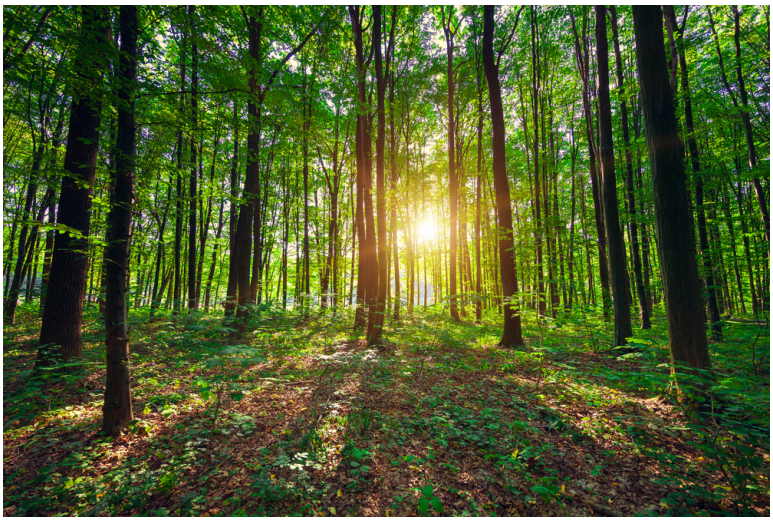
Rapport de projet

Bienvenue dans la forêt ! - Programmation orientée objet



Zineb CHAOUCHE
Licence 2 Informatique

Professeur : Marc CHAMPESME

Université PARIS 13 - Institut Galilée
99, avenue Jean Baptiste Clément
93430 Villetaneuse

Année 2019/2020

Introduction

Dans le cadre de mon cursus scolaire en Informatique, j'ai étudié les bases du langage JAVA durant quelques semaines. En effet, vous trouverez dans mon projet intitulé « Bienvenue dans la forêt ! » le fruit de l'enseignement qui m'a été dispensé par Monsieur CHAMPESME et Monsieur TORRES GONZALEZ, à qui je rends un hommage particulier.

Le jeu « Bienvenue dans la forêt ! » est un jeu où le hasard est roi. En effet, on trouve dans cette forêt des boites mystérieuses. Ces boites renferment soit quelque chose de magique qui mène à la victoire, soit un poison qui mène à la perte…

L'objectif de ce projet est d'implémenter un jeu dont l'auteur choisi les règles, en incluant dans ce jeu des déplacements du joueur et des animaux, la possibilité (ou non) de porter des objets, les reposer, adopter des chiens qui pourront nous aider à porter, libérer les chiens, permettre aux animaux de manger, chasser…

Durant la conception de mes programmes, j'ai fait face à de nombreuses épreuves : élaboration d'algorithmes qui me paraissaient complexes, exceptions de pointeurs 'null', programmes qui compilaient sans erreur mais ne s'exécutaient pas… toutes sortes de difficultés bientôt abordées dans ce rapport de projet.

« Bienvenue dans la forêt ! » : règles du jeu

Dans ce jeu, nous nous trouvons dans la forêt. Vous allez pouvoir porter des boîtes mystérieuses à condition de respecter certaines conditions.

Le but du jeu est de vous déplacer dans la forêt et ramasser les boites qui se trouvent sur votre chemin, à condition qu'elles aient un poids que vous puissiez porter. Ces boites peuvent procurer trois effets sur le joueur : un effet magique, un effet tueur et un effet neutre. Vous ne connaissez l'effet de ces boites qu'une fois que vous les avez ramassées.

