

UNIVERSITÉ PARIS 13

Rapport Projet Zork (deuxième partie)

Nom : **BENSEKHRI**

Prénom : Mohand Lounis

N° étudiant : 11710457

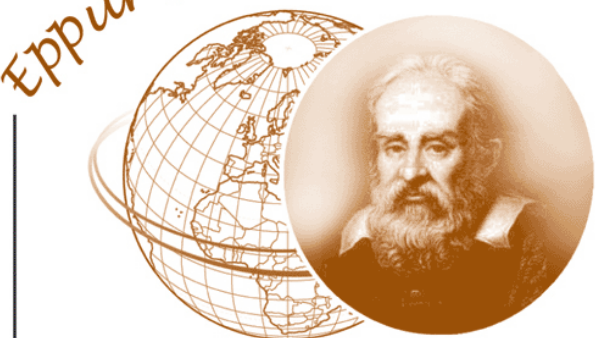
Année Universitaire : 2018/2019

Licence 2 Informatique

M. CHAMPESME

Mme. REKIK

Eppur si muove !



Institut Galilée

Table des matières

INTRODUCTION	3
LE JEU	4
Histoire	4
But	4
Plan des lieux	4
Commandes	5
Explications	5
Règles	5
Les objets	6
Les pièces et leur sorties	7
Cas gagnant !	7
Cas perdant !	8
LE CODE : classes et méthodes	9
ObjetZork	9
ArrayListConteneur	9
Piece	10
Joueur	11
Direction	12
Commande	12
MotCleCommade	12
AnalyseurSyntaxique	12
Jeu	13
zorkfr	13
PROBLEMES RENCONTRES	14

INTRODUCTION

Au cours de cette deuxième partie du projet de programmation Orientée Objet, j'ai créé un jeu d'aventure du Monde Zork qui est un jeu textuel codé en JAVA que j'ai nommé l'amour au bout du fil.

J'ai imaginé un scénario pour le jeu qui permet plusieurs fonctionnalités comme prendre un objet, poser un objet, tuer un monstre... etc.

J'ai choisi un scénario de type médiéval, avec un chevalier du nom de Léon qui sera le joueur et qui fera toutes les actions du jeu, qui a comme objectif de sauver sa princesse nommée Eva qui s'est fait enlever et emprisonner dans un donjon gardé par un Dragon.

Les outils et les monstres sont considérés comme des objets ; soit transportable ou non transportables.

J'ai rencontré quelques problèmes dont j'ai pu trouver les solutions et d'autres non j'en parlerai de cela un peu plus loin dans le rapport.

• Commandes

Les commandes reconnues pour jouer

Commandes de déplacements	Syntaxe	Description
Aller	Co. Direction	Se déplacer dans la une direction.
retour	Co.	Retourner à la pièce précédemment visitée.

Commandes d'actions	Syntaxe	Description
Prendre	Co. oz	Prendre l'objet oz et le mettre dans le sac.
Poser	Co. oz	Poser l'objet oz et le mettre dans la pièce.
Tuer	Co. monstre arme	Tuer le monstre avec une arme
Piece	Co.	Afficher le nom de la salle les objets qui s'y trouve et ses sorties.
Sac	Co.	Afficher le nom du joueur, le poids qu'il peut transporter et les objets qu'il porte.

Commandes d'aide & système	Syntaxe	Description
ouSuisJe	Co.	Afficher le nom de la pièce ou le joueur se trouve et ses sorties.
Aide	Co.	Afficher l'aide du jeu.
Plan	Co.	Afficher le plan du jeu.
Histoire	Co.	Afficher l'histoire du jeu.
Quitter	Co.	Quitter le jeu.

