Manuel développeur

Table des matières

I – Introduction

Le but de ce projet était de réaliser le jeu de société dénommé « Splendor » sur PC à l'aide des connaissances que nous avons acquis tout au long du semestre en programmation objet, à l'aide du Java.

Pour cela, nous devions tout d'abord programmer une version simple du jeu avec des actions restreintes, puis lorsque cette version fut fin prête nous avons pu la récupérer comme base pour la compléter et ainsi avoir la version finale.

Nous allons donc étudier cette première version avant de voir la suite.

II – Version simplifiée

Les différentes classes sont contenues dans le package fr.uge.simpleversion.

CarteDev.java

Cette classe fait référence aux cartes de développements, nous avons tout d'abord le constructeur avec lequel on vérifie si la couleur donnée n'est pas nulle.

Ensuite à l'aide de fromText, c'est une méthode qui ajoute une nouvelle carte.

Joueur.java

Cette classe représente, comme vous l'aurez deviné, le joueur.

On commence tout d'abord par mettre en place les différentes variables qui sont utile au joueur, c'est-à-dire, les jetons qu'il possède, le nombre de prestige etc.

Ensuite nous avons le constructeur, on vérifie bien que les valeurs sont valides et on initialise les variables.

verifJetonJoueur est une méthode qui vérifie si le joueur possède moins de 3 jetons de la couleur donnée en paramètre et renvoie false si c'est le cas sinon renvoie true.
Nous avons ensuite, la méthode ajouteJeton qui ajoute au joueur donné en paramètre un jeton de la couleur elle aussi donnée en paramètre.

Plateau.java

Cette classe représente le plateau de jeu, nous utilisons 2 listes de type CarteDev, une liste pour la pile de carte et une liste pour les cartes qui sont actuellement sur le plateau de jeu en attente d'être achetées par les joueurs.

Comme pour le constructeur de joueur, on vérifie les que les valeurs des jetons sont valides et on initialise les 2 listes.

La méthode adding ajoute une nouvelle carte dans la pile si celle-ci n'est pas nulle.

Ensuite, il y a afficheCarteVisible qui va aller chercher dans la pile et récupérer aléatoirement des cartes et les placer face visible, c'est-à-dire les ajouter dans la liste carte_visible.

Puis, nous avons carteAchat, cette méthode contrôle la couleur donnée en paramètre, si celle-ci ne correspond à aucune des couleurs mentionnées on renvoie null, sinon on renvoie la première carte de la liste carte_visible dont la couleur correspond à celle demandée par le joueur et on supprime cette carte de la liste.

resteJeton compte le nombre de jeton pour chaque couleur encore disponible sur le plateau.

Par la suite, 2 méthodes de vérifications de jetons sont à disposition, verifJeton renvoie true s'il reste au moins 1 jeton de la couleur donnée en paramètre et renvoie false sinon.

Par contre verifJetonSimilaire nous informe en renvoyant true s'il reste au moins 4 jetons de la couleur donnée en paramètre et renvoie false sinon.

Et pour finir, la méthode victoire compare tout d'abord le nombre de prestige entre les joueurs donnés en paramètre, renvoie 1 si le joueur 1 a plus de prestige que le joueur 2, sinon renvoie 2.

Cependant si les 2 joueurs ont un nombre de prestige identique, il faudra donc comparer leur nombre de cartes possédés celui qui en a le plus gagnera par contre si même la somme des cartes est identique on renvoie 0, cela indique ainsi que personne n'a gagné.

Partie.java

Cette classe gère tout l'aspect algorithmique du jeu, on commence donc par retourJeton qui renvoie true si le joueur a dépensé ses jetons pour s'approprier une carte, les jetons utilisés retournent donc sur le plateau, on renvoie false sinon.

Nous avons ensuite, joueurAcheteCarte qui ajoute un point de prestige au joueur qui effectue cette action, renvoie true si le joueur a pu se l'approprier c'est-à-dire s'il avait assez de jeton, sinon renvoie false.

Nous avons, par la suite, 4 fonctions de piochage de jeton.

