Rapport groupe 142

Mattyas MARQUES

February 25, 2023

Contents

1 Problème **1**
1.1 Modélisation des piece et du plateau 1
1.1.1 Pieces<E> 1
1.1.2 Plateau 2
1.1.3 Joueur 3
1.1.4 Jeu 4
1.2 Généricité et intelligence des placement des fonctions 5
1.3 Générer des pieces jouables 5

2 Representation graphique des classes **5**

3 Partie du cahier des charges traité **6**
3.1 Interface Textuels 6
3.2 Règles du jeu de domino 7
3.3 Règles de Carcassonne 7
3.4 Interface graphique 7
3.4.1 JFrame des Joueurs 7
3.4.2 JFrame MainPage 7
3.5 HAL 9000 8

1 Problème

1.1 Modélisation des piece et du plateau

1.1.1 Pieces<E>

1. Domino Cette classe est une extension de Piece<Integer> ce qui force l'utilisation des Integer pour les valeurs du tableau vals. L'Override de

posable lui fait les vérifications nécessaires pour pouvoir poser sur le plateau.

2. Carcassonne Cette Classe est une extension de Piece<ObjectCarcassonne> et gere elle meme la creation de pièces.

3. Généralisation des classes

   Le but est de généraliser et forcer la création de toutes les fonctions dont le plateau pourrait avoir besoin ainsi il y a :

   (a) Tout les champs
   - Le tableau des valeur de E
   - La position relative de la piece par rapport au centre
   - Les pointeur vers les autres pieces dans les 4 direction
   - Des boolean indiquant pour chaque direction si il existe quelque chose

   (b) Certains getters

   (c) La fonctions `public void prepPlateau(List<Piece> posable)` qui va preparer le tableau pour le tour du joueur suivant

   (d) La fonction `tournerUnePiece()`

   (e) La fonction `contains(s(List<Piece> set ,int x , int y)`

   (f) La `equals(Piece obj)`

   (g) La fonction `abstract posable(String orientation , Piece pieceSurPlateau , Piece aPoser)`

   (h) La fonction `int comptePoint(Joueur j, Piece<E> up, Piece<E> right, Piece<E> down, Piece<E> left)`

1.1.2 Plateau

1. Les champs Un premier champs qui est `Collection<Piece<E» pieces` le but d'avoir une Collection est de ne pas avoir à chercher les doubles et pouvoir faire des action avec les stream de donnez et pouvoir créer des map pour l'affichage ou ne pas avoir de fixation au niveau de la taille ou longueur de notre modèle de donnée. Par exemple avec : `coll.stream().collect(Collectors.groupingBy(Piece::getY))` on peut facilement récupérer ligne à ligne le plateau sans avoir de cases vides comme dans un tableau. Et deux autres champs `public HashSet<Piece<E» piecesLibre` et `public ArrayList<Piece<E» piecesPosable`

qui vont servir à garder en mémoire les pièces ou on peut poser un domino, et l'autre qui garde les endroits on ou pourra poser les pièces

2. Les fonctions

   (a) `public HashSet<Piece<E» freePiece()` qui va renvoyer un hashSet composer de toute les piece qui on au moins un coté libre

   (b) `public HashSet<Object[]> borderCombinaison(HashSet<Piece <E» bordersE)` méthode qui elle renverra un HashSet avec tout les triplets ou on peut poser un domino et en plus l'index du côté sur lequel le triplet se trouve sur le domino

      - Le but d'utiliser ici des hashSet est de pouvoir ajouter les triplets sans ce soucier de double

   (c) `public int getMaxX()` et `public int getMaxY()` les deux méthodes elle renverront le minima et le maxima des positions en X des pièces du plateau courant

   (d) `public int getMaxY()` et `public int getMaxY()` les deux méthodes, elles renverront le minima et le maxima des positions en Y des pièces du plateau courant

   (e) `public boolean placePiece (Piece piece , int x , int y , Joueur j)` cette méthode va servir à verifier si la Piece piece est posable en x , y et la placera si c'est possible

   (f) `public void place(Piece piece , int x , int y , Joueur j)` méthode qui place la Piece piece

   (g) `public void prepPlateau()` prépare le plateau pour pouvoir monter et récupérer toutes les coordonnées ou sont posable des pièces à ce tour avec le plateau actuel

   (h) `public int getMaxX()` et `public int getMaxY()` les deux methodes, elles renverront le minima et le maxima des positions en X des pièces du plateau courant

   (i) `public int getMaxY()` et `public int getMaxY()` les deux methodes, elles renverront le minima et le maxima des positions en Y des pièces du plateau courant

1.1.3 Joueur

1. `whatToDo(Controleur c)` qui va demander au controleur de lui renvoyer l'action du joueur

2. `draw(Plateau plateau , Jeu jeu)` va piocher et en fonction du plateau actuel piochera un piece avec de grande chance d'etre posable

3. `playerAction(Controleur controleur , Plateau plateau)`

4. `dumpCurrent()` Défausse la pièce pioché en mettant la valeur de la pièce à null.