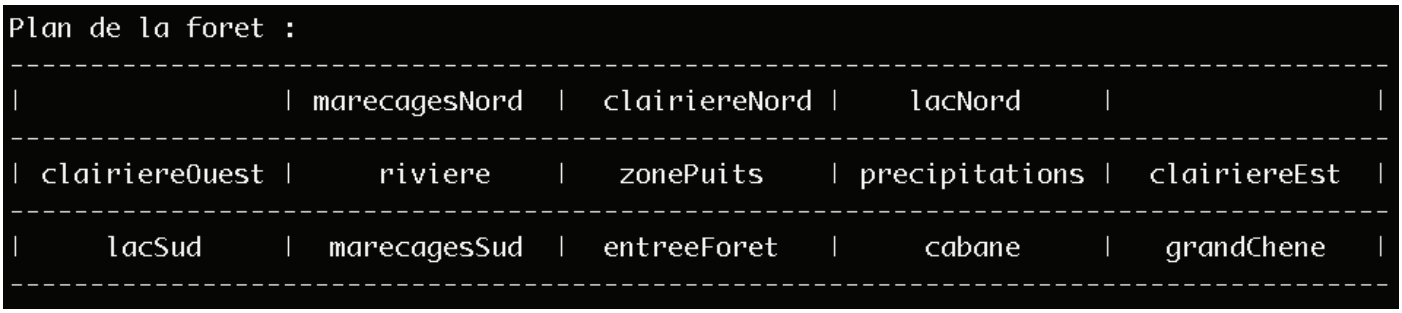
Votre objectif est de ramasser trois boites magiques et de ne pas tomber sur les boites tueuses. Vous l'aurez compris, ici, le hasard est roi. Pour ne pas rendre ce jeu trop facile, vous ne pouvez reposer dans les pièces où vous vous trouvez, que les boites que vous avez ramassé qui ont un effet **NEUTRE.** Toute boite poison ramassée doit rester sur vous. Si vous ramassez trois boites poison, vous perdez !

Les boites de la forêt ont un poids : soit 100 grammes, soit 200 grammes.
Vous, en tant que joueur, n'êtes capable de porter que les boites de 100 grammes et vous ne pouvez porter plus de 700 grammes de boites.

Si vous le souhaitez, vous pouvez adopter et libérer les chiens que vous croisez dans la forêt.

Commandes depuis le terminal

Voici le plan de la forêt :



À tout moment du jeu, vous pouvez retrouver ce plan à l'aide de la commande '`plan`'.

Si vous vous trouvez à l'entrée et que vous souhaitez aller dans la zone 'zonePuits', vous avez simplement à utiliser la commande 'aller' de la manière suivante :

```
aller NORD
```

Et de même si vous souhaitez vous diriger vers lest, l'ouest ou le sud. Veillez à bien entrer la direction souhaitée en majuscules.

Vous l'aurez compris ; vous avez la possibilité d'utiliser une commande 'adopter' suivie du nom du chien que vous souhaitez adopter, pour que celui-ci fasse partie des vôtres. Si le chien Filou vous plait, utilisez la commande suivante :

```
adopter filou
```

La solitude étant appréciée par certains, vous pouvez libérer chaque chien que vous avez adopté avec la commande 'liberer' suivie du nom du chien que vous souhaitez libérer. Ici, cela donne :

```
liberer filou
```

NB : Dans ce jeu, après chaque action du joueur, les animaux libres se déplacent. Ainsi, quand vous libérez un chien, il est normal de ne pas le voir apparaître dans les chiens disponibles de cette pièce, car il s'est déplacé juste après avoir été libéré.

N’oublions pas le principal ! Si vous sentez qu’une boite peut vous intéresser, n’hésitez pas à la prendre. Pour cela, utilisez la commande ‘prendre’ suivie du nom de la boîte, ce qui donne, pour une boîte dont le nom est ‘intuition’ :

```
prendre intuition
```

Vous n’avez plus besoin de cette boîte ? Pas de soucis ! Si elle a un effet neutre sur vous, vous pouvez changer d’avis en utiliser la commande ‘poser’ qui s’utilise comme suit :

```
poser intuition
```

Rassurez-vous, vous n’errez pas seul dans la forêt. C’est pourquoi au besoin, vous pouvez toujours taper `‘aide’` suivi de la commande qui vous pose problème si vous oubliez son utilité. La liste des commandes que vous pouvez utiliser vous sera systématiquement rappelée.

