CHABANE Massinissa

Rapport du Projet

---

1. Introduction

Le projet consiste à créer une simulation de jeu Cascadia en proposant trois versions ou mode de jeu

- Un mode terminal avec des tuiles carrées.
- Un mode graphique avec des tuiles carrées.
- Un mode graphique reprenant la version classique du jeu avec des tuiles hexagonales.

2. Structure du Projet

J'ai organisé le projet en packages qui séparent chaque tâche des autres.

`fr.uge.cascadia`

- **Game.java:** une classe qui gére et lance la partie et le choix du mode du jeu et autre
- **Player.java** represente un joueur qui est représenter par un nom, un score, un plateau et autre

`fr.uge.cascadia.board`

- **Board.java:** gère le plateau qui est la grille de tuiles et gère le placement des tuiles et des jetons ainsi que des fonctions de parcours du plateau pour trouver des groupe d'animaux ou d'habitat;
- **Shelf.java** s'occupe des paires jetons et tuiles qui servent de choix au joueur qui joue son tour, cette classe permet de gérer le cas de la présence de 3 jetons similaires dans un tour, permet de gérer le mode solo et de simuler ce qui a été demandé par le sujet.

`fr.uge.cascadia.controller`

Dans ce package j’ai essayé de rassembler ce qui permet de gérer les tours et l’interaction avec les différents utilisateurs

- **GameInterface.java** une interface implémentée par ControllerTerminal et ControllerGraphic.

Cette interface aide à limiter la duplication de codes et la réécriture du code **contrairement à ce que j'ai produit lors du premier rendu bêta.**

  - **ControllerGraphic.java** un record qui sert à interagir avec l’utilisateur et la visualisation graphique grâce à des cliques de souris ou de clavier, comme le choix des tuiles, prendre des décision comme l’utilisation des jetons nature ou remplacer les jetons similaires….
  - **ControllerTerminal.java** Implémente un mode terminal qui permet de visualiser le plateau et les information dans le terminal et interagir avec l’utilisateur avec les insertion textuelles dans le terminal

.

- **GameManager.java** gère les differentes boucles du jeu et la logique du jeu et organise la suites des étapes, gère aussi les boucles d’insertion et de rotation

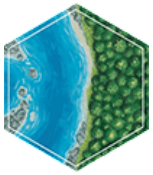
`fr.uge.cascadia.tile`

Dans ce package j'ai essayé de mettre ensemble tout ce qui concerne la représentation d’une tuile dans le jeu.

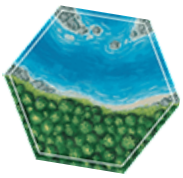
- **TileType.java** enumération qui contient deux type de tuiles (hexagonales ou carré)

- **Habitat.java** enumération qui contient les types d'habitat dans le jeu
- **Tile.java** une interface commune pour les tuiles. Elle sert aussi à créer un sac de tuiles carrés ou hexagonal
    - **SquareTile.java** une tulle carrée qui contient un seul habitat et exactement deux animaux compatibles et qui ne supporte pas de rotation créée d'une façon aléatoire.
    - **HexagoTile.java** une tuile de type hexagonal qui peut avoir entre 1 et 2 habitats, et entre 1 et 3 animaux compatibles. Une tuile hexagonale supporte la rotation gauche et droite et pour mieux représenter cette rotation nous avons une liste de 6 habitats, les trois premiers indices représentent le premier habitat et les 3 derniers représentent le deuxième.

      EX:

      cette tuile contient une liste de 6 qui est:
      {rivers ,rivers , rivers , forests , forests , forests}

      et à une rotation de 60° ça devient
      {forests , rivers ,rivers , rivers , forests , forests }

`fr.uge.cascadia.animal`

Dans ce package on s’occupe des jetons animal ou des animaux dans le jeux

- **Animal.java** enumération des differents types d’animaux dans ce jeu
- **AnimalToken.java** qui représente un jeton spécial d’animal avec un record, on crée aussi un sac de 100 jetons animal (20 chaque type).
- **CardType.java** une énumération qui n’est utilisée que pour calculer le score mais on l’a mis ici car c'est liée aux animaux donc le score sera calculé avec l’une de ces cartes (A,B,C,D)

`fr.uge.cascadia.score`

Ce package regroupe tous les outils utilisés pour calculer le score d’un joueur à partir de son plateau.

- **Score.java** gère le calcul des scores pour chaque joueur, ça permet aussi de gérer les bonus et autre
- **BearScoring.java, ElkScoring.java, FoxScoring.java, BuzzardScoring.java , SalmonScoring.java** sont tous des règles spécifiques selon la cartes faune choisie, chaque .java ici contient 4 cartes faunes en méthodes
- **ScoringCard.java** est une interface implémenté par les animal scoring précendentes
- **ScoringStrategy.java** est une interface qui indique la stratégie utilisée pour le calcul du score, elle est implémentée par
    - **VariantScoring.java** qui représente le calcul du score lié aux animaux en utilisant une seule carte pour tous les animaux qui est famille ou intermédiaire

    - **FauneScoring.java** qui représente le calcul du score lié aux animaux en utilisant une carte faune pour chaque animal.