• Explications

Le jeu commencera dans Le Château : Léon, le chevalier est celui qui effectuera toutes les actions du jeu qui sont soi ; prendre, poser ou tuer un objet (ou un monstre dans le cas de tuer) grâce aux commandes ci-dessus, Léon peut transporter 120 kg dans son sac au début du jeu. Il y a en tout six (06) pièces dans le jeu qui sont : le château, la grotte, la foret perdu, la maison abandonner, le nid du dragon et le donjon. On trouve dans ces pièces plusieurs objets de différentes spécifications ; il y a des objets qui sont transportable, d'autre non. Il y a des objets qui servent juste de décorations dans le jeu et qui ne servent pas vraiment au jeu on peut les distinguer en remarquant qu'ils n'ont pas de poids quand on affiche une pièce grâce à la commande « piece » et enfin y a des objets non transportables mais avec des poids et qui sont les monstre (j'ai crée des objets non transportable avec des poids pour que le poids soit le rapport de force dans le jeu).

• Règles

1. Pour tuer un monstre avec un poids x il faut prendre un objet de poids y avec la condition que x soit inférieur ou égale à y sinon on perd.
2. On peut utiliser qu'une seule fois un même objet pour tuer un monstre.
3. Pour prendre un objet il faut que ce dernier soit dans la même salle que le joueur, qu'il ait un poids inférieur ou égale au poids transportable restant pour le joueur qu'on peut consulter grâce à la commande sac, et il faut qu'il soit transportable (pas d'objet de décoration, ni de monstre).
4. Pour tuer le Gollum qui est dans la grotte il faut pratiquer la condition 1 et répondre à une énigme et qui est :

« Qu'est-ce qui a des racines que personne ne voit,
 Qui est plus grande que les arbres,
 Qui monte, qui monte, Et pourtant ne pousse jamais ? »

La réponse est : « Une montagne »

5. Si on essaye de tuer un objet qui autre qu'un monstre un message d'erreur nous indique que ce n'est pas un monstre, si on essaye de tuer un monstre avec une arme qui est plus légère que le monstre on perd.
6. Pour prendre le sorcier il faut le payer et cela on prenant l'argent qui est dans le château.
7. Si on pose le sorcier on ne peut plus le reprendre car on a déjà utilisé tout l'argent du château et cela dit qu'on perd car nous avons trois monstres à tuer et trois armes pour le faire qui sont l'Epee, le Sorcier et la Lance.
8. On remarque que la lance est beaucoup trop lourde pour qu'elle soit transporté par Léan au début du jeu (on applique la remarque 3) c'est pour cela qu'il y a un objet spécial lorsqu'on le prend le poids maximal que Léon peut transporter se multiplie par 10 ce qui fait que ça devient 1200 kg ce qui lui permettra de prendre la lance qui est la seule arme qui peut on revenir au bout du dragon de poids de 1100 kg, l'objet spécial est le Fruit_Magique que l'on trouve dans la foret perdu. On remarquera que lorsqu'on le prend il disparaît du sac et permet la modification on peut consulter le sac de Léon et on ne trouvera pas l'objet Le Fruit_Magique c'est ce que j'ai appelé manger le fruit magique dans le jeu.
9. Quand on réussit à tuer le dragon on ne pourra pas aller vers le nord (le donjon) si on a pas la clé qui ouvre la porte, cette clé se trouve dans la foret perdue.
10. On ne peut pas aller plus loin qu'une pièce là ou y a un monstre si on ne le tue pas d'abord.
11. Une dernière petite règle qui est le piège du jeu ; pour prendre la princesse Eva il faut obligatoirement que le cheval soit dans le sac sinon on perd, si on essaye de poser le cheval pendant que la princesse Eva est dans le sac on perd.