Plan du jeu et scénarios de test

L'implémentation du jeu a nécessité au préalable l'élaboration d'un plan de la forêt avec toutes ses 'pièces', les boites et les aliments initialement existant dans chaque piece et les animaux et chiens éventuels s'y trouvant. Voici le plan initial du jeu, qui sera modifié au fur et à mesure que le joueur adoptera des chiens, que les animaux se déplaceront, mangeront, ou chasseront, ou que le joueur prendra des objets et les posera éventuellement dans d'autres pièces.

Légende :
Boîtes mystérieuses présentes (gras)
Animaux présents (souligné)
Aliments présents pour les animaux (italique)
N = Boite à effet Neutre
M = Boite à effet Magique
T = Boite à effet Tueur

	marecagesNord - **Grace (100 g) M** - **Fraicheur N (100g)** - Chevre2 - *Poulet* - *Tomates*	**clairiereNord** - **Eclat T (100g)** - **Areline N (100g)** - *Insectes* - *Glands*	**lacNord** - **Magie M (100g)** - Chien2 - *Charogne* - *Châtaignes*	
clairiereOuest - **Morphee M (100g)** - *Oeuf* - *Cereales*	**riviere** - **Pureté T (100g)** - **Flora N (100g)** - Chien3 - *Oiseau* - *Noix* - *polypore*	**zonePuits** - **Intuition M (100g)** - Chien1 - *Cadavre* - *Pignons* - *Framboises*	**precipitations** - **Faveur T (100g)** - Chevre1 - *Chair* - *Graines*	**clairiereEst** - **Diamant M (200g)** - **Titiana N (200g)** - Loup1 - *Foin* - *Amanite*
lacSud - **Joie T (100g)** - **Legerete N (100g)** - *Carcasse* - *Herbes*	**marecagesSud** - **Reve M (100g)** - *Écureuil* - *Fêne*	**entreeForet**	**cabane** - **Secret M (100g)** - **Honneur N (200g)** - *Os* - *Champignons*	**grandChene** - **Aisance T (100g)** - **Roselia N (200g)** - *Choux* - *Bolet*

Nous présenterons dans ce qui suit deux scénarios de test : l'un gagnant, l'autre perdant.

Voici les commandes à effectuer pour aboutir à une situation gagnante de fin de jeu :

```
- aller NORD
- prendre intuition
- aller NORD
- aller OUEST
- prendre grace
- aller SUD
- aller OUEST
- prendre morphee
```

Voici les commandes à effectuer pour aboutir à une situation perdante de fin de jeu :

```
- aller NORD
- aller NORD
- prendre eclat
- aller EST
- aller SUD
- prendre faveur
- aller OUEST
- aller OUEST
- prendre purete
```

Modifications apportées au logiciel

A. Liste des nouvelles classes

Dans le cadre de l'élaboration de ce jeu, j'ai été amenée à implémenter de nouvelles classes. Pour chacune d'entre elles, nous indiquerons pourquoi nous l'avons créée, son rôle dans le fonctionnement global du logiciel et les caractéristiques communes des instances de cette classe.

Classe abstraite Container :

La classe Container a été créée dans le but d'éviter la duplication de code dans les classes Joueur et Piece. Ainsi, elle englobe leurs caractéristiques et méthodes communes. Elle a pour attributs une ArrayList d'ObjetZork que j'ai nommée 'contenu' et un nombre d'objets qui représente la taille de cette liste. J'y ai implémenté des getters et setters sur ces attributs et y ait ajouté l'implémentation des méthodes suivantes :

