université PARIS DIDEROT PARIS 7

Projet POO-IG

Développement d'une famille de jeu Domino

YOUNES EL ARABI
ABD BARI ABASSI

Ecole d'ingénieur Denis Diderot
Informatique
Janvier 2019

Sommaire

- **Introduction**

1- Qu’est ce qu’un domino ?

Un domino est une pièce qui a deux faces , chaque face représente un numéro ou une image et/ou une couleur . Un domino a deux orientations possibles selon le cas d’association . On dit qu’un domino est double si les deux faces se ressemblent , dans ce cas là, la pièce peut être posé verticalement . Dans la suite du problème , on ne tiendra pas compte de ce paramètre pour simplifier le code .
Il est possible de créer plusieurs variantes de jeu en modifiant l’aspect graphique du domino ainsi que les règles d’associations .

2- Différences et points communs entre les quatres jeux :

L'objectif dans ce projet est de modéliser au mieux quatres jeux différents qui ont toutes des similarités avec le jeu du Domino .

Points communs :

1- Les pièces : pour les deux premiers jeux (Domino et Domino gommette) on a la possibilité d’utiliser les mêmes pièces qui changeront seulement dans la vue graphique , les règles d’associations sont presque les mêmes . On a également la possibilité d'utiliser la même grille , le même paquet de pièces et le même moteur de jeu .
2- On peut aussi s’en servir des dominos pour construire le jeu “saboteur” .
3- Tous les jeux ont besoin d’un ou plusieurs joueurs pour jouer aux jeux .
4- Tous les jeux requièrent un pack de pièces .

Différences :

1- On a remarqué qu’on n’aura pas besoin de pièces pour modéliser le jeu puzzle , en effet pour construire ce jeu on aura seulement besoin d’une configuration qu’on modélisera avec un tableau d’entiers , il s’ensuit qu’on utilisera pas la même grille du jeu domino/gommette . Il est toujours possible d’utiliser les pièces et la même grille , mais ça compliquera seulement la construction du jeu qui est normalement simple .

2- La construction d’une chaîne de dominos est différente de la construction d’une chaîne de jeu saboteur , on a donc choisi d’utiliser une grille différente pour le jeu saboteur .

