Etudiant :

DeJesusMilitar Dylan
Bilingi Biayi

Architecture du Splendor

1 | Fichiers constituant le Projet

A) Packages

- fr.umlv.copie (Contient la classe copie)
- fr.umlv.objects (Contient les objets servant au jeu : *Cartes, tuiles, jetons*…)
- fr.umlv.game (Contient les fichiers servant au *règles*, et *déroulement d'une partie*)
- fr.umlv.data_files (Contient les fichiers de données : *Cartes_Devs.txt* & *Tuiles.txt*)
- fr.umlv.affichage (Contient les classes liées à l'*affichage du jeu*)
- fr.umlv.players (Contient les différents types de joueurs : *Humain & IA*)
- fr.umlv.saisie (Contient fichier en relation avec la *saisie utilisateur*)

A) Fichiers java

- AffichageLigneCommande.java
- AffichageGraphique.java
- Affichage.java

- Mode
- ModeI
- ModeII

- Tuile.java
- Carte.java
- CarteDev.java
- Jeton.java

- Saisie.java

- Participant.java
- Joueur.java
- IA.java

- Partie.java
- Splendor.java

B) Fichiers stockant les données

Tous les fichiers de données sont rassemblés dans le package : **fr.mlv.data_files**

- Tuiles.txt
- Cartes_Devs.txt

2 | Choix d'implémentation

Dans un premier temps nous allons vous expliquer les différents choix de classe, d'interface et de records.
Nous avons choisis de représenter les Jetons et les Cartes développement par des **record** afin de les utiliser comme des "objets" immuables (non-mutable). Dans une partie ces éléments sont simplement un moyen de stocker les différentes informations que l'on lit (points de prestiges, bonus attribués…) et ne sont donc pas modifiables.

Par ailleurs, on peut voir ces deux éléments comme des sous-types de l'interface Carte (dans Carte.java) car ils peuvent être tous deux utilisés de la même façon à certains endroits (génération de cartes via fichier).

Nous avons décidé de créer une classe AffichageLigneCommande, qui sert uniquement à l'affichage "graphique" dans le terminal. Il nous a semblé important de séparer cette partie car elle est en lien avec la plupart des autres modules et nous permettrait d'envisager une utilisation plus agréable au moment de créer un autre type d'affichage (Phase 3).

Nous avons aussi développé la classe AffichageGraphique, qui est utilisée avec un rôle similaire à AffichageLigneCommande mais pour une version plus esthétique du projet : l'affichage graphique.

De plus, nous avons implémenté les modules "ModeI" et "ModeII". Ces modules servent respectivement à gérer la partie en fonction du mode de jeu. étant donné que "ModeII" correspond au mode de jeu 2, il contient plus de fonctionnalités.

La partie qui gère les saisies utilisateur a fait également l'objet d'une classe (Saisie.java) pour la clarté et la segmentation de certaines tâches dans le programme de déroulement d'une partie lorsque les intéractions avec l'utilisateur sont importantes.

Une classe "Copie" a également été créée. Elle permet de gérer quelques méthodes de copies, notamment les méthodes génériques.

A cela s'ajoute la classe Splendor. Cette dernière est nécessaire pour lancer une partie graphique car il n'est pas possible de lancer depuis la classe "Partie" si elle gère aussi le déroulement de la partie. En effet, cela entraîne un message d'erreur.

Nous avons aussi opté pour une classe permettant de gérer une partie de Splendor (Partie.java) et un joueur de la partie (Joueur.java). L'IA est aussi une classe qui est similaire à Joueur, seulement les fonctions de joueur qui appellent le module Saisie, sont gérées par l'IA tout seul.

En ce qui concerne les interfaces il en existe une pour les différents modes de jeux appelées "Mode". Elle a été créée pour permettre de traiter les situations du jeu différemment selon le mode du jeu et aussi pour le cas où nous aurions développé plus de deux modes.

Il existe une seconde interface qui concerne les modules graphiques. Cette interface se nomme "Affichage" et permet de gérer le jeu différemment, selon si les joueurs jouent avec le clic ou la version qui utilise uniquement le terminal.

La troisième interface permet de gérer les deux types de joueurs, elle se nomme participant. Elle se justifie par le fait que des situations similaires arrivent aux deux types de participants (IA et joueur). Elle est aussi nécessaire pour gérer les situations où ses deux participants différents sont dans des listes.

La dernière interface est l'interface "Carte". Cette interface permet de gérer les cartes de développement et les tuiles car ce sont deux types de cartes.

Ces concepts ne peuvent pas être considérés comme des objets non-mutables, et laissent place à des modifications pour d'éventuelles améliorations ou autre, ce qui nous a poussé à les représenter avec des classes.