- `public boolean contient(ObjetZork oz);` (qui nous indique si l'instance contient l'objet oz)
- `public void ajouter(ObjetZork oz);` (qui nous permet d'ajouter l'objet oz à la liste d'objets de l'instance)
- `public boolean retirer(ObjetZork oz);` (qui nous permet de retirer l'objet oz de la liste d'objets de l'instance)
- `abstract boolean retraitPossible(ObjetZork oz);` (qui nous indique si le retrait de l'objet oz est possible dans le contenu de l'instance)
- `abstract boolean ajoutPossible(ObjetZork oz);` (qui nous indique si l'ajout de l'objet oz est possible dans le contenu de l'instance)
- `abstract void afficherObjets();` (qui affiche les objets que contient cette instance)

Classe EffetObjetZork :

On le sait, un ObjetZork a un effet sur le joueur. Cet effet peut être magique, tueur ou neutre. Ainsi, on a créé une classe enum pour ces effets, et chaque ObjetZork aura un attribut de cette nature pour nous indiquer l'effet qu'il a sur le

joueur. (On notera que les aliments sont des ObjetZork à effet neutre que le joueur ne peut pas porter).

B. Liste des classes modifiées

—> MotCleCommande.java

À l'attribut '`commandesValides[]`', j'ai ajouté aux commandes déjà présentes les commandes dont on a besoin pour jouer à savoir '`plan`', '`prendre`', '`poser`', '`adopter`', '`liberer`'.

Dans la méthode '`afficherToutesLesCommandes`', j'ai simplement modifié l'affichage des différentes commandes pour une raison ergonomique.

—> Jeu.java

Méthode `creerPieces()` du logiciel Zork :

J'ai modifié la méthode `creerPiece()` pour l'adapter à mon projet à savoir que les pièces dans mon projet correspondent à des zones de la forêt. J'ai donc créé et initialisé chacune de celles-ci.

J'ai également créé et initialisé les objets contenus dans ces pieces qui seront plus tard portés et éventuellement déplacés par le joueur, des animaux qui se déplaceront, mangeront et éventuellement chasseront ou se feront adopter, et des aliments qui seront mangés uniquement par les animaux. De plus, j'ai créé des ArrayList d'objets, d'animaux et d'aliments pour remplir chaque piece grace a son constructeur.

Toujours dans cette même méthode, j'ai initialisé la sortie des pieces. Ainsi, j'ai pu créer un plan concret de ma forêt.

L'interface Animal n'ayant pas de getter sur la piece courante de l'animal en question, j'ai créé dans la classe Jeu une HashMap pour associer à chaque animal une piece. Donc j'ai également rempli cette HashMap dans la fonction `creerPiece()`.

Puis j'ai créé le joueur.

J'ai implémenté dans cette classe une méthode '`clearConsole()`' pour que l'affichage constant du jeu soit plus clair et permette une fluidité et un confort de

jeu. Cette méthode permet d'afficher un terminal de commande vidé de l'historique des commandes précédentes à chaque nouvelle entrée de commande par le joueur.

Méthode `jouer()` du logiciel Zork :

Dans cette méthode, j'ai ajouté l'utilisation de la méthode '`clearConsole()`' dans un premier temps. J'ai aussi apporté des modifications d'affichage comme des tirets '-' que j'ai ajoutés pour y voir plus clair pendant la partie.

Toujours dans un but d'amélioration du confort de jeu, j'ai ajouté un certain nombre d'utilisations de méthodes comme `afficherPlan()`, `afficherArticles()` que j'appelle sur la pièce courante, `afficherToutesLesCommandes()` pour que le joueur sache quelles commandes il peut utiliser a chaque fois qu'il doit en entrer une nouvelle…

Méthode `afficherMsgBienvennue()` du logiciel Zork :

Les modifications que j'ai apportées à cette méthodes sont toujours liées à l'affichage sur le terminal. J'explique au joueur brièvement les règles du jeu et l'utilisation des commandes.