5. `addPoints(int comptePoint)` ajoute les points au joueur

1. Bot normal

   (a) `public boolean playerAction(Controleur c,Plateau p)` Reprend la fonction prise dans les fonctions gérant les tours, on créer un tableau tabPosable bidimensionnel de int, on teste chaque pièce libre du plateau pour tester si la pièce piochée rentre dans une case, si elle n'y rentre pas -1 est placé dans le tableau. On test ensuite si le tableau est rempli de -1 si oui il défausse la carte, si non il choisit une pièce au hasard et pose donc la pièce.

   (b) `private void tournePiece(int i)` sert à tourner la piece i fois.

   (c) `private static boolean isEmpty(int[][] tab)` sert à regarder si le tableau est remplis de -1.

1.1.4 Jeu

1. `public boolean tour()` permet de gérer automatiquement un tour et envoi un boolean true si c'est la fin d'une partie.

2. `public void play(boolean isDomino)` lance la parti et fait notamment un while(!tour){} qui permet de jouer tour apres tour.

3. `private void win()` Affiche le classement des joueurs en appelant la fonction `afficheClassement()` de la class Controleur.

4. `public int getJoueursPoints(int indJoueur)` Getter qui renvois le nombre de point du joueur, l'argument est l'indice du joueur permettant de retrouver simplement le joueur.

5. `public String getJoueursName(int indJoueur)` Getter qui renvois le nom du joueur, l'argument est l'indice du joueur permettant de retrouver simplement le joueur.

6. `public void nextPlayer()` augmente de 1 l'indice et le remets au premier indice s'il est nécessaire

7. `public Object[][] getWinners()` Renvois le classement des joueurs sous forme de tableau.

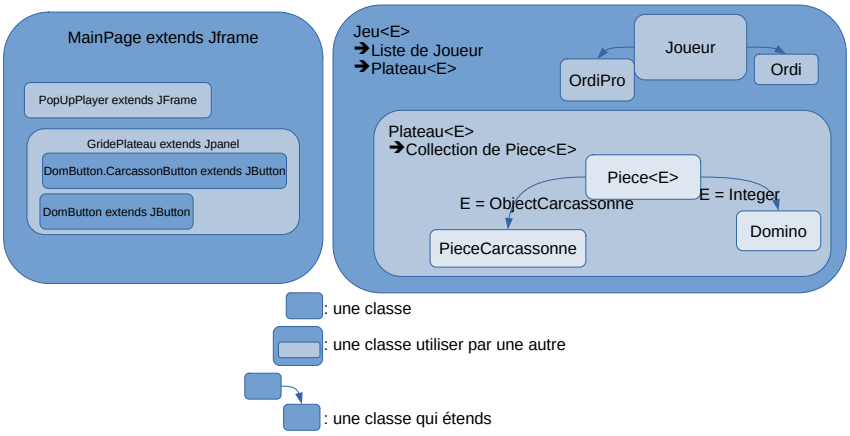
1.2 Généricité et intelligence des placement des fonctions

Cette partie nous a causé beaucoup de problèmes en effet la différence entre une Piece et une Piece<?> nous a été difficile à comprendre et interpréter. Cette notion nous a causé tant de problèmes que le fait de pouvoir poser des pions n'a pas été réalisé à cause de cette utilisation de la généricité et de la mauvaise gestion que nous en avons eu.

1.3 Générer des pieces jouables

Pendant un long moment au début du projet ce problème nous a fait réfléchir. Comment rendre le jeu jouable, nous avons décidé de trouver un moyen de toujours récupérer des triplets de chiffres posables. En faisant le tour du terrain nous récupérons tous les triplets ou on peut poser une pièce et on en choisit entre 0 et 4 au hasard pour en faire une nouvelle pièce au detail près que nous ne mettons pas le triplet comme ça dans la nouvelle pièce nous inversons ce triplet pour que le domino soit effectivement posable.

