MANUEL DEV SPLENDOR

LEDOUX JOHAN / THAYANANDAN ABIDESH

Architecture

Fichier java	Convention	Description
Main	Class	Prépare les ressources nécessaires pour le lancement du jeu.
Game	Class	Représente le plateau avec les joueurs, les jetons, les cartes de Développement ainsi que les cartes Nobles.
Player	Class	Représente le joueur avec toutes les actions qu'il peut faire ainsi que toutes ses caractéristiques.
Gem	Class	Représentation des gemmes, elles sont la monnaie du jeu.
Cards	Interface	Représentation des cartes présentent sur le plateau.
CardDev	Record	Représente un type de carte, les cartes de Développement, que le joueur peut acheter ou réserver.
CardNob	Record	Représente le second type de carte, les cartes Nobles qui offrent du prestige aux joueurs concernés.
CardCity	Record	Représente le troisième type de carte (seulement dans l'extension "Les Cités") qui changent les conditions de victoire du jeu de base.
Graph	Class	Assemblage de nombreuses méthodes statiques privées permettant d'afficher le jeu graphiquement.
CoordoClicCard	Class	Représente une association entre une carte et un clic, ce qui permet par exemple de différencier les cartes Nobles et cartes Dev dans l'affichage graphique.
Robot	Class	Permet de définir un joueur Robot. Le joueur robot continuera d'utiliser les mêmes méthodes qu'un joueur normal.

Conception

Main

Après avoir récupéré le nombre de joueurs et leur prénoms, on ouvre le fichier des cartes ***data/Cartes.csv*** pour récupérer les **cartes Dev et Nobles**.
Pour éviter les erreurs, on doit ouvrir ce fichier avec un *try-with-resources (try catch)*.
Si l'ouverture est un succès et que les cartes sont initialisées, on va créer les gemmes du terrain puis lancer la partie.
Si on souhaite une partie graphique, on initialisera la fenêtre zen5.

Game

Lorsque le jeu demande une action au joueur, il lance la méthode **select_action** qui va demander l'action à effectuer au joueur actuel.
Pendant la partie, lorsque l'on souhaite update les cartes sur le terrain (parce qu'une carte vient tout juste d'être achetée par exemple) on lance la méthode **updateDevTerrain**.
Cette fonction va piocher des cartes jusqu'à ce que chaque niveau soit rempli de 4 cartes.
updateDevTerrain doit être lancée après chaque action du joueur pour s'assurer qu'il y aura toujours suffisamment de cartes dans le jeu.

Player / Robot

Cette classe possède toutes les méthodes permettant au joueur de jouer, c'est-à-dire acheter une carte, prendre des gemmes sur le terrain, réserver une carte et recevoir la visite d'un noble.
Chaque méthode enregistre les modifications des gemmes et des points de prestige pour le joueur en question, en plus de vérifier si le joueur peut réellement effectuer l'action voulue ou non (si il entre des valeurs valides par exemple).
À la fin de chaque méthode, on lance **updateDevTerrain** pour mettre à jour les cartes du jeu.
Un Robot se comporte comme un Player, on utilise simplement sa classe pour y déposer ses méthodes exclusives à lui.

Gem

Cette classe sert surtout pour **ajouter / supprimer une gemme** d'une certaine couleur, ainsi que pour initialiser une nouvelle liste de gemmes pour le terrain et les cartes.

Cards / CardDev / CardNob / CardCity

L'interface **Cards** permet de lier les méthodes communes entre **CardDev** et **CardNob** c'est-à-dire l'initialisation des cartes depuis le fichier ***Cartes.csv***.
CardDev et **CardNob** sont des Records qui représentent respectivement les cartes de Développement et les cartes Nobles, ils n'ont pas de méthodes compliquées car ils sont simplement présents pour représenter deux types de cartes différents.
CardCity s'utilise de la même manière que les nobles, donc on l'a également inclus dans l'interface **Cards**. Ses méthodes d'actions modifient simplement le fonctionnement de **win**.

Graph / CoordoClicCard

Graph concentre toutes les méthodes d'affichage et d'interactions graphiques. À chaque frame du jeu, on doit lancer **drawGame** pour mettre à jour l'affichage.
La presque totalité des méthodes de **Graph** sont **private static** car on les utilise nulle part ailleurs. **CoordoClicCard** permet de simplifier le code entre les différents types de cartes lors d'un clic dans la fenêtre graphique.

Améliorations et corrections

- **Nous avons ajouté le Robot.** Pour mettre un joueur Robot à la place d'un humain, il faut le nommer "**robot**" lors du lancement du jeu.
  Les actions du robot remplacent donc alors les actions de l'humain, c'est-à-dire que **select_action** va lancer **GameIA** à la place de **GameHuman** pour les choix du robot.
  Le Robot mémorise chaque action réalisée et choisit son action en fonction de l'état du terrain et des gemmes possédées.
  Il va **principalement réserver une carte** et **essayer de l'acheter à tout prix**. Sinon, il peut acheter des cartes sur le terrain à la place.

- **Pour l'extension de la Cité**, l'ajout a été simple et rapide car nous avions déjà créé **l'interface des cartes**. Il suffisait alors d'ajouter les cités dans le fichier **.csv** et de changer les conditions de victoire, tout simplement.

✪ Lors de la **soutenance beta β**, nous avons reçus des retours que nous avons ensuite corrigés:

- ➢ Le code **franglais** est désormais en **anglais** (sauf les commentaires).
- ➢ Les **méthodes de +20 lignes ont été séparées en 2 méthodes** distinctes (sauf une car on avait pas le courage de la modifier).
- ➢ L'affichage en terminal a été légèrement optimisé et est plus propre qu'avant comme demandé (mais bon, qui voudrait jouer à ça alors qu'on a une version graphique incroyable ? Oui je me disais bien…)
- ➢ La méthode **updateDevTerrain** qui était supposée non optimisée est en réalité correcte. Lorsqu'on ajoute les cartes Dev dans le deck (le deck est représenté par une liste, deck qui contient donc tous les Dev de tous les niveaux), à ce moment-là on a pas encore séparé les cartes Dev du deck dans les 3 niveaux de cartes Dev.
  C'est justement le travail de **updateDevTerrain**, faire en sorte de découper proprement le deck pour remplir les cartes Dev dans la liste de liste des niveaux de cartes Dev.
  Voilà pourquoi on a créé la méthode **getSizeByLevel**, c'est cette méthode qui récupère la bonne taille dans le deck non découpé.