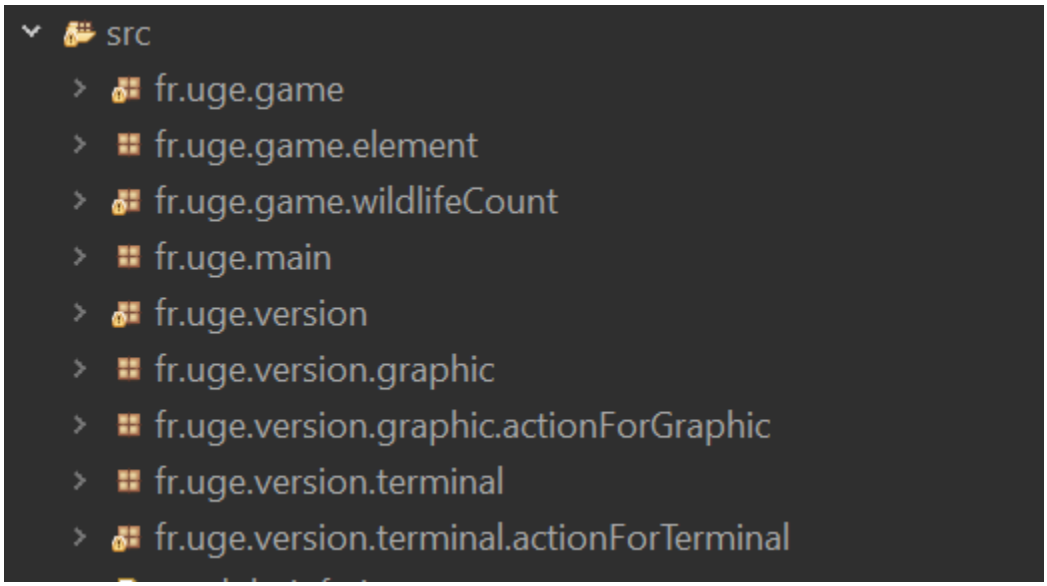
Programmation Objet : Java

Cascadia

Architecture du projet



1) Amélioration apporté depuis la soutenance **1**
2) Structure de donnée **1**
3) Architecture du projet **2**
a) Model 2
b) View / Controler 3

1) Amélioration apporté depuis la soutenance

Depuis la soutenance, nous avons amélioré énormément de points suite aux remarques de notre professeur. Avant cette soutenance, nous avions un programme mal organisé, l'utilisation de fonctions prédéfinies non recommandées et pas d'enum pour les biomes et les animaux . Nous avons ainsi décidé de changer toutes les fonctions pour la lecture sur la console en utilisant les éléments du package " java.io.BufferedReader" et de rajouter un Enum "Biome" et "Animals". Nous avons également décidé de réorganiser tout notre projet pour qu'il puisse être au maximum compréhensible et ranger dans des packages. De plus, nous avons changé notre structure de donnée concernant le plateau en une Map contenant en valeur l'habitat et en clé sa coordonnée.

2) Structure de donnée

Toutes nos structures de données sont stockées dans la classe Structure contenu dans le package "fr.uge.game". Excepté celle du plateau du joueur étant dans la classe joueur du package "fr.uge.game.element" et celle de la pioche contenue dans la classe Draw "fr.uge.game.element".

- plateau du joueur : Nous avons décidé de faire une HashMap<Coordinate, Habitat> pour nous faciliter la gestions des voisins hexagonaux. De plus, nous savons beaucoup mieux gérer les map que les listes de liste.
- pioche : Nous avons deux listes, une pour les tuiles et une autre pour les jetons faunes. De plus la l'indice d'une tuile correspond au même indice que son jeton faune.
- jeton nature : Nous avons une HashMap<Animals, Integer>. Cette map nous sert à associer une animal aux nombre de jetons restant de celui-ci.
- les cartes décomptes : Nous avons décidé de faire une ArrayList<WildlifeCount>.. WildlifeCount est une interface nous servant à relier chaque record des cartes décomptes(explication dans la partie 3)a)).

- joueurs et tuiles de départ et normal : Nous avons décidé de faire une ArrayList pour les tuiles et les joueurs car c'était la structure qui nous paraissait la plus approprié et une liste de liste de tuile de départ parce que nos tuiles de départ sont constitué de de trois tuiles.

