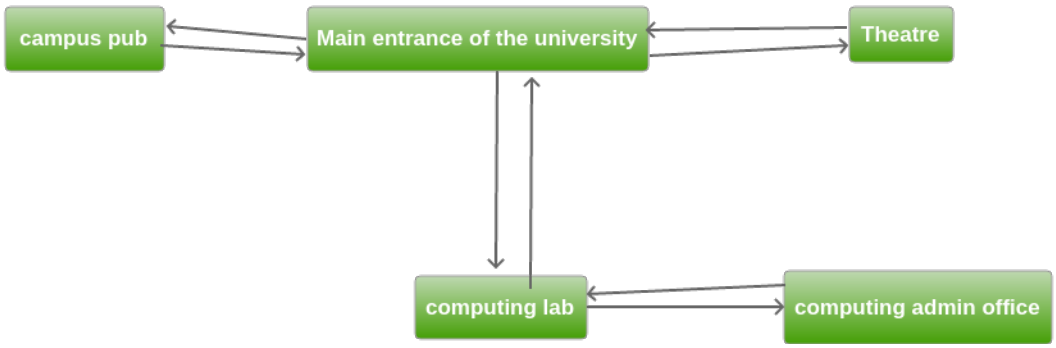
7.1.1 Que fait cette application ?
Cette application est implémenté en mode texuelle permet le joueur de se balader dans des pièces a traves l'interaction en ligne de commande (ou autres).

7.1.2 Quelles commandes le jeu accepte-t-il ?
Le jeu accepete les commandes suivantes :
go quit help

7.1.3 Que fait chaque commande ?
go : permet le joueur de se déplacer dans quatre direction *north,south,east,west* et entrée dans une chambre s'il existe.
help : pour le moment cette commande permet de localisé vagement l'endroit du joueur. quit : permet de quitter le jeu.

7.1.4 Combien de pièces y a-t-il dans le scénario ?
il existe 5 chambres : *outside, theatre, pub, lab, office*

7.1.5 Dessinez une carte des pièces existantes.
Plan du jeu inventé :


campus pub
Main entrance of the university
Theatre
computing lab
computing admin office

II.2 Exercice 7.2

Rôle des classes :
CommandWord : Cette classe permet tout simplement de tester si une chaine est equivalente au tableau de commande qu'elle contient à traves la méthode isCommand qui renvoie true dans le cas les chaines sont equivalentes sinon false.

Parser : Cette classe permet de récuperer la chaine à partir de la ligne de commande en utilisant la bibliothèque Scanner à traves la méthode getCommand On récuperer le premier mot et le deuxième et ignore le reste.

Command : Cette classe permet la vérification individuel des commandes saisie de la part de l'utilisateur décomposé en deux parties au par avant avec le parser.

Room :Cette classe permet la création des salles avec une petite description on constate aussi la méthode setExits qui permet de mettre des sorties de cette salle.

Game :C'est la classe main du jeu permet d'excuter le jeu en boucle tant que on quit pas le jeu toute en lisant les commandes entrées par l'utilisateur et faire les déplacements dans les salles selon les envies du joueur.

II.3 Exercice 7.3

Plan du jeu inventé :
Vous êtes le chef d'équipage d'un bateau de pirates, vous naviguez sur les mers afin de visiter différentes îles, se ravitailler en vivres ou équipements, combattre d'autres équipages et la marine afin de s'approprier et trouver les plus grands trésors dont le One Piece.

II.4 Exercice 7.4

Voir code.

II.5 Exercice 7.5

Voir code.