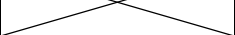







• Les objets

Nom	Poids (kg)	Transportabilité	Lieu	Description
La_princesse_Eva	45	True	Le Donjon	A sauver

Dragon	1100	False	Le Nid Du Dragon	Monstre à tuer avec la lance
Sorcier	100	False	La Maison Abandonnée	Monstre à tuer avec le sorcier
Gollum	63	False	La grotte	Monstre à tuer avec l'Epee

Epee	65	True	Le Château	Arme
Sorcier	110	True	La grotte	Arme
Lance	1200	True	Le Château	Arme

Argent	1	True	Le Château	Outil (prendre le sorcier)
Fruit_Magique	2	True	La Forêt Perdu	Outil (prendre la lance)
cle	1	True	La Forêt Perdu	Outil (ouvrir la

				porte du donjon)
Cheval	7	True	Le Château	Outil (prendre la princesse)
Chaines		False		Décoration
Œufs		False	Le Nid Du Dragon	Décoration
Flammes		False	Le Nid Du Dragon	Décoration
Chaudron		False	La Maison Abandonnée	Décoration
Patte_de_Poulet		False	La Maison Abandonnée	Décoration
Restes_de_Nourriture		False	La Grotte	Décoration
Eau		False	La Grotte	Décoration
Lit		False	Le Donjon	Décoration
Fenêtre		False	Le Donjon	Décoration
Miroir		False	Le Donjon	Décoration

• Les pièces et leur sorties

Nom	Sortie
Le Château	NORD / OUEST
La Grotte	OUEST / EST
La Forêt Perdu	EST
La Maison Abandonnée	NORD / SUD
Le Nid Du Dragon	NORD / SUD
Le Donjon	SUD

• Cas gagnant !

Pour gagner on a dis qu'il fallait ramener la princesse Eva au château depuis le donjon et pour cela il faut suivre les étapes suivantes. (On commence dans le château)

- ♦ On prend l'Epee et l'argent.
- ♦ On part vers l'OUEST on tue le Gollum avec l'Epee et en répondant à l'énigme.
- ♦ On prend le sorcier (car nous avons de quoi le payer dans notre sac).
- ♦ On part vers l'OUEST.
- ♦ On est dans la forêt perdue ; on prend la clé et le fruit magique (on remarque que le fruit magique n'est plus dans le sac et que le poids maximal transportable par Léon est désormais de 1200 kg.
- ♦ On revient au château en allant deux fois vers l'EST.
- ♦ On part vers le NORD.
- ♦ On tue la sorcière avec le sorcier.
- ♦ On revient au château (on peut utiliser la commande retour), on pose la clé et on prend la lance.
- ♦ On part vers le NORD deux fois.
- ♦ On est dans le nid du dragon.
- ♦ On tue le dragon avec la lance.
- ♦ On revient au château on prend la clé et le cheval.
- ♦ On part vers le NORD trois fois.

- ♦ On prend la princesse Eva.
- ♦ On revient au château et on pose la princesse, et c'est GAGNER !

• **Cas perdant !**

Il y a plusieurs cas qui nous font perdre qui sont :

1. Essayer de tuer un monstre avec un objet plus léger que le monstre.
2. Prendre le sorcier en ayant l'argent dans le sac après le poser (on peut plus le reprendre car on n'a pas d'argent et il y a trois monstre à battre et on vient de perdre la troisième arme nécessaire pour cela).
3. Prendre la princesse sans cheval dans le sac ou prendre la princesse avec le cheval dans le sac après poser le cheval.

Voici les étapes à suivre pour le premier (01) cas : (on est dans le château)

- ♦ On prend l'Epee de poids 65 kg.
- ♦ On va vers le NORD on essaye de tuer la Sorcière de poids 100 kg, et c'est PERDU.

Voici les étapes à suivre pour le troisième (02) cas :

- ♦ On prend l'argent.
- ♦ On va vers l'OUEST.
- ♦ On prend le Sorcier.
- ♦ On pose le Sorcier, et c'est PERDU.

Pour le troisième cas qui est constitué de deux cas :

Quand on aura réussi à tuer le dragon comme expliquer dans le cas GAGNANT.