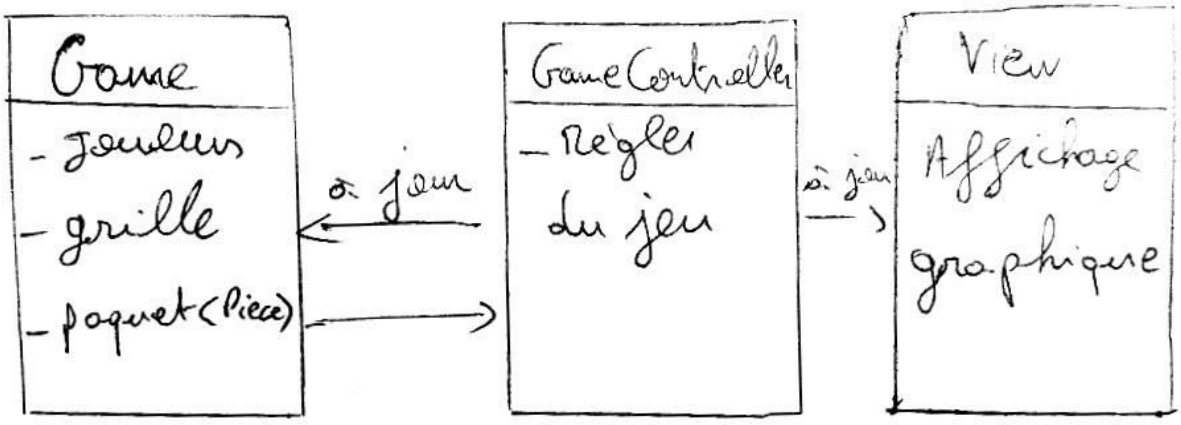
- **Modélisation :**

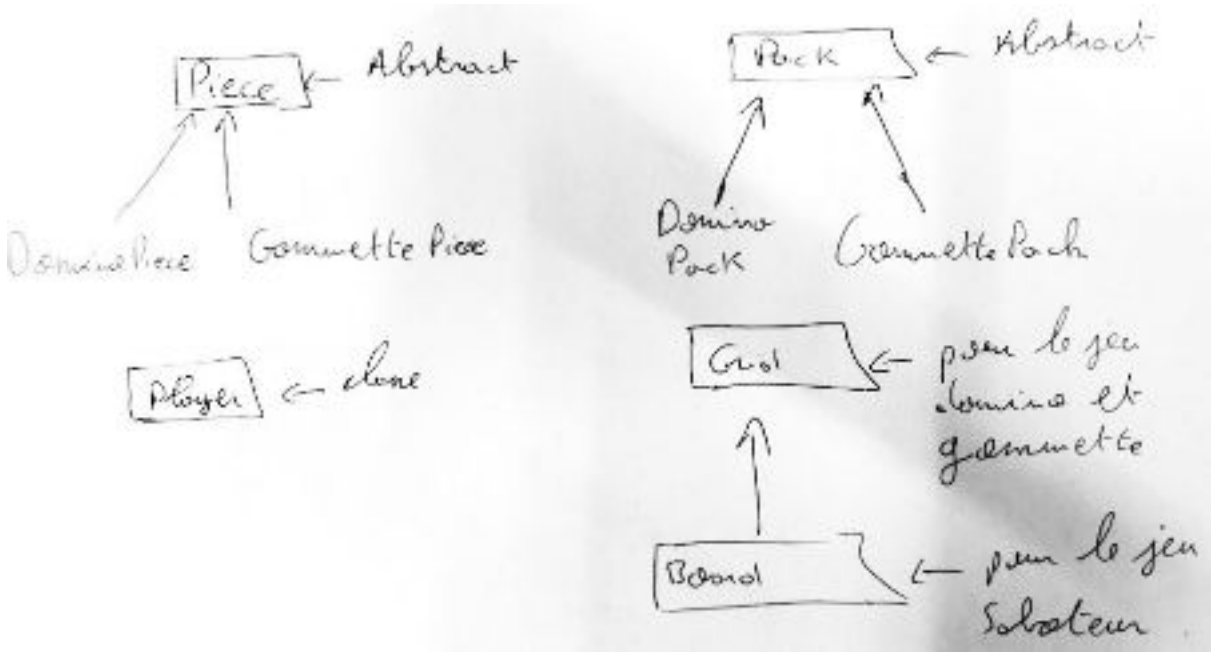
1- Architecture principale :

Premièrement , on tient à séparer la vue du modèle , c’est à dire d’adopter un design pattern MVC (model - view - controller) pour faciliter le développement futur des jeux sur différentes plateformes .
On construit un paquet “model” qui regroupe les différentes classes qui seront utilisées pour coder d’autres jeux : Pièce , player , pack , Grid . DominoPiece , DominoPack , GommettePiece , GommettePack , CarteChemin , CarteAction .

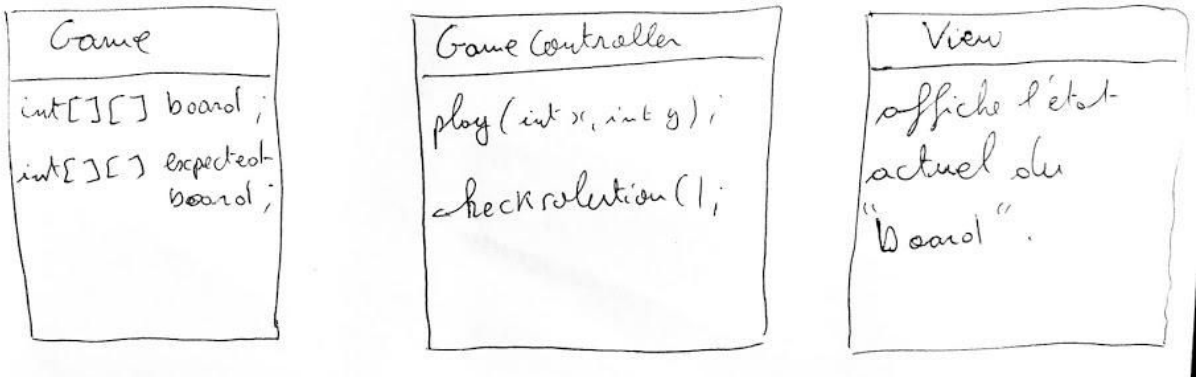
Le classe abstraite : Pièce , dont on hérite pour créer des DominoPiece et des GommettePiece et le pack des différents pièces crées .

La classe DominoGame a comme attributs des joueurs , un paquet de pièces et une grille , son constructeur prend en paramètre un paquet et une grille , cela nous permettra ensuite de créer différents jeux qui ont seulement des pièces différents comme pour le jeu “Gommette” .





Le jeu Puzzle ne demande pas une logique d'association de pièces ou même l'utilisation de pièces , on a donc réfléchi à une solution qui consiste à créer deux configuration de tableau d'entiers , une qui ne change pas et une deuxième qu'on mélange et que le joueur essaie de remettre à son état de départ .
Une seule case vide est présente sur la matrice qui nous permet de permuter une cellule vide avec sa voisine non vide .



Le jeu Saboteur est presque identique au jeu Domino , il suffit de redéfinir les méthodes d'associations et les méthodes des tours de jeu et de créer une grille adéquate.