Voyons dans un deuxième temps les différents champs qui composent certaines classes et certains records.
Tout d'abord le record "jeton" qui est composé uniquement du champ "couleur". Cela permet de déterminer la catégorie de jeton qui est utilisée en fonction de sa couleur.
Ensuite le record CarteDev. Il est composé des champs niveau, couleur, points, object, HashMap<String, Integer> coût.
Les champs, niveau, couleur et points permettent, comme leurs noms l'indiquent, de déterminer ces informations sur la carte. Le champ "object" permet de connaître le nom de la carte et le champ coût permet de savoir le coût d'une carte et le type de ressource associé au coût pour l'acheter.
L'IA possède, en plus des champs d'un joueur humain, d'autres champ lui permettant de faire sa prise de décision de façon "intelligente", c'est-à-dire sans erreur. Le champ achat_type lui permet de savoir si son prochain achat sera l'achat d'une carte réservée ou du "board". Elle possède aussi le champ "next_token" qui permet de déterminer la ressource qu'il va prochainement acheter. De plus il possède "next_achat" qui lui permet de déterminer les informations de la carte qu'il achètera au prochain tour (son numéro et son niveau). Il possède aussi une graine qui est utile pour ses actions qui sont "aléatoires" mais calculées au sens où **il ne peut pas faire d'erreur**.

Il est aussi important de noter que dans beaucoup de cas, on ne retrouve pas des préconditions sur certains arguments car des valeurs null sont parfois acceptées par le programme.

3 | Liens entres les modules

La partie se déroule dans Partie.java, les méthodes d'initialisation de chaque champ représentant une partie sont propre à ce fichier:

- Pioches (Cartes développement/Tuiles)
- Board (Cartes développement de chaque niveau & tuiles noble)
- Joueurs participant à la partie
- Jetons disponibles dans la banque

Les affichages des choix à effectuer pour le joueur, ainsi que les éléments de la partie (cartes, nobles, jetons…) sont générés, pour la partie "non-graphique " grâce au module : AffichageLigneCommande

L'affichage des états du jeux lors de parties graphiques sont gérés par le module : AffichageGraphique

Les choix effectués par le joueur sont récupérés dans Partie et Saisie (Un des points d'améliorations).

Les actions liées au joueur : achat de carte, réservation de carte, prise de jetons…. sont gérés par le module Joueur.java

Les actions liées à l'IA : achat de carte, réservation de carte, prise de jetons, tester des actions possibles…. sont gérés par le module IA.java

Les actions liées à la partie : déterminer le vainqueur, gestion des pioches… sont gérés localement par le module Partie.

Les méthodes de copie : elles sont gérées par le module Copie.

Les autres modules sont des représentations des objets du Splendor ou des interfaces qui font le lien entre différents modules et sont utilisés dans les champs des classes Joueur, IA et Partie.. Leurs rôles ayant été précisés précédemment nous vous invitons à relire la partie 2/ de ce rapport.

4 | Corrections et changements apportés depuis la soutenance

Dans une quatrième partie, voyons les corrections apportées par rapport aux conseils qui nous avaient été donnés lors de l'entretien :

-nous avons réduit la taille du main qui dépassait les centaines de lignes. Ce dernier a été segmenté grâce à des fonctions et fait maintenant une vingtaine de lignes.

-nous avons utilisé des interfaces comme il nous avait été recommandé pour séparer les deux modes deux jeux, pour séparer l'IA d'un joueur humain ainsi que pour les différents types de carte.

5 | Axe d'amélioration/développement

Dans une cinquième partie, voyons les axes d'améliorations possibles pour le projet.
En ce qui concerne les améliorations possibles, nous avons les suivantes :

-Corriger les problèmes d'affichage liés à la partie graphique. Elle affiche parfois du texte avec du retard.
-Ajouter du texte à chaque fois que le joueur doit se rendre dans le terminal dans la version graphique pour que cela soit plus clair pour lui quand il doit le faire.

-effacer ce qui a été écrit précédemment dans le terminal dans une partie graphique, si cela est possible. Cela permettrait de rendre le jeu plus lisible.

-Utiliser du machine learning pour rendre l'IA plus redoutable.

6 | Problèmes rencontrés

Lors de la réalisation du rendu final, nous avons rencontré un problème dans la conception du fichier build.xml, nous avons eu beaucoup de difficultés à appréhender le développement de ce fichier. Nous n'avons donc pas pu développer ce fichier.

Conclusion

Ce projet nous a permis d'utiliser d'améliorer nos connaissances en java en nous retrouvant dans des situations où nous devions réutiliser des notions comme l'interface ou la copie profonde pour des cas plus concrets. Cela nous a notamment permis de nous améliorer dans notre prise de décision car nous devions choisir d'implémenter ou non certaines fonctionnalités. De plus nous avons pu développer des compétences en apprenant à nous servir de module que nous n'avions pas utilisé en cours, le module graphique.