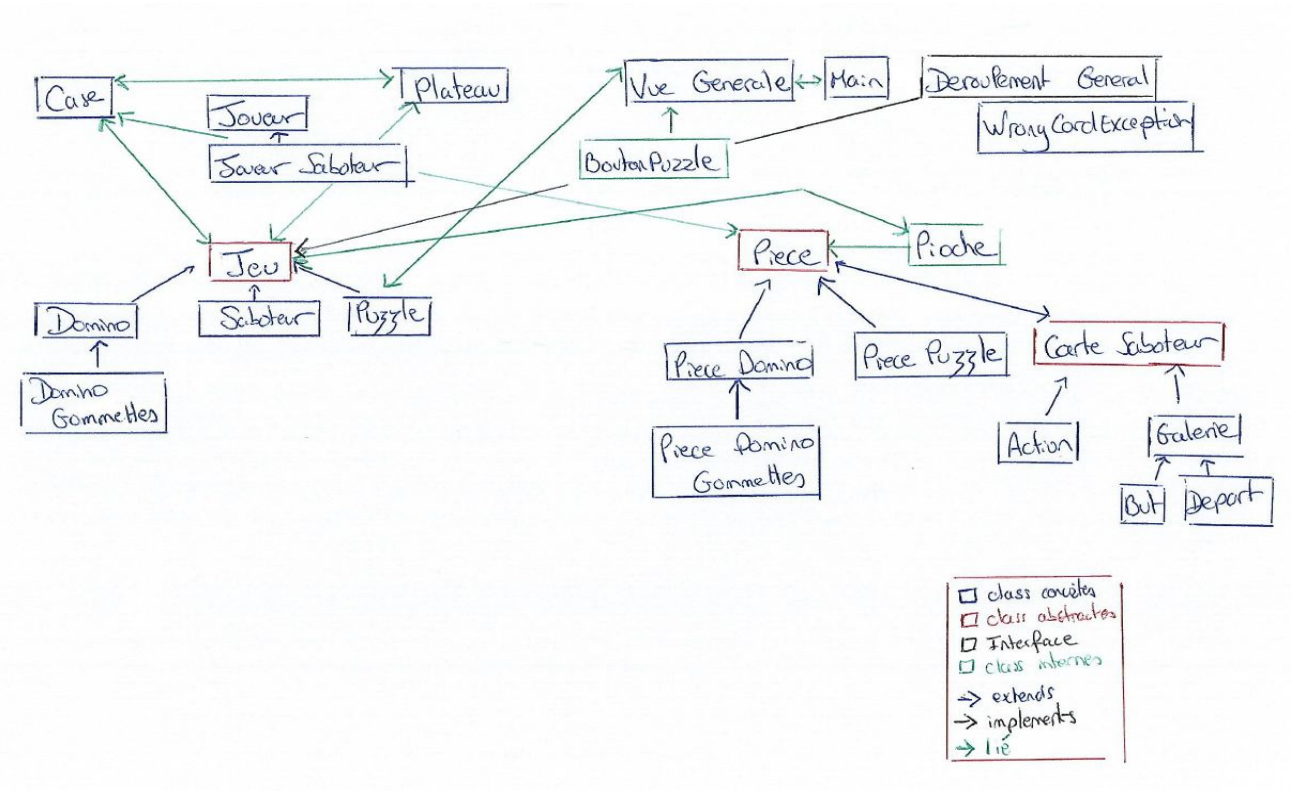
Fromont Léonard
Jallade Marie-France

M1LI
pour le 04/01/2019

Projet de POOIG

I - Parties traitées

a) Représentation de l'architecture du programme



b) Concepts utilisés et leurs limites

L'héritage

Les 4 jeux à réaliser héritent de la même classe abstraite Jeu, car ils possèdent des attributs communs (comme le plateau ou les joueurs). DominoGommettes hérite en plus de Domino parce que ce sont presque les mêmes jeux puisque toutes les méthodes de DominoGommettes sont des redéfinitions des méthodes de Domino. Nous avons choisi que l'héritage entre jeux ne concernerait que Domino et DominoGommettes, parce que le puzzle et le saboteur ne fonctionnent pas de la même manière, ce que nous détaillerons plus tard.