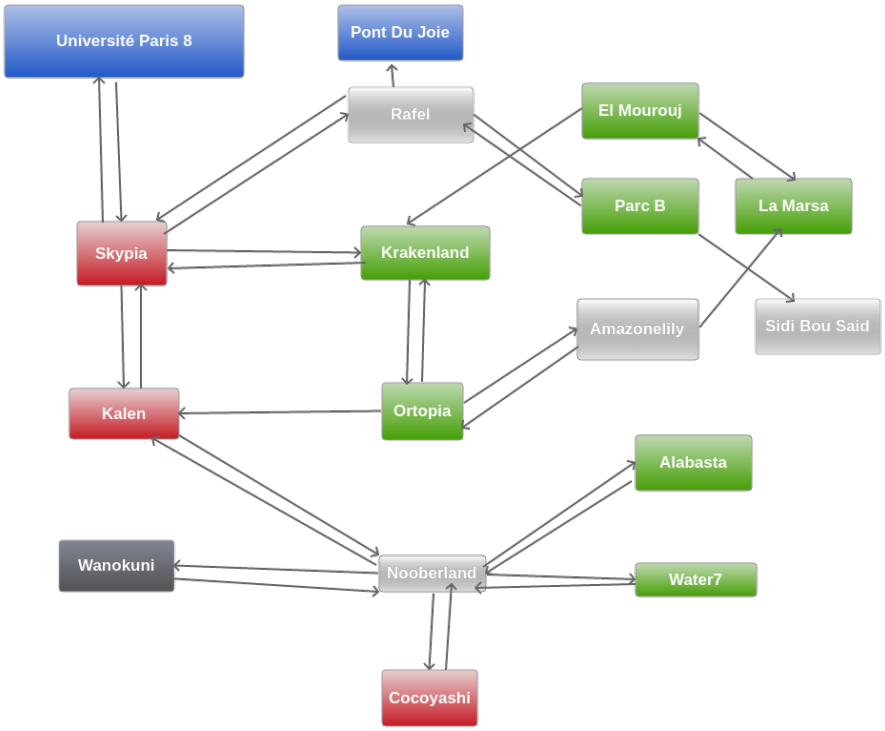


FIGURE 1 – carte du jeu

II.6 Exercice 7.6

Voir code.

II.7 Exercice 7.7

Voir code.

II.8 Exercice 7.8

Voir code.

II.9 Exercice 7.9

Recherchez la méthode keySet dans la documentation de HashMap. Qu'est ce que ça fait ?
La méthode keySet() est utilisée pour obtenir une vue d'un Map,Un Map est un ensemble de clé valeur,et là méthode keySet permet de voir nos clés.Les modifications apportées au Map sont donc reflétées dans l'ensemble, et inversement.

II.10 Exercice 7.10

Expliquez, en détail et par écrit, comment la méthode getExitString indiqué dans le code 7.7 fonctionne.

```
String returnString = "Exits:";
```

Cette ligne on déclare une chaine de caractère simple.

```
Set<String> keys = exits.keySet();
```

Ici on déclare notre Set on obtient toute les clés grâce a keySet() du HashMap exits

```
for(String exit : keys)
        returnString += " " + exit;
```

Ici on boucle sur ces clés et on ajoute nos clés a notre chaine de départ returnString

```
return returnString;
```

Renvoie la chaine avec les clés de exits .

II.11 Exercice 7.11

Voir code.

II.12 Exercice 7.12

Draw an object diagram with all objects in your game, the way they are just after starting the game

II.13 Exercice 7.13

How does the object diagram change when you execute a go command ?

II.14 Exercice 7.14

Voir code.

II.15 Exercice 7.15

Voir code.

II.16 Exercice 7.16

Voir code.

II.17 Exercice 7.17

Si vous ajoutez maintenant une nouvelle commande, avez-vous encore besoin de changer la classe de jeu ? Pourquoi ?
Oui ,dans la méthode processCommand qui permettera de tester si la commande saisie par l'utilisateur existe ou pas comme ceci :

```
if(commandWord.equals("newcommand"))
        dosomething();
```

par contre on est plus obligé de l'écrire dans printHelp puisqu'elle s'appelle automatiquement a partir du parser :

```
public void showCommands(){
        commands.getCommandList();
}
```

III Itération II

III.1 Exercice 7.18

Voir code.

