Léo DECHAUMET

# Le jeu « Zork »

# Projet :

# Programmation orientée objet

# Chapitre 1

# Introduction.

Le projet « Zork » avait pour but de nous initier à un nouveau paradigme de programmation : l'orientée objet. Afin de nous donner un point de départ, nous avions à notre disposition un programme de base, le jeu Zork. Dans ce jeu nous pouvions naviguer dans certaines pièces de l'Université Paris 13. Nous devions alors ajouter de nouvelle fonctionnalités, comme l'ajout d'objets dans certaines pièces, l'ajout de nouvelles pièces, de nouvelles commandes… pour rendre ce jeu plus intéressant. Ces améliorations devaient être réalisées selon la philosophie de l'orienté objet.

Dans la suite de ce document je vous présenterai en détail le jeu Zork, puis je vous décrirai les modifications que j'ai apportées au logiciel de base.

# Chapitre 2

# Présentation du jeu

## 2.1 Présentation générale

Le jeu « Zork » est un jeu dans lequel un joueur peut se déplacer dans des pièces de l'université Paris 13, récupérer des objets et interagir avec certains d'entre eux.
Il est développé en Java et utilise les principes de l'orienté objet.

## 2.2 But du jeu

Le but du jeu est de découvrir l'utilité d'un mystérieux objet. Cet objet est en réalité un portail, qui mène vers une planète inconnue nommé « Zork ». Pour gagner la partie, il faut trouver une clé, grâce à laquelle le joueur pourra activer le portail et remporter la partie.

## 2.3 Liste des commandes

Dans cette section seront présentées sous forme de liste toutes les commandes implémentées par le jeu Zork

- **Aller**

  Syntaxe

  ```
  aller DIRECTION
  ```

Description

Permet de se déplacer dans la pièce située dans la direction indiquée par le paramètre. Le paramètre `direction` peut valoir : `nord`, `sud`, `est`, `ouest`. Si aucune pièce n'est présente dans la direction donnée, la commande affiche un message d'erreur et le joueur ne change pas de pièce. Si la pièce courante possède une sortie dans la direction donnée, le joueur se déplace dans cette pièce (la pièce courante devient cette pièce).

- **Retour**

Syntaxe

```
retour
```

Description

Retourne dans la pièce précédente. Si la commande est exécutée après que le joueur a au moins une fois exécuté la commande `aller`, alors le joueur retourne dans la pièce précédente. Si le joueur n'a jamais exécuté la commande aller, le joueur ne change pas de pièce. Le jeu ne stocke pas d'historique des pièces traversées : si la commande est exécutée plusieurs fois de manière consécutive, alors le joueur naviguera entre deux pièces.
Exemple : Le joueur se situe dans une pièce A, laquelle possède une sortie vers une pièce B au nord. Voici comment la variable `pieceCourante` évolue après l'exécution des commandes :

```
aller nord      pieceCourante = B
retour          pieceCourante = A
retour          pieceCourante = B
retour          pieceCourante = A
```

- **Prendre**

Syntaxe

```
prendre OBJET
```

Description

Récupère l'objet passé en paramètre et le place dans l'inventaire du joueur. Le paramètre doit être le nom de l'objet à prendre. La commande retire l'objet correspondant au nom passé en paramètre de la pièce courante et place cet objet dans le sac du joueur sauf dans les cas suivants :
- Aucun objet de ce nom ne se situe dans la pièce.

- L'ajout de cet objet entraînerait un dépassement de la capacité de l'inventaire du joueur
- L'ajout de cet objet entraînerait un dépassement du poids maximal que peut transporter le joueur.
- L'objet n'est pas transportable.

Quand un de ces cas est rencontré, la commande affiche un message d'erreur.

- **Poser**

Syntaxe

```
poser OBJET
```

Description

Pose l'objet passé en paramètre et le place dans la pièce courante. Le paramètre doit être le nom de l'objet à poser. La commande retire l'objet correspondant au nom passé en paramètre de l'inventaire du joueur et place cet objet dans le sac du joueur, sauf si le nom passé en paramètre ne correspond à aucun objet dans le sac du joueur. Dans ce cas la commande affiche un message pour signaler l'erreur.

- **Examiner**

Syntaxe

```
examiner [OBJET]
```

Description

Affiche la description de l'objet correspondant au nom passé en paramètre de la commande. L'objet doit se trouver dans l'inventaire du joueur, ou dans la pièce dans laquelle se situe le joueur. Si aucun objet ne correspond au nom passé en paramètre, la commande affiche un message d'erreur.
Cas particulier : `examiner commande` affiche une courte description de ce que fait chaque commande.
Si la commande ne possède pas de paramètre, affiche l'état du jeu, à savoir :
- La description de la pièce où se trouve le joueur
- La liste des objets présents dans la pièce
- La liste des sorties que possède la pièce, avec leurs directions (nord, etc) et les descriptions des pièces adjacentes
- La description du sac, avec le nombre d'objets transportés ainsi que la liste de ces objets.