La classe abstraite

Nous en avons trois: Jeu, Piece et Carte Saboteur. Ces trois objets ne nous sont pas utiles en soit parce que ce sont exclusivement leurs classes filles qui sont utilisées, néanmoins

les différentes classes filles pour une de ces trois classes partagent des attributs et méthodes communes (tous les jeux possèdent un plateau et un scanner par exemple, de même que toutes les pièces possèdent un attribut nom).

La classe interface

Comme nous avons fait le choix de ne pas faire hériter tous les jeux de Domino, il nous fallait trouver un autre moyen de rendre compte de comportements identiques aux 4 jeux. C'est pourquoi les méthodes qui sont communes aux quatre jeux (la création des pièces et des joueurs, et le lancement du jeu) ont été réunies dans une interface implémentée par Jeu et donc par tous les jeux.

La généricité

Nous avons paramétré Jeu comme étant une classe générique prenant en paramètre le type de pièces compatibles avec le jeu crée. Grâce à ce paramètre, nous pouvons créer une LinkedList contenant ces pièces (c'est la pioche), ce qui nous évite de créer une pioche pour chaque jeu en spécifiant "à la main" le type des objets contenus dans la LinkedList.

La classe interne

Nous avons utilisé une classe interne dans l'encodage de nos jeux et plusieurs classes internes pour l'encodage de l'interface. Ainsi, Piece.Pioche est une classe interne à Piece. Nous avons fait ce choix parce que la pioche contient des objets de type Piece, elle est donc toujours lié à cette dernière sans pourtant être une de ses filles. Nous nous sommes demandés si Piece.Pioche devait être statique ou non. Finalement, nous avons choisi de la rendre statique afin que toutes les pièces soient liées à une seule et même pioche. De plus, étant donné que les méthodes et l'attribut dans Piece ne sont pas statiques (les méthodes sont des getters et setters pour l'attribut "nom" d'une pièce), la pioche ne peut pas y avoir accès, ce qui est avantageux pour nous parce qu'on ne voudrait pas penser que le nom de la pioche est le même que le nom d'une instance de Piece. Cette question n'est pas problématique pour tous les jeux, mais elle l'est pour le saboteur, jeu dans lequel les cartes n'ont pas toutes le même nom.

L'exception

Nous avons aussi utilisé une exception pour le saboteur lorsque le joueur voulait réaliser une action qui n'était pas possible étant donné le type de carte qu'il jouait.

Nous avons énuméré ci dessus les concepts utilisés pour rendre compte des similitudes entre les jeux, mais ces jeux sont assez différents entre eux pour avoir chacun leur particularités que nous allons évoquer ci après:

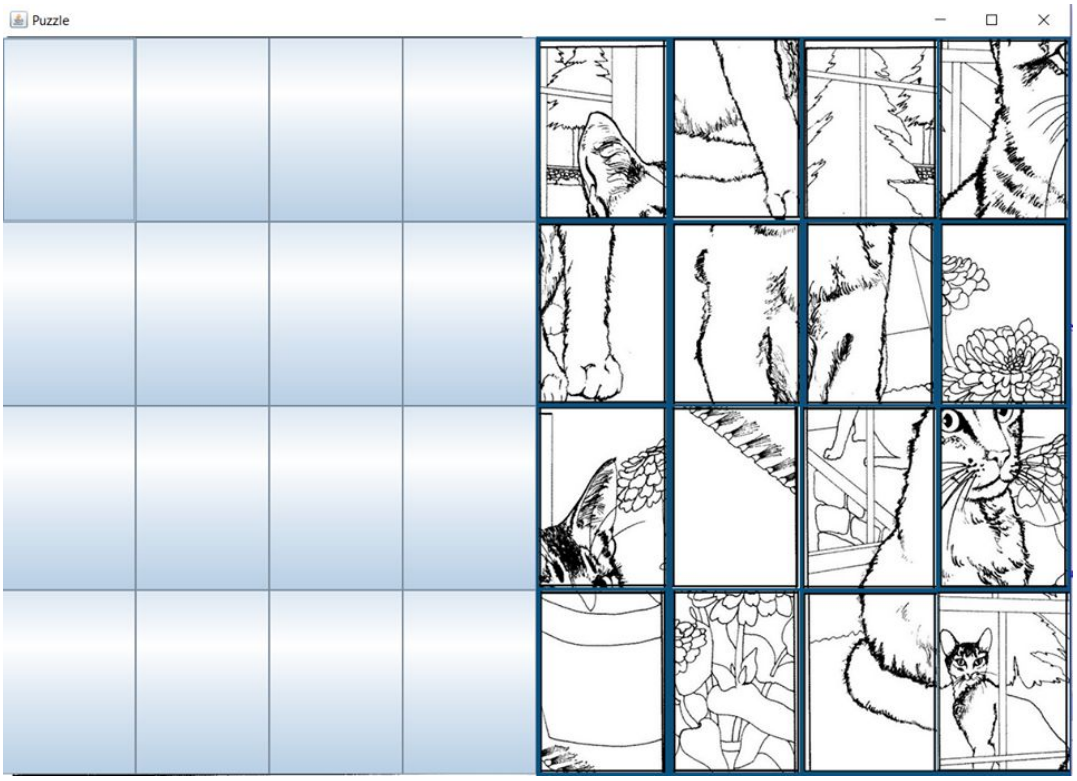
Pour le Domino gommettes, la différence avec sa classe mère réside dans l'encodage des pièces. Les pièces du domino contiennent deux informations (les deux int qui composent le domino) alors que les pièces du domino gommettes en contiennent quatre. En effet, pour

les deux côtés de la pièce, il nous faut savoir l'int qui correspond à la couleur ainsi que l'int qui correspond à la forme. Alors, ce ne sont pas les mêmes comparaisons qui sont faites pour vérifier le placement d'une pièce domino et d'une pièce domino gommette.

Pour le Puzzle, nous avons utilisé deux plateaux. Un plateau classique sur lequel le joueur pose ses pièces, qui est commun aux autres jeux, ainsi qu'un plateau invisible pour le joueur mais vital pour le bon déroulement du jeu. Ce plateau invisible est un plateau sous jacent et final sur lequel les pièces sont déjà posées correctement. Alors, il ne nous reste plus qu'à comparer la pièce posée sur une case par le joueur sur le plateau classique avec la pièce de cette même case sur le plateau "caché", qui correspond à la bonne case pour cette pièce.