III.2 Exercice 7.18.1

Comparer son projet à celui-ci (gestion des sorties et des descriptions du projet zuul-better.)
La version acctuel du projet est compatible avec zuul-better.Il y a aucun warning.

III.3 Exercice 7.18.2

StringBuilder Avant d'aborder Stringbuilder, remémorons nous qu'un objet de type string est non mutable, aux antipodes de ceci, StringBuffer est mutable, ce qui signifie que l'on peut changer la valeur de l'objet. Ainsi, Stringbuilder sera favorable

```
public StringBuilder getExitString(){
        StringBuilder returnStringBuilder=new StringBuilder("Exits: ");
        for(String exit : exits.keySet())
                returnStringBuilder.append(exit+" ");
        return returnStringBuilder;
}
```

Il existe aussi la méthode insert qui prend en paramètre la position et la chaine à inserer.
Aussi delete qui permet de supprimer en revanche il prend deux argements qui seront la position de départ et d'arrivé de la supression.

III.4 Exercice 7.18.3

Voir dossier images.

III.5 Exercice 7.18.4

Le jeu est intitulé One Piece Treasure Cruise

III.6 Exercice 7.18.5

Voir code.

III.7 Exercice 7.18.6

Voir code.

III.8 Exercice 7.18.7

addActionListener() :

AddActionListener ajoute l'écouteur d'action spécifique pour recevoir les évènements d'action de ce champ de texte. Elle prend en paramètre l'écouteur d'action à ajouter. Exemple : écouter la musique et chanter.(vouloir chanter quand on marche). écouter la musique est un évenement et chanter une action on peut ajouter la marche une action,Dans ce cas on a fait addAction() si en marche on chante aussi.

actionPerformed() :

ActionPerformed prend en paramètre une action, et fait appelle à la fonction processCommand afin de créer une interface pour le champ de texte de saisie, lire la commande et faire ce qui est nécessaire pour la traiter. Ici obligatoire, car Userinterface implémente l'interface ActionListener.

Résumons :

L'interface d'écoute pour recevoir des événements d'action. La classe intéressée par le traitement d'un événement d'action implémente cette interface et l'objet créé avec cette classe est enregistré avec un composant, à l'aide de la méthode addActionListener du composant.
Lorsque l'événement action se produit, la méthode actionPerformed de cet objet est appelée.

III.9 Exercice 7.18.8

voir code.

III.10 Exercice 7.19.1

voir code.

III.11 Exercice 7.19.2

voir code.

III.12 Exercice 7.20

voir code.

III.13 Exercice 7.21

voir code.
La classe Room c'est elle qui contient la fonction reponsable de l'affichage Tout simplement parce que c'est les items présent juste dans la chambre et que cette fonction fera appel à la description de chaque item .

III.14 Exercice 7.22

voir code.

III.15 Exercice 7.22.1

L'utilisation d'un HashMap composé des clés et d'items,le choix c'est porté sur un HashMap tout simplement pour la raison de cohérence avec les sorties de la chambre.

III.16 Exercice 7.22.2

voir code.

III.17 Exercice 7.23

voir code.

III.18 Exercice 7.24

La commade 'back' ignore le deuxième paramètre.

III.19 Exercice 7.25

Si on tape back deux fois ça nous renvoie deux fois en arrière.

III.20 Exercice 7.26

Oui la commande back marche mais elle nous indique qu'on est au début on peut plus reculer

III.21 Exercice 7.26.1

Documentation gènérer.

IV Itération III

IV.1 Exercice 7.27

Les tests unitaires et les tests de non-régression.