2- Règles d’association :

On compare chaque côté de domino avec les deux côtés du domaine qu’on souhaite lui associer , si par exemple le côté droit du domino 1 correspond au côté droit du domino 2 on permute les deux côtés du domino 2 .
Les côtés des dominos sont des nombres comprises entre 0 et 6 .

```
public boolean tomatchRight(Piece p){
        //
        if (this.RightSide == p.getLeftSide() ) {
                return true;
        }
   if( this.RightSide == p.getRightSide()) {
                p.Flip();
                return true;
        }
   //
   return false;
}
public boolean tomatchLeft(Piece p) {
        //
        if (this.LeftSide == p.getLeftSide() ) {
                p.Flip();
                return true;
        }
   if( this.LeftSide == p.getRightSide()) {
                return true;
   return false;
}
```

Les deux faces des pièces gommettes sont des nombres à deux chiffres , le premier chiffre correspond a une couleur et le deuxième a une forme . Il suffit donc de redéfinir les méthodes d'association et d’utiliser le même moteur de jeu ainsi que la même classe graphique pour l’affichage .
On n’a pas réalisé l’affichage graphique du jeu Gommette car on a pas eu le temps de créer les images adéquates.

```
public boolean tomatchRight(Piece p) {

		if (this.RightSide/10 == p.getLeftSide()/10    || this.RightSide%10 == p.getLeftSide()%10 ) {
	return true;
	}
	   if (this.RightSide/10 == p.getRightSide()/10   || this.RightSide%10 == p.getRightSide()%10 ) {
			p.Flip();
			return true;
		}
	   //
	   return false;
	}

	@Override
	public boolean tomatchLeft(Piece p) {
		if (this.LeftSide/10 == p.getLeftSide()/10     ||  this.LeftSide%10 == p.getLeftSide()%10 ) {
			p.Flip();
			return true;
		}
	   if (this.LeftSide/10 == p.getRightSide()/10    || this.LeftSide%10 == p.getRightSide()%10) {
			return true;
		}
	   //
	   return false;
	}
```

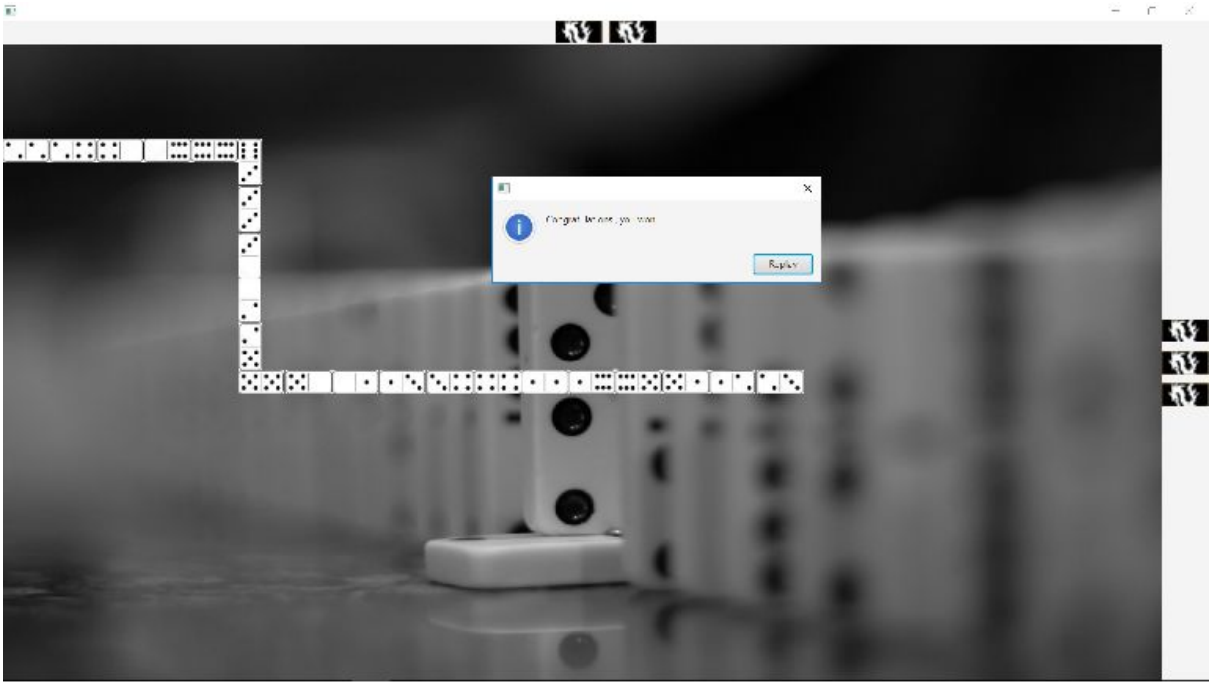
- **Design :**

1 - Modèle graphique :