- On ne prend pas le cheval et on essaye de prendre la princesse, et c'est PERDU.
- Ou bien on prend le cheval et la princesse et on pose le cheval et c'est PERDU.

LE CODE : classes et méthodes

Le jeu comporte dix (10) classes qui sont complémentaire l'une pour l'autre est qui sont : ObjetZork, ArrayListConteneur, Piece, Joueur, Direction, Commande, MotCleCommande, AnalyseurSyntaxique, Jeu, zorkfr.

Pour compiler le programme en tape la commande : « javac *.java » dans le terminal.

Pour exécuter le programme en tape la commande : « java zorkfr » dans le terminal.

✳ **ObjetZork :**

Un objetZork est constitué de trois instances qui sont : la description : le nom de l'objet, le poids : qui désigne son poids, et un boolean transportable : qui est nous permet de savoir si ce dernier est transportable ou non.

Cette classe est l'une des classe les plus importantes du jeu en effet c'est celle qui nous permettra de créer un Objet dans le jeu que nous poserons dans une pièce et qui peut être transportable par le joueur. Cette classe nous permet d'évaluer si il est transportable ou non, lui donner un poids si on le souhaite (on peut créer des objets sans poids) et lui donner une description. Elle permet notamment de récupérer les instances d'un objet et de comparer deux objets.

- Constructeurs :

J'ai créé deux constructeurs :

Le premier nous permet de créer un ObjetZork seulement avec une description et c'est celui que j'ai choisi pour créer les décorations du jeu.

Le deuxième constructeur nous permet de créer un ObjetZork avec une description, un poids et un boolean de transportabilité, j'ai créé ce constructeur pour permettre de créer des Objet (des monstres) non transportables avec de différents poids qui sera une manière d'évaluer la force d'un monstre et aussi pour créer les armes et les différents outils du jeu.

- Guetteurs :

J'ai créé plusieurs guetteurs qui nous permettent à chaque fois soit de renvoyer :

La description courte d'un Objet c'est-à-dire juste sa description.

La description longue d'un Objet est celle-ci est un peu différente de la précédente car en effet elle nous donne la totalité des informations sur un objet quelconque ; sa description, son poids et sa transportabilité. C'est le guetteur que j'ai utilisé pour afficher les objets d'une pièce grâce à la commande « piece ».

Le poids de l'objet.

La transportabilité de l'objet.

- Méthodes :

J'ai créé plusieurs méthodes qui sont :

La méthode « equals » cette méthode nous permet de comparer deux ObjetZork.

La méthode « equalsDescription » qui permet de savoir si deux ObjetZork ont la même description.

De plus que ces deux méthodes j'ai ajouté la méthode « clone » qui clone un ObjetZork et les deux méthodes « hashCode » et « toString ».

✳ **ArrayListConteneur :**

La classe ArrayListConteneur est une super classe qui est une classe abstraite, cette classe sera hériter par les deux classe filles ; Piece et Joueur.

La classe ArrayListConteneur est créée dans le but de pouvoir stocker les ObjetZork et puisque les deux classes ; Piece et Joueur ont besoin de cette notion de stockage d'ObjetZork j'ai préféré implémenter cette classe qui aura les différentes méthodes qui permettront la manipulation de ces ObjetsZork qui seront accessibles par les deux classes filles ; Piece et Joueur.

J'ai choisi de mettre la classe ArrayListConteneur en classe abstraite pour le fait que nous aurons pas intérêt de créer une instance nommée ArrayListConteneur.

- Constructeurs :

La classe ArrayListConteneur aura un seul constructeur qui sera appelé implicitement, c'est-à-dire qu'on n'aura pas à créer un new ArrayListConteneur car on n'a pas le droit du fait que c'est une classe abstraite mais nous aurons besoin de cet appel de constructeur qui sera implicite pour les constructeurs des deux classes filles ; Piece et Joueur et cela grâce à la méthode super() ; Le constructeur de ArrayListConteneur permet de créer un ArrayList d'ObjetZork.