3) Architecture du projet

Nous avons décidé de faire faire trois gros package contenant eux même des package. L'un étant "fr.uge.main" contenant le main, l'autre étant "fr.uge.game" contenant les éléments du jeu et enfin "fr.uge.version" contenant les affichage, action et choix du joueur. De plus, l'implmentation du model view controler est respecté dans notre architecture.

a) Model

La partie model, correspond à notre package "fr.uge.game". Dans celui-ci, nous allons avoir tous les "petit" et "gros" élément correspondant à la programmation de chaque éléments.Les gros element font référence aux classe permettant la mise en place de notre jeux ainsi les classes :

- Structure (pour les structure de donnée)
- GameRound (pour un tours de jeux)
- CountPoint (pour le comptage des points)
- Menu (pour le menu afficher en terminal qui nous avons besoin pour la mise en place du jeux)

Nous avons également une interface scellée nommée "WildlifeCount" permettant de relier les petits éléments, contenue dans l'un package dédié et étant "fr.uge.game.wildlifeCount". Nous avons décidé de faire cette interface pour nous faciliter l'implémentation des cartes décomptes et de ces variantes mais également car il nous a été recommandé par notre professeur

de le faire. De plus, celle-ci est scellée car nous ne pouvons ajouter aucune autre carte décomptée.

Les autres “petits” éléments sont dans un package “fr.uge.game.element”. Ce package nous sert principalement à la lisibilité de l’architecture.

b) View / Controler

La partie view et controler sont tous les deux contenue dans le même package étatant “fr.uge.version”. Nous avons dans ce package dédié une partie pour pour le controler et une autre pour la view.

Il y a plusieurs parties dédiées à la view. La première partie est dans le package nommé “fr.uge.version.graphic” (ce package contient une partie controler que nous verrons par la suite). Nous avons dans celui-ci deux autres package, un étant également pour le view et un autre étant pour le contrôler. Le package “fr.uge.version.graphic” contient deux records pour l’affichage de la pioche hexagonale et carré. Ces deux records sont reliés entre eux par une interface pour faciliter l'écriture du code, nous éviter une classe assez longue pour afficher la pioche et nous éviter les répétitions de variable. De plus nous avons une classe s'appelant DisplayForGraphic permettant l’affichage graphique. Nous avons cette classe également dans le package “fr.uge.version.terminal”. Ces deux classes sont reliées par une interface se nommant “Display”, celle-ci est implémentée pour les mêmes raisons que toutes les interfaces précédentes. Nous avons eu le même raisonnement pour l'action et les choix du joueur.

Pour la partie controler, nous avons décidé de la séparer en deux classes: une pour les choix du joueur et une pour les actions du joueur. Ces deux classes sont faites pour la version terminale et graphique. Elles sont ainsi dans le package “fr.uge.version.terminal.actionForTerminal” et le package “fr.uge.version.graphic.actionForGraphic”. Nous avons décidé de les mettre dans le package version où se trouve également la view parce qu’il

nous paraissait cohérent de les mettre ensemble parce que tous les deux font de l'affichage.

c) Classe englobant d'autre classe

Il y a la classe Draw qui englobe Structure et Player. Nous avons décidé de mettre Structure dans Draw car cette classe contient toutes les structures de données. De plus, si nous le faisons pas chacune de nos fonctions aurait eu énormément de paramètre car Draw est une classe permettant de mettre dans la pioche de nouvelle tuile et jeton faune et ainsi les enlève de la pioche se trouvant des listes de tuiles se trouvant dans la classe Structure. Nous avons également mis la classe Player dans Draw car celle-ci était souvent prise en paramètre de fonction. Ainsi cette implémentation permet de rendre le code plus clair.

Il y a également la classe Habitat qui englobe le record Tile. Cette classe prend ce record car il nous semblait logique d'avoir

4) Implémentation réalisé

Phase 1 :

- nombre de joueur 2 ✅
- implémentation tuile carré ✅
- carte famille et intermédiaire ✅
- affichage terminale ✅

Phase 2 :

- zen pour les tuiles carrées ✅
- MVC ✅

Phase 3 :

- cartes décomptes (pour carré et hexagonale)
    - renard A ✅ B ✅ C ✅ D ✅
    - buses A ✅ B ✅ C ✅ D ✅
    - ours A ✅ B ✅ C ✅ D ✅
    - wapiti A ✅ B ☑️ C ✅ D ☑️
    - saumon A ✅ B ✅ C ✅ D ✅
- implémentation tuile hexagonale ✅
- zen pour les tuiles hexagonale ✅
- joueurs entre 2 et 4 ✅

Phase 4 :

- mode solo ❌
- sauvegarde ❌
- réussite cascadia ❌

5) Répartition des tâches

Nous sommes reparties la phase 1 et Ryad a fait la phase 2 et Féryel a fait la phase 3. Plus globalement Feryel s'est occupé des tuiles carrés et Ryad hexagonal.