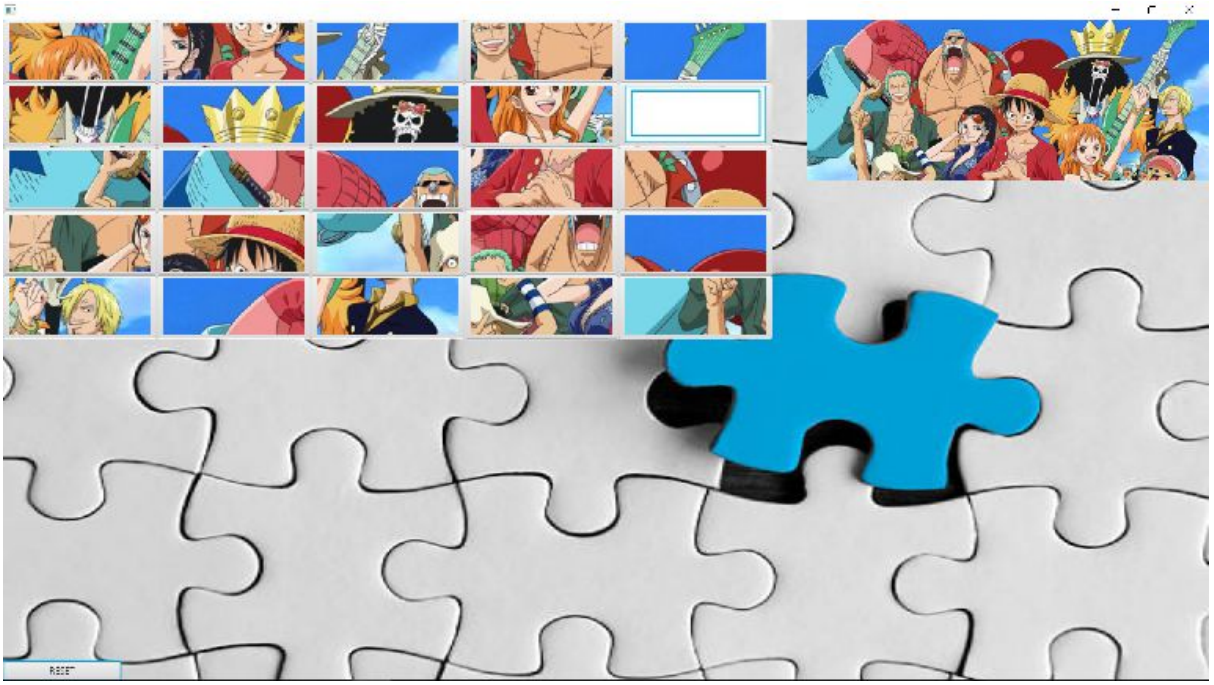
La même classe graphique du jeu domino peut être utilisé pour afficher le jeu gommette.

Le contrôleur enregistre dans différents listes les images de la grille , de la main des joueurs et du paquet de pièce , et en parallèle la classe vue reçoit ces images . A chaque tour de jeu ces images sont mises à jour par le contrôleur et la classe vue redessine le contenue . Il est clair qu'on respecte le modèle MVC puisque la vue n'a aucune information sur le déroulement d'un jeu mais affiche seulement un contenu graphique selon le jeu .

Pour construire une chaîne de dominos on a choisi d'utiliser une trajectoire préalablement définis. Les dominos joués suivront toujours la même trajectoire imposée . A chaque tour de jeu, on redessine la chaîne des dominos , l'affichage des dominos sera donc décalée d'une pièce a chaque jour .



Pour le puzzle on utilise une GridPane ou on insère des boutons avec les images adéquates de chaque élément de la grille de jeu .



2 - Entrées utilisateurs :

Dans les 3 jeux ou on a développée une interface graphique , les éléments à jouer sont représentés par des boutons indexé auquel on ajoute des Listeners . A chaque clique du joueur on récupère l'indice du bouton sélectionnée et on le passe au contrôleur qui s'en charge de traiter le tour de jeu . L'avantage est qu'on aura jamais une ArrayOutofBoundsExcepetion puisqu'on clique toujours sur un element qui existe dans la grille du jeu .

- ## Améliorations possibles :

1 - Compléter les méthodes qui permettent d'associés les pièces dans le jeu saboteur et implémenter une 'interface graphique .

2 - Définir une liste de joueurs au lieu de fixer dans la classe Game deux joueurs seulement .

3 - Développer une IA pour l'ordinateur afin de définir plusieurs niveaux de difficultés dans les jeux .

4 - Créer un système de score .

4 - Construire un modèle graphique qui s'adapte à tous les jeux .

Conclusion :

Le développement de plusieurs variantes du jeu domino n'est pas facile si on ne modélise pas correctement le problème . On peut améliorer encore mieux l'architecture de ce projet pour écrire un minimum de code .