IV.2 Exercice 7.28.1

Créer une nouvelle commande test acceptant un second mot représentant un nom de fichier, et exécutant toutes les commandes lues dans ce fichier de texte Avant la création de la nouvelle commande, il nous faut au préalable rajouter dans la chaine de caractère « test » dans la liste des commandes valides de CommandWord. Ensuite nous avons créer la méthode test : Nous avons eu la charge de réaliser une nouvelle commande « test » pour effectuer des test de commandes pour notre jeu. En effet, lorsqu ?un code est changé, il est probable que des erreurs soient introduites dans celui-ci, il est donc important de procéder avec prudence et d ?établir une série de tests de notre programme. La méthode test prend en paramètre un fichier texte dans lequel sera inscrit à chaque ligne les différentes commandes de notre jeu. Pour réaliser cette fonction nous avons employé les méthodes hasNext, next et exceptions qui sont utiles dans la lecture des fichiers de texte. Tout d ?abord, l ?usage de la méthode hasNext nous permet de vérifier si notre fichier contient une ligne suivante, elle va retourner true si notre itération contient d ?autres éléments (lignes à parcourir), ici nous lirons notre fichier tant qu ?il contient des lignes à parcourir. Ensuite nous découvrons la méthode next qui va tout simplement récupérer la ligne suivante afin de l ?interpréter. Enfin pour vérifier que le fichier a bien été lu, nous faisons appel à exception, une exception est une erreur qui survient dans un programme, généralement elle produit l ?arrêt de celui-ci. Dans notre code nous pratiquons la capture d ?exception qui généralement sert à repérer un morceau de code qui pourrait générer une exception, pour notre part FileNotFoundException signalera qu ?une tentative d ?ouverture de fichier à échouer, cette exception sera levée par les constructeurs FileInputStream, FileOutputStream et RandomAccessFile lorsqu'un fichier avec le chemin spécifié n'existe pas et sera également lancée par eux si le fichier existe mais qu ?il est cependant inaccessible(Par exemple lors d ?une tentative d ?ouverture en lecture seule mais pour l ?écriture).

IV.3 Exercice 7.28.2

Deux fichiers créér mais vu que le sénario n'est encore claire parce que on a besoin des enemies pour ganger dans notre cas.

IV.4 Exercice 7.29

voir code.

IV.5 Exercice 7.30

voir code.

IV.6 Exercice 7.31

voir code.

IV.7 Exercice 7.32

voir code.

IV.8 Exercice 7.33

voir code.

IV.9 Exercice 7.34

voir code.

IV.10 Exercice 7.35

La modification des fichiers de tests acceuil bien le take,drop,eat,check

IV.11 Exercice 7.35.1

voir code.

IV.12 Exercice 7.35.2

voir code.

IV.13 Exercice 7.36

voir code.

IV.14 Exercice 7.37

Pour le moment on a décicder de laisser les mots clés jusqu'a l'établissement des enemies vu que notre but c'est la réalisation du jeu One Piece.

IV.15 Exercice 7.38

Oui ça marche puisque le type est enum donc on peut avoir même GO("avancer") par exemple

IV.16 Exercice 7.39

Pour nous les directions seront plus moins les mêmes donc pas besoin de changer en autre mots. Et tout simplement parce que c'est des direction du bateau.

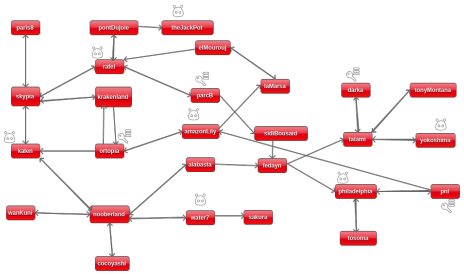
IV.17 Exercice 7.40

Oui, et bien sûr ne pas oublier de l'implémenté dans le switch de interpetCommand

IV.18 Exercice 7.41

Oui ça change automatiquement.

