Manuel d’architecture du code :

ANTONI Jade, TWARDAWA Yanis

Sommaire

1. Introduction
2. Structure globale du projet
3. Organisation des packages
4. Améliorations et corrections depuis la soutenance β
5. Conclusion

---

1. Introduction :

Ce projet avait pour but de créer une version informatisé du jeu de société nommé Cascadia ; un jeu de plateau stratégique où l’objectif est de créer des combinaisons entre jetons animaux et tuiles habitats afin de recréer l’écosystème de la région.
Le projet nous a permis de mieux comprendre comment découper correctement un code Java en packages. Il nous a obligé à réfléchir à de nombreuses reprises sur nos implémentations ainsi sur le choix des types de nos classes.

---

2. Structure globale du projet :

Le projet est organisé en plusieurs répertoires et fichiers principaux :

- **src/** : Contient les sources du projet, organisées en packages pour les différents modules du jeu.
- **docs/** : Contient les manuels utilisateur (*user.pdf*) et développeur (*dev.pdf*), ainsi que la Javadoc dans son répertoire dédié : doc.
- **lib/** : Contient les bibliothèques externes nécessaires à l’exécution du projet (zen6).
- **classes/** : Contient les fichiers compilés (.class) pendant l’exécution.
- **build.xml** : Script ant permettant la compilation, le nettoyage, la création de l'archive JAR, et la génération de la documentation.
- **README** : Rapport, en format Markdown, de l’organisation du travail et des problèmes rencontrés tout au long du projet.

---

3. Organisation des packages :

Chaque package

1. Module dédié aux animaux :
    - **fr.uge.animal** : Enum listant les différents types d'animaux présents dans le jeu. Cet enum a été placé seul dans son package pour montrer son importance et pour plus de lisibilité. Il sert d'élément à part entière du jeu, permettant l'implémentation des jetons ainsi que des cartes animaux.
    - **fr.uge.animal.card** : Regroupe les classes liées aux cartes animaux : un record pour l'implémentation d'une carte, une classe pour leur gestion. Elles permettent de déterminer les configurations gagnantes des plateaux, et donc la stratégie à adopter pour les joueurs.
2. **fr.uge.bag** : Implémente la gestion des sacs de jetons, de tuiles, et des tuiles de départ.
3. Module dédié au plateau :
    - **fr.uge.board** : Enum listant les deux types de plateau : celui à tuile carrée et celui à tuile hexagonale. Tout comme l'enum « animal », celui-ci a une place à part dans son package puisque l'entièreté d'une partie, des variantes, et des plateaux, tournent autour de celui-ci.
    - **fr.uge.board.players** : Regroupe les classes liées aux plateaux des joueurs. Permet la mise à jour des informations du plateau comme l'état de leur écosystème ou la progression dans la partie.
4. **fr.uge.game** : Contient les classes liées aux modules du jeu en lui-même : un enum pour les variantes, un record pour l'implémentation d'un début de partie avec tous ses paramètres, une classe pour la gestion de la partie.
5. **fr.uge.graphic** : Regroupe les classes liées à l'affichage terminale et l'affichage graphique Zen6, gérées par une interface commune, simplifiant ainsi le passage de l'un à l'autre ainsi que l'ajout de nouvelles méthodes.
6. **fr.uge.interaction** : Contient les classes s'occupant des interactions entre l'utilisateur et le jeu lors d'une partie en format terminal ou en format graphique, toutes deux sous une interface commune.
7. **fr.uge.main** : Point d'entrée du programme, contenant la classe principale.
8. **fr.uge.score** : Regroupe les classes implémentant les cartes de scores et calculant le score d'un plateau d'un joueur.
9. **fr.uge.stack** : Regroupe les classes implémentant la pioche tout au long de la partie, et gérant les cas de surpopulation.
10. **fr.uge.tile** : Contient toutes les classes liées aux tuiles : les tuiles de départ, les tuiles d'habitats, ainsi que leur format carré et hexagonal lié par une interface commune.

---

4. Améliorations et corrections depuis le rendu β :

1. Avancement dans la partie graphique, début d'affichage graphique.
2. Changement d'implémentations : nous avons revu plusieurs de nos implémentations, tel que les coordonnées de position de nos tuiles, qui allait poser problème lors de la phase suivante.
3. Changement de plusieurs classes en record selon les recommandations de la soutenance.

---

5. Conclusion :

Notre but pour ce rendu était de rendre un projet clair, modulaire et fonctionnel.
De nombreuses fonctionnalités pourraient être optimisées, voire même encore implémentées avec un peu plus de temps. Malheureusement, le temps nous a manqué à cause de problèmes personnels (Cf README pour plus de détails).
Cependant, nous nous sommes assurés d'avoir une bonne implémentation, afin d'avoir une base solide et extensible au besoin, malgré la phase 4 et une partie de la phase 3 de manquantes.