- Getteurs

Il y a plusieurs getteurs qui nous permettent de renvoyer :

Le nombre d'objets qui sont dans le conteneur d'ObjetZork.

Le ArrayList des ObjetZork (problème de clonage dont je parlerai dans la partie problèmes rencontrés).

« Ces getteurs seront accessibles par les classes filles et cela grâce à super. »

- Méthodes

J'ai implémenté plusieurs méthodes qui soit nous permettront de savoir le nombre d'objets spécifiés sont contenus dans le conteneur, soit de savoir si cet ObjetZork est contenu ou non dans le conteneur, soit d'ajouter ou retirer un ObjetZork et enfin de renvoyer la première occurrence de l'objet qui aura une certaine description spécifiée.

« Ces getteurs seront accessibles par les classes filles et cela grâce à super. »

« J'ai importé la bibliothèque ArrayList pour avoir accès aux méthodes qui manipulent les listes ».

* **Piece :**

Une pièce est constituée d'une description : qui est le nom de la pièce, des sorties : qui nous permet de relier les pièces entre elles et de se déplacer de l'une à l'autre grâce aux directions et à la commande « aller » et d'une liste d'objets : qui sont les objets que chaque pièce dispose celle-ci peut être vide si la pièce est vide.

La classe Piece hérite de la classe ArrayListConteneur et cela pour pouvoir stocker les ObjetZork qu'elle contient et aussi accéder à toutes les méthodes implémentées dans la super classe.

Cette classe est très importante car c'est celle qui nous permet de créer une pièce avec une description et dans laquelle le joueur pourra prendre ou poser des objets et se déplacer vers les différentes sorties de cette dernière. Elle nous permet aussi de récupérer les instances d'une pièce et d'afficher les différents objets qui s'y trouvent.

- Constructeurs :

Il y a un seul constructeur pour cette classe ; il nous permet de créer une pièce grâce à une description sa liste d'objets et créé grâce à l'appel implicite du constructeur de la super classe ArrayListConteneur grâce à la méthode super() ; et enfin ses sorties sont initialisé à null.

- Guetteurs :

Il y a plusieurs guetteurs qui nous permettent de renvoyer :

La description courte d'une pièce c'est-à-dire juste sa description

La description longue d'une pièce celle-ci donne plus d'informations sur la pièce en effet elle renvoie le nom de la pièce et ses sorties.

La description Sorties qui nous permet de connaître toutes les sorties possibles d'une pièce.

La liste des Objets d'une salle

La pièce suivante en prenant une direction possible à partir de la pièce courante.

La classe Piece hérite de tous les guetteurs implémenté dans la classe ArrayListConteneur.

- Méthodes :

J'ai créé plusieurs méthodes qui soit nous permettrons de définir les sorties d'une pièce (setSorties), et afficher tous les objet que contient une salle (afficherPiece) et c'est la méthode que j'ai utiliser pour la commande « piece ».

* **Joueur** :

Un joueur est constitué d'un nom, d'une pièce qui est la pièce ou le joueur est présent, un poidsMax, qui est le poids maximal qu'un joueur peut transporter, et enfin d'une liste d'objets que le joueur transporte.

La classe Joueur hérite de la classe ArrayListConteneur et cela pour pouvoir stocké les ObjetZork qu'il contient et aussi accéder à toutes les méthodes implémenté dans la super classe.

La classe Joueur est aussi l'une des classe les plus importante du jeu en effet elle nous permet de calculer le poids transporter par le joueur ce qui nous permet de savoir si il peut en prendre davantage, changer la pièce ou se trouve le joueur, et afficher tous ce qu'il y a à savoir sur ce joueur que ce soit son nom, la pièce ou il se trouve, le poids qu'il peut transporter, le poids qu'il lui reste à transporter et enfin les différents objets qu'il a transporte.