IV.19 Petite changement de carte



V Itération IV

V.1 Exercice 7.42 et 7.42.1

Ajout d ?une limite de temps au jeu qui cause la perte d ?un joueur si celui-ci n ?a pas terminé le but du jeu dans un temps imparti. Pour cela nous faisons appel à la classe LocalTime, ici LocalTime est un objet date-heure immuable qui représente un temps, souvent considéré comme heure-minute-seconde. Le temps est représenté avec une précision de l'ordre de la nanoseconde. Nous avons donc implementé la fonction play qui permet d'obtenir le temps réel, le joueur aura 20 minutes pour gagner sinon il mourra, l'équipage mourra aussi toutes les 1 minute. Ci-dessous se trouve son implémentation :

```
public void play(UserInterface userInterface) {
    setGUI(userInterface);
    printWelcome();
    int finish = 0;
    // Maybe giving some extra time
    LocalTime time = LocalTime.now();
    LocalTime localTime3 = time.plusMinutes(10);
    LocalTime breakfeastTime = time.plusMinutes(1);

    do {
        LocalTime currentTime = LocalTime.now();
       // gui.sound("src/music/Pirate.wav");
        if (currentTime.getMinute() == localTime3.getMinute()) {
            endGame();
            finish = 1;
        }
        if (currentTime.getMinute() == breakfeastTime.getMinute()) {
            player.setStrength(player.getStrength() - 10);
            gui.setStrength(player.getStrength());
            breakfeastTime = currentTime.plusMinutes(1);
        }

        if (player.getStrength() <= 0 || player.getLife() == 0) {
            endGame();
            finish = 1;
        }
        if (player.checkItemInTheBag("OnePiece") != null) {
            winGame();
            finish = 1;
        }
    } while (finish != 1);
}
```

V.2 Exercice 7.43

Dans cet exercice, nous avons créé une pièce franchissable dans qu'un seul sens, la solution de cet exercice consiste à créer une direction permettant d ?aller dans une room, mais de ne pas créer la direction inverse. Par exemple il serait possible de rentrer dans une salle avec go nord mais ne pas pouvoir faire go sud.

V.3 Exercice 7.44

Il nous a été demandé d ?ajouter un téléporteur qui doit pouvoir être ramassé dans une première pièce, puis pouvoir être chargée dans une deuxième pièce et enfin déclenché dans une 3e, il est réutilisable, mais doit être rechargement au préalable.

Puis, dans la classe GameEngine, nous avons ajouté une méthode charge() et une méthode fire(), si la première nous permet de nous souvenir de notre pièce donc de la charger, la deuxième elle nous permet d ?effectuer le téléportement.

```
    private void fire(){
        if(beamerCharged!=null){
            currentRoom=beamerCharged;
            gui.println("You was teleported to "+currentRoom.getDescription());
            gui.println("You Maybe Missed those "+currentRoom.getItemsDescription());
            if(currentRoom.getImageName()!=null){
                gui.showImage(currentRoom.getImageName());
            }
        }
        else{
            gui.println("charge your beamer before please");
        }
    }

private void charge() {
        if (player.checkItemInTheBag("beamer") != null) {
            if (player.checkItemInTheBag("ammo") != null) {
                player.removeItemFromBag("ammo");
                beamerCharged = currentRoom;
            } else {
                gui.println("You are OUT OF AMMO");
            }
        } else {
            gui.println("You must have a beamer first");
        }
    }
```

Puis dans enum CommandWords Nous rajoutons la valeur fire et charge

```
GO("go"), QUIT("quit"), HELP("help"), LOOK("look"), EAT("eat"), BA

TEST("test"), TAKE("take"), DROP("drop"), CHECK("check"),OPEN("oper

PAY("pay"),CHARGE("charge"),FIRE("fire"),TALK("talk"),GIVE("give")

ATTACK("attack"),HIRE("hire"),RECOVER("recover"),SAVE("save"),UNKNO
```

V.4 Exercice 7.45.1

La mise à jour les fichiers de test, notamment celui qui doit tester le jeu le plus exhaustivement possible.