La première étant joueurPiocheJetonSimilaire, on vérifie tout d'abord s'il y a au moins 4 des jetons choisis par le joueur et après on crédite donc ces jetons au joueur, on les décrédite du plateau et on renvoie true une fois ces étapes complétées, sinon on renvoie false.

joueurPiocheJetonDifferentUneCouleur, joueurPiocheJetonDifferentDeuxCouleurs et joueurPiocheJetonDifferent sont quasiment identiques on peut piocher x jetons de couleurs différentes, la seule chose qui change est lorsqu'il reste moins de 3 jetons au total sur le plateau on fait appel à joueurPiocheJetonDifferentDeuxCouleurs, s'il reste moins de 2 jetons au total on fait appel à joueurPiocheJetonDifferentUneCouleur et par défaut on fait appel à joueurPiocheJetonDifferent.

Cela évite donc une faille qui permet au joueur de piocher des jetons même lorsqu'il y en a plus.

verifAction est une méthode qui demande à l'utilisateur quelle action souhaite-t-il exécuter, le joueur n'a plus qu'à répondre sur a console.

choixAction récupère le choix souhaité par le joueur et effectue une action en fonction de ce choix.

stream demande au joueur quelle couleur souhaite-t-il récupérer, le joueur doit répondre à l'aide de la console.

III – Version complète

Les différentes classes sont contenues dans le package fr.uge.splendor.

CarteDeveloppement.java

Ce record est quasiment identique à la classe CarteDev de la version simple, c'est seulement le constructeur qui change légèrement avec de nouveaux paramètres, on fait donc les vérifications nécessaires pour ces changements.

Petit changement pour fromText car les cartes ont maintenant des niveaux dans cette version complète, on renvoie donc les cartes en fonction de leur niveau respectif.

Joueur.java

Par rapport à Joueur.java de la version simple, nous avons rajouté ici 2 listes de type CarteDeveloppement, reserve et cartepossede, qui représentent respectivement les cartes réservées par le joueur et les cartes possédées, ensuite une liste de type Nobles qui représentent donc les nobles (que nous allons voir plus tard) que le joueur possède et puis une Map qui représente les jetons.

Ensuite, nous avons nbCarteBlanc qui renvoie le nombre de bonus blanc et nous faisons ceci pour le reste des couleurs.

jetonBlancTotal qui retourne le nombre total de jeton (c'est-à-dire jeton + bonus) du joueur, et nous faisons ceci pour le reste des couleurs.

Jeton.java

ajouteJeton initialise les jetons du plateau en fonction du nombre de joueur.

nombreJoueur qui demande à l'utilisateur le nombre de joueur (entre 2 et 4) puis renvoie la valeur envoyé par le joueur.

InputStream

verifAction demande à l'utilisateur quelle action souhaite-t-il effectuer puis renvoie la valeur entrée par le joueur.

choixCouleur demande à l'utilisateur quelle couleur souhaite-t-il puis renvoie la valeur entrée par le joueur.

ManipulationCarte.java

selectionCarteVisible demande au joueur parmi les carte 1 à 4 des cartes se trouvant sur plateau laquelle il souhaite prendre sachant que la carte 1 se trouve à l'extrême gauche et la carte 4 se trouve à l'extrême droite, on renvoie la valeur donnée par le joueur et on soustrait 1 à celle-ci.

selectionCarteReserve demande au joueur parmi les carte 1 à 3 des cartes réservées par lui-même laquelle il souhaite prendre sachant que la carte 1 se trouve à l'extrême gauche et la carte 3 se trouve à l'extrême droite, on renvoie la valeur donnée par le joueur et on soustrait 1 à celle-ci.

selectionCarteNiveau demande au joueur de choisir parmi les piles de carte de niveau 1 à 3, on renvoie la valeur donnée par le joueur.

Nobles.java

Ce record représentera donc les cartes nobles, on initialise à l'aide du constructeur les différents champs du record et on vérifie les champs afin de ne pas avoir de problème de valeur.

Les nobles sont identifiés par un nom et par un nombre x de certaines couleurs.

Plateau.java

Comme pour la classe plateau de la version simple, cette classe représente le plateau de jeu de la version complète, on l'a donc modifié par rapport à la première version.

Tout d'abord nous utilisons un HashMap pour représenter les cartes, la clé est un Integer et fait référence au niveau de la carte et on utilise une ArrayList de type CarteDeveloppement en tant que valeur de cette map qui représente donc la liste des cartes de développement.

Nous avons ensuite, 3 List de type CarteDeveloppement qui eux représentent la liste des cartes faces visibles, 1 liste pour chaque niveau donc.

Puis, il y a 2 autres listes de type Nobles, la première représente toutes les cartes nobles possibles et la deuxième les cartes nobles présentent actuellement en jeu.