- **Utiliser**

Syntaxe

```
utiliser OBJET
```

Description

Réalise l'action définie dans l'objet passé en paramètre. Le paramètre doit être le nom de l'objet à utiliser. L'objet doit se situer soit dans l'inventaire du joueur, soit dans la pièce courante. Les actions des objets peuvent être une des actions de cette liste :

- Afficher un texte défini dans l'objet
- Demander un code sous forme de texte à l'utilisateur, si ce code est correct ajoute un objet à la pièce courante
- Ajoute un objet dans la pièce courante si et seulement si l'utilisateur possède la clé reliée à cet objet
- Aucune action : l'objet peut ne pas posséder d'action, dans ce cas le message « Cet objet ne possède pas d'action spécifique » est affiché.

L'objet peut être détruit après appel de la commande si il est défini ainsi.
Il n'existe pas de moyen formel de connaître l'action d'un objet avant l'appel de la commande sur cet objet.
Si aucun objet ne correspond au nom passé en paramètre, la commande affiche un message pour le signaler.

- **Aide**

Syntaxe

```
aide
```

Description

Affiche la liste des commandes utilisables dans le jeu.

- **Quitter**

Syntaxe

```
quitter
```

Description

quitte le jeu normalement. La partie n'est pas enregistrée.

## 2.4 Plan du jeu

| | Objet | Sortie |
|---|---|---|
| Dehors | Ciseaux<br>Styleau | Nord : Taverne<br>Est : SalleTD<br>Sud : batimentC |
| SalleTD | | Nord : bibliotheque<br>Ouest : dehors |
| Taverne | coffre | Sud : dehors |
| BatimentC | note | Nord : dehors<br>Est : bureau<br>Sud : caverne |
| Bureau | | Ouest : batimentC |
| Bibliothèque | Livre<br>etagere | Sud : salleTD |
| Caverne | portail | Nord : batimentC |

## 2.5 Scénarios de test

### Situation gagnante

Une seule situation est considérée comme gagnante, le cas ou le joueur trouve la clé permettant d'actionner le portail, et actionne le portail. Cette situation est amenée par les commandes :

```
aller sud
```

[OPTIONNEL]

Vous pouvez aller examiner ce mystérieux objet en vous rendant dans la caverne, avec les commandes :

```
aller sud
examiner portail
aller nord
```

```
prendre note
```

```
utiliser note
```
(affiche le code pour le coffre : 80526)

```
aller nord
```

```
aller nord
```

```
utiliser coffre
```
Cette commande demande un code pour le coffre, il faut entrer le code marqué sur la note (80526)

```
prendre cle
```

```
aller sud
```

```
aller sud
```

```
aller sud
```

```
utiliser portail
```
Le portail peut être utilisé car vous possédez la clé le permettant.

Situation perdante

# Chapitre 3

# Modifications apportées au programme initial

## 3.1 Les nouvelles classes

Dans cette section nous présenterons les classes ajoutées au logiciel de base.

- **ObjetZork**

  Classe créée pour pouvoir créer les objets du jeu Zork. Ils permettent de rendre le jeu plus intéressant. Les objets sont situés initialement dans des pièces, ils peuvent être pris puis posés s'ils sont transportables.

  Les instances de cette classe sont immuables, on ne peut pas les modifier après initialisation. Cette caractéristique permet d'éviter que le hashCode de l'instance ne change, cela permet notamment d'éviter les erreurs liées à la collection Map « `CorrespondanceCle` » utilisée dans la classe jeu (Des instances de objetZork sont utilisées comme clé). Les classes filles de cet objet sont créées pour redéfinir les méthodes `actionne.` Elles doivent aussi être immuables

- **ObjetZorkCle**

  Classe qui hérite de ObjetZork. Classe créée pour pouvoir gérer des objets contenant un second objet. De tels objets peuvent être des coffres, que le joueur peut ouvrir à l'aide d'une clé et qui, une fois ouverts, ajoutent un nouvel objet dans la pièce. Ces objets peuvent aussi être des objets que le joueur peut activer à l'aide d'une clé. En effet, le portail qui permet de finir le jeu est un objet de type ObjetZorkCle.
  Lorsqu'il est activé, le portail est supprimé et le portail fonctionnel est débloqué, ce qui permet de gagner. Cet objet peut être débloqué si l'utilisateur possède la clé associée.
  Les correspondances Clé-objet sont définies dans une collection située dans la classe jeu.

- **ObjetZorkNote**

Classe qui hérite de ObjetZork. Classe créée pour pouvoir gérer des objets contenant un texte à lire quand on l'appelle avec la commande `utiliser`. Dans le jeu, une instance de cette classe contient le code permettant de déverrouiller le coffre contenant la clé.

- **ObjetZorkCode**

Classe qui hérite de ObjetZork. Comme les instances de la classe ObjetZorkCle, les instances de ObjetZorkCode possèdent un objet caché. Ici, pour récupérer cet objet il faut entrer le bon code. Dans le jeu, une instance de cette classe est utilisée pour stocker la clé du portail.