Test des	fonctions
go north	take money
go east	go northWest
take beamer	go southWest
go west	take magicKey
go west	go west
give money	go southEast
go east	go northEast
go northWest	go east
take ammo	go northEast
take gold	go north
attack Dalton5	take magicKey
go north	hire
go north	go south
take cookie	take banana
go south	eat banana
go northEast	go northWest
attack Dalton6	go northWest
hire	pay
go southEast	go southWest
take magicKey	go north
go southEast	talk
take kiwi	give money
go south	open
take avocat	go east
eat avocat	attack Dalton7
go southEast	hire
attack Dalton4	hire
go east	attack Dalton7
take magicKey	take OnePiece

V.5 Apprentissage

La Classe Random Java permet de générer un flux de nombres pseudo-aléatoires. La classe utilise une graine de 48 bits, qui est modifiée à l'aide d'une formule congruentielle linéaire.

La méthode nextInt(int n) est utilisée pour obtenir une pseudo-aléatoire, uniformément distribuée entre 0 (inclus) et la valeur spécifiée (exclusive), tirée de la séquence de ce générateur de nombres aléatoires.

Le seedest la valeur initiale de l'état interne du générateur de numéro pseudo-aléatoire qui est maintenu par la méthode next(int). Donc est un point de départ, à partir duquel quelque chose grandit. Dans ce cas, une séquence de nombres. Ceci peut être utilisé soit pour générer toujours la même séquence (en utilisant une semence constante connue), ce qui est utile pour avoir un comportement déterministe. C'est bon pour le débogage, pour certaines applications réseau, la cryptographie, etc.

V.6 Exercice 7.46

Nous avons eu la charge d ?ajouter une salle de téléportation. Chaque fois que le joueur entre dans cette pièce, il est transporté au hasard dans l'une des autres pièces. Pour notre part, nous avons créé une classe TransporterRoom héritant de la classe Room, celle-ci contient un constructeur, une méthode findRandomRoom() retournant une pièce aléatoire en utilisant le Scénario donné dans le constructeur de cette classe retourne une pièce aléatoire en utilisant la méthode findRandomRoomRoom().

```java
package src;

public class TransporterRoom extends Room{


    /**
    * @param description String description of this room.
    * @param scenario The Scenario that's used in the Game class.
    */
    public TransporterRoom(String name,String description,String image){
        super(name,description,image);

    }

    /**
    * @return a random room using the findRandomRoom() method.
    */
    @Override
    public Room getExit(String direction){
        return findRandomRoom();
    }

    /**
    * @return a random room using the Scenario given in this class's constructor
    */
    public Room findRandomRoom(){
        Scenario a=new Scenario();
        return a.getRandomRoom();
    }

}
```

L'héritage a déjà été utilisé pour la classe Beamer.

Commentaires javadoc des classes mises à jour.

Javadocs progdoc et userdoc régénérée et mise à jour.

V.7 Exercice 7.47

voir code.

V.8 Exercice7.48

Ajout des personnages au jeu. Les personnages sont similaires aux objets, mais ils peuvent parler. Ils parlent un peu de texte quand nous les rencontrons pour la première fois, et ils peuvent nous donner de l'aide si nous leur donnons le bon article. La classe Character a été créée dans le jeu. Elle crée des personnages caractérisés par un nom, la room actuelle, une String faisant référence à son dialogue et l'item qu'il peut recevoir de l'utilisateur.

Création d'un package pkg characters :

V.9 Exercice 7.53

voir code.

V.10 Exercice 7.54

Le jeu peut être lancer de deux façons, la première avec la méthode main(),en y accédant avec cd chemin vers le jeu. La deuxième consiste à créer un fichier .jar dans blueJ avec pour classe principale Game, nous pouvons lancer le jeu par un simple double-clic ou dans le panneau de commande en plaçant le chamin vers le fichier.jar .

V.11 Exercice 7.58

Lancer le jeu avec la commande java ? jar et voir exo 7.54

V.12 Exercice 7.59

voir code.