Et pour finir, on utilise une nouvelle fois un map mais cette fois-ci pour les jetons, la clé est définie par la couleur et la valeur par le nombre de jeton.

On utilise le constructeur pour initialiser toutes ces collections.

add ajoute les cartes dans une liste en en fonction du niveau de celles-ci.

ajouteCarteVisible1 récupère les cartes de la pile de niveau 1 et les ajoute de manière aléatoire dans la liste de carte visible de niveau 1 (c'est-à-dire les cartes faces développement face visible).
On effectue cette opération pour ajouteCarteVisible2 et ajouteCarteVisible3 pour les niveaux 2 et 3.

addNoble ajoute les cartes nobles dans la liste noble.

afficheNoble récupère les cartes noble et les ajoute de manière aléatoire dans la liste de carte noble.

initJeton initialise les jetons de couleurs vertes, bleues, rouges, blanches, noires à nbJeton et or à 5.

resteJeton vérifie le nombre de jeton dont la quantité est strictement positive, on incrémente un compteur qu'on renverra.

Partie.java

visiteDeNoble on utilise un itérateur pour parcourir la liste des nobles actuellement sur le plateau, puis on le supprime de cette liste et on rajoute +3 points de prestige au joueur.

verifJetonJoueur renvoie true si le joueur a assez de jeton pour acheter une carte, renvoie false sinon.

achatNiveau1 on vérifie tout d'abord si le joueur a assez de jeton pour effectuer l'achat si c'est le cas, on recrédite les jetons du joueur sur le plateau, on lui ajoute la carte qu'on finira par supprimer avant de révéler une nouvelle carte ensuite on renvoie true, sinon on renvoie false.

achatNiveau2 et achatNiveau3 agit comme achatNiveau1 mais pour les cartes de niveau 2 et 3.

achatReserve on regarde si le joueur possède des cartes réservées précédemment, puis on vérifie s'il a assez de liquidité pour effectuer l'achat si c'est le cas, on recrédite les jetons du joueur sur le plateau, on lui ajoute la carte qu'on finira par supprimer avant de révéler une nouvelle carte ensuite on renvoie true, sinon on renvoie false.

joueurPiocheJetonDifferentUneCouleur, joueurPiocheJetonDifferentDeuxCouleurs et joueurPiocheJetonDifferent sont quasiment identiques on peut piocher x jetons de couleurs différentes, la seule chose qui change est lorsqu'il reste moins de 3 jetons au total sur le plateau on fait appel à joueurPiocheJetonDifferentDeuxCouleurs, s'il reste moins de 2 jetons au total on fait appel à joueurPiocheJetonDifferentUneCouleur et par défaut on fait appel à joueurPiocheJetonDifferent.

joueurPiocheJetonSimilaire, on vérifie tout d'abord s'il y a au moins 4 des jetons choisis par le joueur et après on crédite donc ces jetons au joueur, on les décrédite du plateau et on renvoie true une fois ces étapes complétées, sinon on renvoie false.

Reservation.java

reservationPioche cette méthode permet au joueur de réserver directement via la pile de carte, on lui demande ainsi le niveau de la carte dont il souhaite piocher, on ajoute donc la carte réservée dans sa liste, on oublie pas de supprimer cette carte de l'ancienne liste.

reservationVisible cette méthode permet au joueur de réserver directement via les cartes actuellement sur le plateau face visible, on lui demande ainsi le niveau de la carte dont il souhaite piocher, on oublie pas de supprimer cette carte de l'ancienne liste.

typeReservation demande à l'utilisateur s'il souhaite piocher va la pile ou bien via les cartes qui sont face visible.

reservationOr vérifie s'il reste encore des jetons or sur le plateau, si oui lorsque le joueur réserve une carte il récupère donc un jeton or, sinon il ne récupère aucun jeton mais garde la carte qu'il veut réserver.

Victoire.java

Cette classe contient différentes méthodes qui donnent la victoire aux joueurs par rapport au nombre de prestige et/ou de cartes accumulés.

NbJoueurs.java

Cette classe contient les différentes méthodes d'initialisation de la partie en fonction du nombre de joueur présent.

Graphique.java

Cette classe correspond à la partie graphique du projet.