On ne peut pas mettre à `null` l'attribut `objetCache`, lorsque l'objet a été utilisé car il est immuable. Cela pose un problème lorsque l'objet n'est pas détruit après son utilisation car l'objet peut alors être utilisé plusieurs fois et un même objet peut donc être créé plusieurs fois. C'est pour cela qu'il faut créer un deuxième objet qui remplacera l'objet après son utilisation et le passer en paramètre du constructeur. Exemple : le coffre qui possède la clé, qui est de type `ObjetZorkCode` sera, après son utilisation, remplacé par un coffre ouvert de type `ObjetZork`, qui ne possède aucun objet.

- **ConteneurObjetZork**

Classe abstraite, créée pour permettre à d'autres objets de pouvoir contenir des objets de type `objetZork`. Les classes `Sac` et `Piece` en héritent car elles peuvent contenir plusieurs objets. Cette classe a été créée pour éviter de recopier les même méthodes et attributs dans les classes `Sac` et `Piece` et pour permettre de modifier plus facilement le mode de stockage des objets, si nécessaire.

- **Sac**

Hérite de la classe ConteneurObjetZork. Représente l'inventaire du joueur. C'est dans cette classe que sont stockés les objets que le joueur prend. Il ne peut y avoir qu'une seule instance de cette classe, c'est un **singleton**. Cette caractéristique lui permet de pouvoir être accessible n'importe où dans le programme et notamment dans la classe ObjetZork ainsi que dans ces classes filles, qui peuvent modifier le contenu du sac (à l'appel de la méthode actionne notamment).

## 3.2 Les classes modifiées

Dans cette section nous présenterons toutes les classes qui ont été modifiées par rapport au programme de base.

- **Jeu**

### Attribut ajouté

`sac`

Un attribut contenant le sac du joueur. Équivalent à `Sac.getInstance()`

`objetVictoire`

Cet attribut contient la référence de l'objet qui permet de gagner. Est lu dans la méthode `gagne`.

`CorrespondanceCle`

Collection de type `Map` qui fait correspondre chaque instance de la classe `ObjetZorkCle` à une instance de `ObjetZork`.

### Méthode modifié

`creerPiece`

Les fonctionnalités suivantes ont été ajoutées à la méthode :
- Ajout de nouvelles pièces (la bibliothèque et la caverne)
- création et initialisation des objets
- placement des objets dans les pièces
- initialisation de la collection `CorrespondanceCle`
- initialisation de l'attribut `ObjetVictoire`

`afficherMsgBienvennue`

Modification du message de départ.

`traiterCommande`

Dans cette méthode qui associe une commande à une action, les nouvelles commandes ont été ajoutées dans la structure conditionnelle. À la fin de la méthode, un test a aussi été rajouté pour tester si le joueur a remporté la partie (appel de la méthode `gagne`).

`deplacerVersAutrePiece`

Cette méthode est appelée quand l'utilisateur change de pièce. Pour la commande `retour`, il faut donc stocker la direction de la pièce précédente par rapport à la nouvelle pièce. C'est pour cela que le bloc « `switch` » (ligne 241) a été ajouté.
Une instruction a aussi été ajoutée :
« `Piece.set_pieceCourante(pieceCourante);` » (ligne 256), pour changer la pièce courante dans la classe `Piece` (voir classe `Piece` si dessous).

## Méthode ajoutée

`prendreObjetZork`

Méthode ajoutée pour permettre à l'utilisateur de prendre un objet et de le placer dans son inventaire, lorsqu'il utilise la commande `prendre`.

`poserObjetZork`

Méthode ajoutée pour permettre à l'utilisateur de se délester d'un objet de son inventaire et de le placer dans la pièce courante lorsqu'il utilise la commande `poser`.

`examiner`

Méthode qui exécute la commande `examiner`

`retour`

Méthode ajoutée pour pouvoir exécuter la commande `retour`.

`utiliser`

Méthode ajoutée pour pouvoir exécuter la commande `utiliser`.

`gagne`

Méthode créée pour permettre une gestion facile des situations gagnantes. Si on veut changer la manière de gagner, il suffit de modifier cette méthode.

`afficherVictoire`

Méthode qui permet de pouvoir facilement changer le message affiché lorsque le joueur gagne le jeu.

- **Piece**

La pièce hérite maintenant de la classe `ConteneurObjetZork`, elle peut donc contenir des objets.

### Attribut ajouté

`pieceCourante`

Attribut statique, cet attribut à été ajouté pour pouvoir accéder à la pièce courante depuis la classe `ObjetZork`, ainsi que depuis ces classes filles. Cela leur permet de modifier le contenu du tableau d'objet que contient la pièce, notamment dans certains cas lorsque la méthode `actionne` est appelée.

### Méthode modifiées

`descriptionLongue`

La description longue d'une pièce affiche désormais une description plus détaillée des sorties de cette pièce, en affichant, en plus de la direction de la sortie (nord, est, etc) la pièce située dans cette direction. Cela permet à l'utilisateur du jeu une meilleure représentation de la disposition des pièces.

### Méthode ajoutées

`descriptionSortiesLongue`

Génère une chaîne de caractère avec les directions associées au pièce du jeu.

`descriptionObjetZork`

Permet de renvoyer une description de tous les objets que contient la pièce, comme la pièce contient maintenant des objets il faut les ajouter à sa description.