- **HabitatAnalyzer.java** est une interface sert à calculer les scores liés aux habitats qui implémente **HexagoHabitatAnalyzer.java et SquareHabitatAnalyzer.java**

`fr.uge.cascadia.view`

Gère tout ce qui est de la représentation graphique

- **GameView.java** une interface qui s'occupe de l'affichage en graphique du tableau et autre information du joueur ainsi que la conversion des coordonnées des clics de souris coordonnées dans la tableau et autre. Ainsi que l'affichage de l'écran de la fin du jeu. Elle est implémentée par **HexagoView.java** et **SquareView.java**
- **ViewUtils.java** fournit des méthodes utilitaires pour gérer l'affichage
- **ImageLoader.java** s'occupe de charger les images et les rotations

! cette classe a été reprise de l'exemple fourni par l'enseignant

`fr.uge.cascadia.success`

Ce package s'occupe des réussites de cascadia et des objectifs atteints par le joueur.

- **ScenarioSuccess.java** gère les succès liés à des scénarios prédéfinis
- **GameSuccess.java** gère les succès liés au à des réussites d'une partie dite `normale`

- **SuccessManager.java** une interface qui s'occupe de verifier les succès scénario et partie normale et s'occupe de l'écriture d'un fichier pour les stickers;
- **SuccessUtils.java** fournit des méthodes pour vérifier les succès réalisés.

3. Explication de nos choix

1. Après consultation des enseignants j'ai décidé de garder les mêmes tuiles hexagonales pour toutes les parties, même si la classe de **HexagoTile.java** nous donne la possibilité de le faire d'une façon aléatoire. J'ai donc créé deux fichiers `initialTiles.txt` et `hexagoTilesFile.txt` dans lesquels chaque ligne est représenté par une tuiles hexagonales de façon

   nb habitat1 habitat2 nb animal1 animal2 …

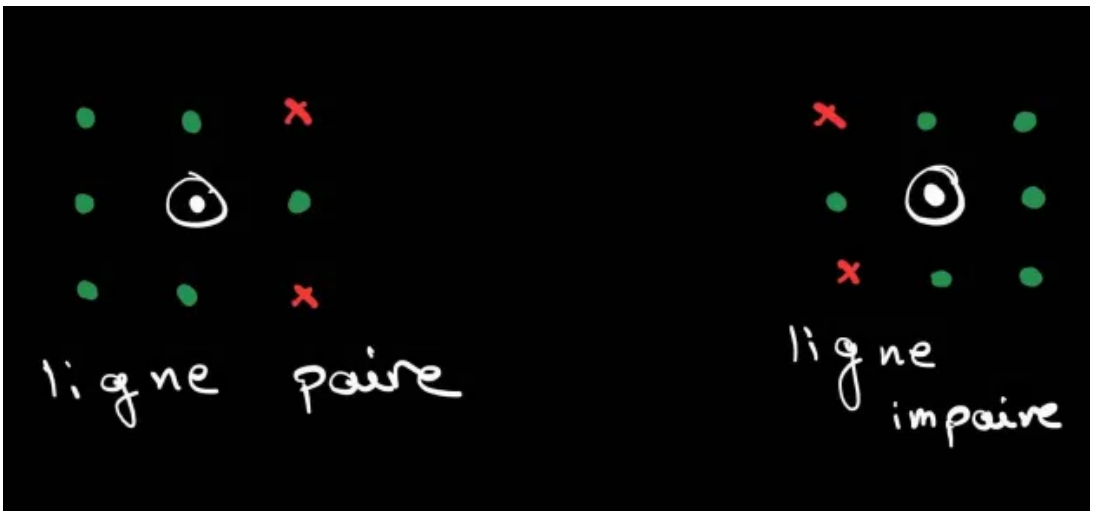
exemple: 2 Forests Wetlands 2 Bear Fox

2. Pour la représentation d'un plateau avec des tuiles hexagonales j'ai gardé la même conception que les tuiles carrées en changeant juste les voisins.

   une tuile carrée a 4 voisins

   { (x,y-1) , (x,y+1) , (x+1,y) , (x-1,y) }

   Mais une tuile hexagonale à 6 voisins, dans notre cas les voisins changent selon la parité de la ligne.



4. Données et ressources

- **hexagoTilesFile.txt** les tuiles hexagonales du jeu en ligne de texte
- i**nitialTiles.txt** les tuiles initiales en ligne de texte.
- **hex/** un dossier contenant des images de tuiles hexagonales
- **data/** un dossier contenant des images de tuiles carrées et de jetons animal
- **Scenarios.txt** Contient des scénarios prédéfinis
- **successNormal.txt** contient des succès standard

Outils et liens externes:

- Ce projet a été réalisé avec la bibliothèque graphique Zen6.

5. Difficultés

Parmi les difficultés que j’ai rencontrées lors de la réalisation de ce projet

- La gestion des grilles hexagonale
- Assurer la transition entre les mode graphique et terminal
- Calcul des score et validations des succès
- Gestion du temps

6. Perspectives d'amélioration

- Ajouter un mode multijoueur en ligne
- Intégrer une intelligence artificielle en mode solo

7. Utilisation du build.xml

- **ant compile :** compiler les sources
- **ant jar :** création du Cascadia.jar
- **ant javadoc :** génère la javadoc dans dans le répertoire docs/doc
- **ant clean :** nettoie le projet

8. Modifications apportés au rendu beta

- Suppression de duplication de code entre terminal et graphic controller
- Reproduction du ImageLoader
- Respect du modèle MVC
- Non-oubli des Objects.requireNonNull();