2 Representation graphique des classes



3 Partie du cahier des charges traité

3.1 Interface Textuels

Chaque fonction servira à imprimer dans le terminal ou déclarer l'interface graphique. Chaque fonction est nommée après ce qu'elle fait toutes les fonctions askQuelQueChose() seront des fonctions que le contrôleur récupère pour interpréter les données et les transmettre au model. Au contraire chaque fonction afficheQuelqueChose() est utilisée dans le sens inverse et affichera les données traitées par le contrôleur récupéré de l'état actuel du model.

1. `public <E> void afficheListePiece(Collection<Piece<E» list)`
2. `public <E> void afficheListePiecePosable(Collection<Piece<E» list , ArrayList<Piece<E» listeWherePosable)`
3. `private <E> void afficheListePieceLine(Collection<Piece<E» coll , int line)`
4. `private <E> void afficheListePieceLinePos(Collection<Piece<E» coll , int line , List<Piece<E» listeWherePosable)`
5. `public void afficheMain (Piece p)`
6. `public void affichePiece(Piece p)`
7. `private String posableInd(int i)`
8. `public void affichePointJoueurs(String playerName , int joueursPoints)`
9. `public String askWhereToPlay(String name)`
10. `public String askWhereWhatToDo()`
11. `public void afficheClassement(ArrayList<Joueur> joueurs)`
12. `public String askToTurn()` Demande si le joueur veut tourner la pièce.
13. `public String askToDump()` Demande si le joueur veut défausser la pièce et passer son tour.
14. `public String askPlayers()` Demande le nombre de Joueur présent dans la partie.

15. `public String askBots()` Demande le nombre de Bots présent dans la partie.

16. `public String askName(int i)` Demande le nom du joueur à l'id i.

17. `public String askTuiles()` Demande le nombre de tuiles présent dans la partie.

18. `public void afficheActualPlayer(String name, int points)` Affiche le nombre de points du joueur.

3.2 Règles du jeu de domino

Toutes les règles sont présentes dans les deux versions du jeux, nous pouvons poser tourner une pièce gagner des points, passer notre tour, abandonner et gagner.

3.3 Règles de Carcassonne

Les règles de Carcassonne nous ont posé un problème au niveau du code avec le placement d'un pion à cause de la généricité de notre code et la manière dont on a construit et géré cela. Mais le placement d'une pièce est possible.

3.4 Interface graphique

Cette interface graphique ce sépare en 4 classes et deux JFrames

3.4.1 JFrame des Joueurs

Ce JFrame est la pour montrer la pièce courante du joueur, il est composé de 4 boutons qui sont : Piocher ; Pivoter ; Abandonner/Give Up ; et Passer. Chaque bouton porte le nom de son action

3.4.2 JFrame MainPage

Cette JFrame est là pour accueillir le plateau et les menus.

1. Le plateau Le plateau est assuré par la classe GridPlateau qui affichera tous les dominos/tuiles présentes sous forme de boutons et plus précisément d'un objet créer par nous qui étend le JButton.

2. Les Menus Le premier menu est simple et sous la forme de deux boutons propose de choisir les dominos ou Carcassonne. Dans les deux cas appuyer sur un bouton mènera vers un menu qui pour les dominos propose de configurer le nombre de joueurs, le nombre de bots set le nombre de dominos qu'on veut utiliser. Pour Carcassonne les mêmes choix sont proposés a la différence que le nombre de tuiles est fixé et inchangeable.

3.5 HAL 9000

1. `public boolean playerAction(Controleur c,Plateau p)` Reprend la fonction prise dans les fonctions gérant les tours, on créer un tableau tabPosable bidimensionnel de int, on test chaque piece libre du plateau pour tester si la piece piochée rentre dans une case et rentre le nombre de point de celle ci, si elle n'y rentre pas -1 est placé dans le tableau. On test ensuite si le tableau est remplis de -1 si oui il défausse la carte, si non il choisit la piece qui lui rapporte le plus de point et la pose ensuite.

2. `private void tournePiece(int i)` sert à tourner la piece i fois.

3. `private static boolean isEmpty(int[][] tab)` sert à regarder si le tableau est remplis de -1.

4. `private static int max2(int[][] tab)` sert à renvoyer l'indice i ou se trouve la piece qui rapporte le plus de point.

5. `private static int max(int[] tab)` sert à renvoyer l'indice j ou se trouve la piece qui rapporte le plus de point.