Méthode `traiterCommande(commande)` du logiciel Zork :

Dans cette méthode, j'ai ajouté aux arguments l'argument '`joueur`' pour que les commandes s'y référant puissent avoir lieu sur cette instance-là. En effet, dans le logiciel rudimentaire Zork, la classe Joueur n'existe pas donc il a fallu l'implémenter.

De plus, j'ai ajouté pour chaque commande ne figurant pas dans le logiciel Zork le traitement de celle-ci. Prenons pour exemple la commande '`prendre`' :

Si le seconde mot de la commande correspond à la chaîne '`prendre`' alors j'appelais la méthode `commandePrendre()` qui se chargera d'executer la commande en question. C'est le cas pour toutes les commandes du jeu, je trouve que cela permet de voir plus clair dans la méthode '`traiterCommande(commande, joueur)`'.

Nous voici donc à présent dans la méthode `commandePrendre()`. Celle-ci ajoute les chiens du joueur à la pièce en première instruction et les retire en dernière instruction (voir « Méthodes ajoutées dans cette classe »). En effet, pour chaque méthode relative à une action du joueur, je fais cette manipulation pour que la pièce courante du jeu contienne les animaux du joueur et ainsi ceux-ci peuvent effectuer, comme les autres animaux, les actions comme ramasser un aliment ou le manger.

Ensuite, la méthode compare chaque nom d'objet porté par le joueur au second mot de la commande. Si ce sont les mêmes, alors on fait poser l'objet au joueur.

Des variables que j'appelle « de succès » sont utilisées en cas d'erreur de la part du joueur. Par exemple, ma variable de succès '`entreeFor`' peut m'indiquer que le joueur s'est trompé sur le nom de la boite et cela me permettra d'afficher un message sur le terminal pour rappeler le joueur à l'ordre.

Méthodes ajoutées dans cette classe :

Dans cette classe, j'ai ajouté toutes les méthodes relatives aux commandes passées par le joueur.

De plus, j'ai ajouté une méthode '`suiteActionJoueur()`' qui permet aux animaux de chercher à manger, éventuellement chasser s'il s'agit d'un loup, ramasser des aliments, en manger un, se déplacer, et ce après chaque action du joueur. Ainsi, cette méthode est appelée à la fin de chaque méthode relative à une action du joueur comme '`aller`'.

J'ai également ajouté les méthodes suivantes :

- `setPieceAnimal(Animal ani, Piece p)` qui permet de faire correspondre une pièce à un animal (car on n'a pas de getter sur la piece courante dans l'interface Animal)
- `ajouterChiensDuJoueur(Joueur joueur, Piece pieceCourante)` qui permet d'ajouter les chiens du joueur à la pièce courante et faire associer aux chiens du joueur la piece courante (dans notre HashMap).
- `retirerChiensDuJoueur(Joueur joueur, Piece pieceCourante)` qui, de même que précédemment, permet de retirer les chiens de la pièce à cette pièce.

Conclusion

Ce projet m'a beaucoup appris. D'abord, il m'a permis de réviser une importante partie du semestre. Mais surtout, il m'a permis de joindre l'utile à l'agréable.

En effet, j'ai appris à me servir de mes connaissances dans des cas concrets et je ne me pensais pas capable de produire ce genre de travail, à savoir implémenter un jeu. J'ai réellement pris plaisir à chercher mes erreurs, les trouver et surtout, trouver des solutions. Tester mon programme et le faire fonctionner comme je le souhaitais m'apportait un sentiment de grande satisfaction.

J'ai pu m'améliorer dans ma manière de m'organiser car la charge de travail était assez conséquente compte tenu des autres matières, j'ai appris à répartir équitablement mon travail dans mon emploi du temps.

Mon esprit s'adapte de mieux en mieux aux machines : j'essaie de penser comme une machine et j'ai appris à décomposer un gros problème en plusieurs sous-problèmes. Les longues fonctions ne m'effraient plus comme elles m'effrayaient en début de semestre en Java.