Cliker

On utilise une liste de type Rectangle2D pour pour enregistrer les valeurs lorsqu'on clique dans une zone allouée.

addToken, on ajoute dans la liste les valeurs lorsqu'on clique au niveau des jetons.

addDeck, on ajoute dans la liste les valeurs lorsqu'on clique au niveau des différentes piles.

addCards, on ajoute dans la liste les valeurs lorsqu'on clique au niveau des différentes cartes face visible.

addReserve, on ajoute dans la liste les valeurs lorsqu'on clique au niveau des différentes cartes réservées.

checkClick vérifie si l'on a cliqué sur une zone allouée et renvoie un entier en fonction de la zone où l'on a cliqué, sinon renvoie -1.

Draw

drawBackground affiche le background.

drawLv1 affiche les 4 cases des cartes de niveau 1.

drawLv2 affiche les 4 cases des cartes de niveau 2.

drawLv3 affiche les 4 cases des cartes niveau 3.

drawNoble affiche les cartes de noble.

drawDeck affiche les 3 piles de carte.

drawTokenPlace affiche les places des jetons.

drawPlayerStatus affiche l'interface.

drawPlayerReserveButton affiche les cartes réservées par le joueur.

Values

tokens, affiche la valeur des jetons globaux.

playerTokens, affiche la valeur des jetons du joueur.

playerReserveColor, affiche la valeur des jetons de la carte réservée par le joueur.

playerReserveColor, affiche le nombre de bonus accumulé par le joueur.

noblesValues , affiche la valeur des cartes nobles.

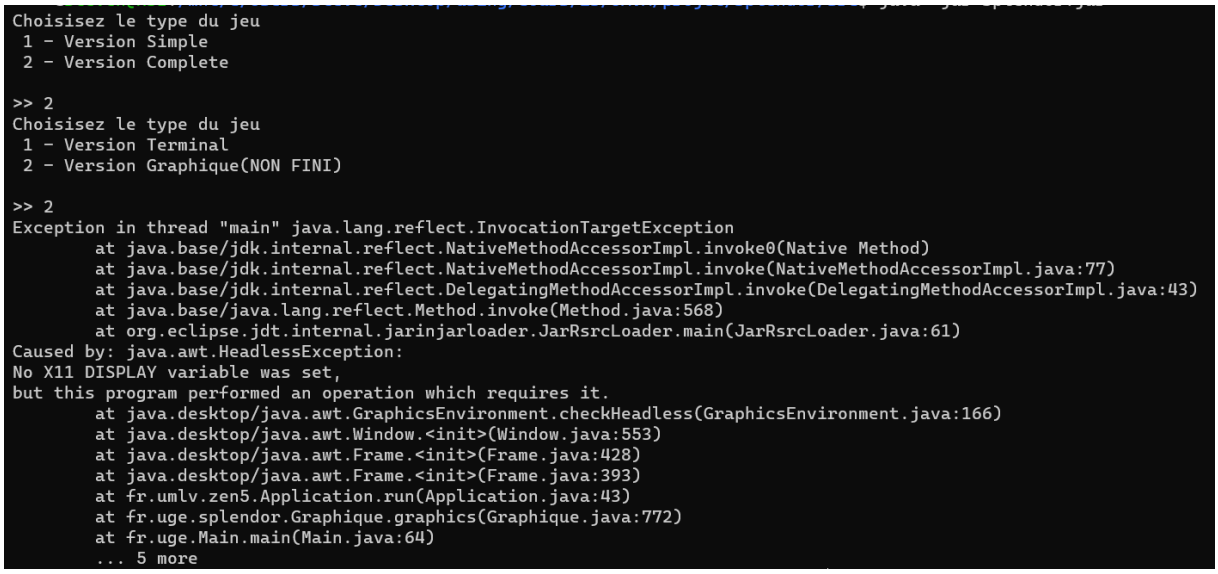
cardValues, affiche toutes les jetons requis pour acheter une carte développement.

Complication

Dû au manque de temps et incompréhension de certaines applications de zen5, nous n'avons pas pu finir la partie graphique.

Lorsque nous lançons le projet avec le jar l'affichage graphique ne s'affiche pas, cependant en utilisant eclipse, le graphique s'affiche, il est tout de même possible de jouer sur le terminal.

(via le terminal)



(via eclipse)

```
Main (15) [Java Application] C:\Program Files\Java\jdk-
Choisisez le type du jeu
 1 - Version Simple
 2 - Version Complete

>> 2
Choisisez le type du jeu
 1 - Version Terminal
 2 - Version Graphique(NON FINI)

>> 2
size of the screen (1536.0 x 864.0)
```