- Constructeurs :

Il y a un seul constructeur qui nous permet de créé un joueur grâce à un nom, une pièce et un poids maximal qu'il peut transporter. Et aussi la liste où il pourra stocker les ObjetZork qu'il transportera et cela grâce au constructeur de la super classe.

- Guetteurs :

J'ai créé plusieurs guetteurs qui nous permettront de renvoyer qu'on le souhaite soit :

Le nom du joueur, la pièce où se trouve le joueur, le poids maximal que le joueur peut transporter, la description longue du joueur c'est-à-dire son nom, le poids maximal qu'il peut transporter et le poids qu'il lui reste a transporter.

Tous les guetteurs implémentés dans la super classe ArrayListConteneur sont accessible par la classe Joueur.

- Setteurs :

J'ai créé deux setteurs qui sont setPoidsMax qui nous permettra de changer la valeur du poids maximal qu'un joueur peut transporter, et un autre setteur qui est setPiece qui permet de changer la pièce ou se trouve le joueur.

- Méthodes :

J'ai implémenté plusieurs méthodes qui dans quelques cas servent juste pour faciliter l'implémentation d'autre méthodes comme : poidsTrans (aide pour la méthode poidsRestantATrans), et d'autres qui sont

là pour faire un travail précis comme `poidsRestantATrans` qui calcule le poids qu'un joueur peut encore transporter, `afficherLesObjets` qui affiche les objets que le joueur transporte, j'ai implémenté notamment deux autres méthodes qui nous permet de savoir si un ajout est possible soit avec seulement le poids restant soit avec d'autres facteurs, et enfin j'ai redéfini les deux méthodes `ajouter` et `retirer` de la super classe et cela en mettant `@Override` pour indiquer que c'est une redéfinition et cela est dû au fait que dans la classe `Joueur` entre en jeu le facteur poids qui change après l'ajout et la retraitation.

✱ **Direction :**

A propos de la classe `Direction` je n'ai rien changé.

C'est une classe qui permet d'initialiser les différentes directions que l'on peut prendre à partir d'une pièce ; `NORD`, `EST`, `SUD`, et `OUEST`.

✱ **Commande :**

Dans cette classe j'ai fais plusieurs modifications.

Cette classe est celle qui répertorie les informations liée à une commande entrée par l'utilisateur.

Une commande est constituer au plus de trois(03) mots qui sont le mot de commande, le second mot qui est le nom de l'objet à prendre ou a poser ou bien le nom de l'objet à tuer, le troisième mot qui sert juste pour la commande tuer qui est le nom de l'objet qui sert d'arme.

- Constructeurs :

J'ai modifié le constructeur pour qu'il prenne trois (03) mots au lieu de seulement deux (02).

- Guetteurs :

J'ai créé plusieurs guetteurs qui nous permettrons de récupérer les différents mots entrés par l'utilisateur ; le mot de commande, le second mot et le troisième mot.

- Méthodes :

A propos des méthodes de cette classe ; se sont que des testeurs qui servent pour dire qu'une ligne de commande entré par l'utilisateur et valide ou pas ; (si elle contient un mot clé reconnue par le joueur , si elle a un second mot... etc.)

✱ **MotCleCommade :**

J'ai apporté quelques modifications à cette classe.

Pour les commandes valide j'ai rajouté la commande : `retour`, `prendre`, `poser`, `tuer`, `sac`, `piece`, `ouSuisje`, `aide`, `plan`, `histoire` en plus des deux commandes `aller` et `quitter` qui y sont déjà.

La description et la syntaxe des commandes ci-dessus sont déjà présenté dans la partie `Jeu`.

J'ai notamment changé l'implémentation de la méthode « `afficherToutesLesCommandes` » et cela est juste pour avoir un meilleur affichage des commandes pour une meilleure compréhension de la part du joueur.

✱ **AnalyseurSyntaxique :**

A propos de cette classe j'ai apporté seulement les modifications qui permettront a l'utilisateur d'entrer trois (03) mots dans la ligne de commande au lieu de seulement deux (02) et bien-sûr en gardant le fait que tout ce qui suit les mots de commandes dont on a besoin est ignorer.

C'est-à-dire que si je tape la commande : prendre Epee efeih

L'analyseur Syntaxique prend en compte que le premier et le deuxième car la commande prendre a besoin que d'un seul complément.

* **Jeu :**

A propos de cette classe j'ai apporté plusieurs modifications.

La classe Jeu est la classe principale du jeu en effet c'est celle qui comporte la boucle du jeu la méthode jouer et qui permet aussi de comprendre ce qu'il faut faire quand on tape une ligne de commande.

Un jeu est constitué d'un analyseurSyntaxique qui lit les entrées sur le terminal, d'une pièce courante celle où le joueur se trouve, d'une pièce précédente qui est la pièce qu'un joueur a précédemment visitée et d'un joueur.

- Constructeurs :

Cette classe a un seul constructeur qui permet d'initialiser les différentes pièces du jeu grâce à la méthode « creerPieces » et créer un analyseur syntaxique.

- Guetteurs :

J'ai créé plusieurs guetteurs qui permettront soit de renvoyer la pièce courante où est le joueur, la pièce précédente, le joueur.

- Méthodes :

A propos des méthodes cette classe en possède plusieurs modèles on trouve des méthodes qui servent d'affichage comme le menu (message de bienvenue), l'aide, l'histoire et le plan. Une méthode qui sert pour la création du jeu qui est « creerPieces » qui sert à créer les différentes pièces du jeu indiquer leur sorties, créer le joueur et lui indiquer l'endroit où il débutera le jeu, et enfin créer les différents objets du jeu (armes, monstre, princesse, outils, et décorations). Une méthode qui sert de boucle pour le jeu qui est la fonction « jouer » qui tourne tant que le boolean termine donné par la méthode « traiterCommande » est de valeur false et enfin les méthodes d'action qui sont les commandes du jeu comme : « deplacerVersAutrePiece » qui sert à déplacer le joueur d'une pièce à une autre et sauvegarder la pièce qu'il vient de quitter dans l'instance pièce précédente au cas où le joueur veut revenir en arrière, « prendre » qui permet au joueur la prise d'un ObjetZork cette Objet doit être présent dans la pièce courante, transportable, et enfin que le poids de l'ensemble des Objet portés dans le sac du joueur additionné à celui de l'objet doit être inférieur au poids maximum transportable par le joueur, la méthode « poser » qui permet le dépôt d'un ObjetZork dans la pièce courante, et pour cela il faut que l'objet à déposer soit présent dans le sac du joueur, et enfin la méthode qui implémente la commande « tuer » qui sert à tuer un Monstre du jeu avec une arme du jeu qui tous deux des ObjetZork.

« J'ai importé la bibliothèque Scanner qui permet à la lecture des entrées sur le clavier que j'ai utilisé pour tuer le Gollum car il faut répondre à l'énigme posée ».

* **zorkfr :**

Je n'ai apporté aucune modification à cette classe.

La classe zorkfr est celle qui comporte la méthode main qui permet de faire tourner le jeu (l'exécution du jeu) après compilation.

PROBLEMES RENCONTRES

A propos des problème rencontrés ; j'en ai eu beaucoup, j'ai réussi à corriger une grande partie, néanmoins il me reste un problème dont je ne comprends pas la raison ; le problème est que dans la méthode « getLesObjets » dans la classe ArrayListConteneur j'ai essayé de cloner la liste d'ObjetZork contenu dans le ArrayList, donc j'ai donné une définition que vous trouverez en commentaire dans le code source de la méthode, mais le problème c'est que avec cette définition que j'ai donné pour la méthode « getLesObjets », ne fonctionne pas, ce qui m'a poussé à renvoyé l'attribut directement sans clonage.