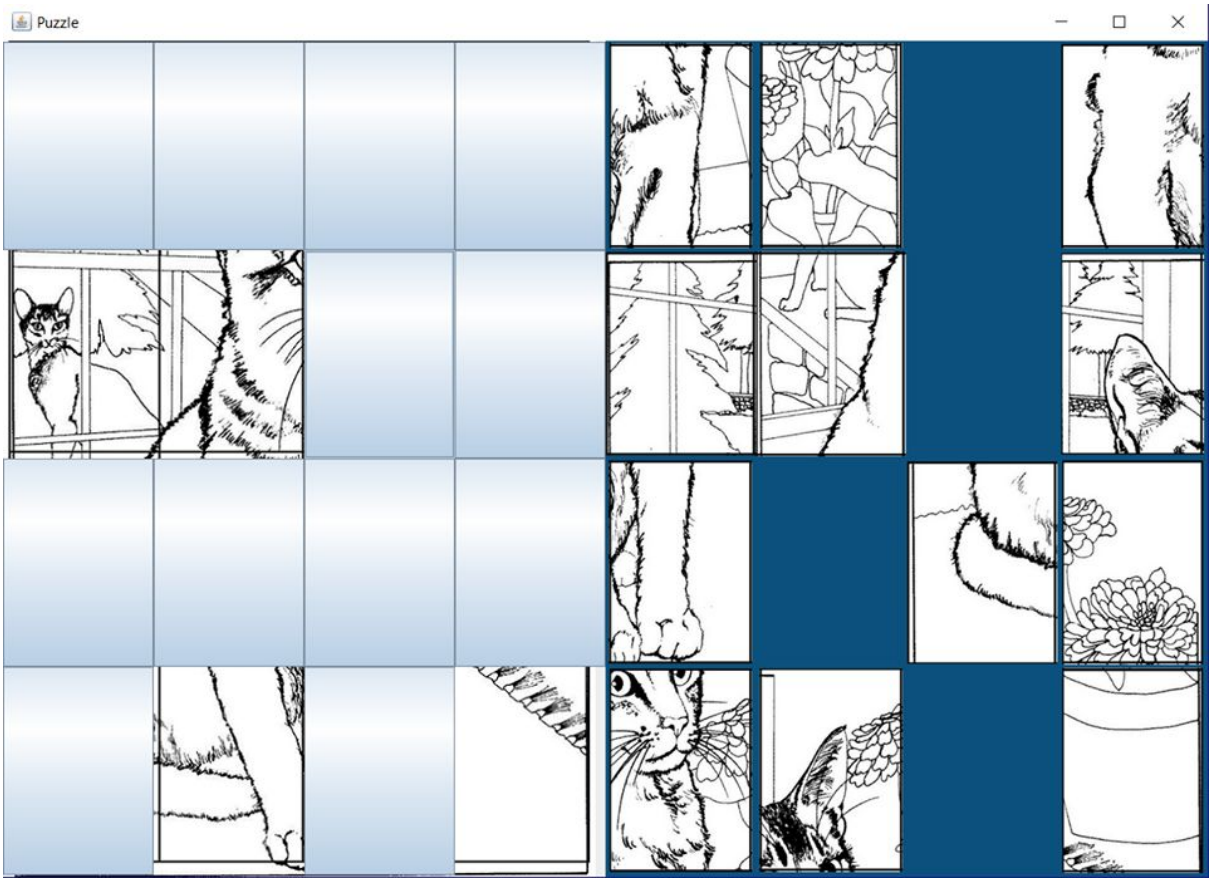
Pour le Saboteur, nous avons utilisé une LinkedList contenant la liste des chemins possibles entre le début et une des arrivées qui nous permet de vérifier si le joueur continue bien un chemin qui existe déjà (et ainsi éviter qu'il ne pose directement sa carte galerie à côté d'une des arrivées en déclarant victoire). Nous avons aussi utilisé une carte abstraite CarteSaboteur qui permettait de mettre en relation les 2 principaux types de cartes nécessaires pour notre jeu (les cartes actions et les cartes galeries). De plus, nous avons créé une classe JoueurSaboteur car un joueur saboteur possède des attributs que les joueurs des autres jeux ne possèdent pas (comme son identité secrète ou les cartes actions posées devant lui).

c) Le fonctionnement de l'interface graphique



Screen de l'interface au lancement du jeu

L’interface est séparée en deux parties. A gauche se trouve le puzzle complet qui est caché par des boutons, c’est le plateau. A droite se trouve la pioche, c’est-à-dire les pièces à mettre sur les bons boutons. Pour placer une pièce, il faut cliquer dessus puis cliquer sur le bouton qui correspond au bon emplacement. Si le joueur a cliqué sur le bon emplacement, le bouton qui cachait cette partie du puzzle disparaît du plateau et la pièce disparaît de la pioche. Cela se traduit dans le code par un “emprunt” de certains éléments Puzzle pour nous permettre la représentation de nos pièces dans notre fenêtre. Ainsi, nous avons créé un nouveau type de bouton qui hérite de JButton et qui a comme attribut une pièce du puzzle. Associer un bouton à une pièce nous permettait de comparer indirectement deux boutons en comparant leurs pièces respectives.



Screen de l’interface après le placement de quelques pièces

II - Problèmes connus

a) La combinaison généricité et héritage

Nous aurions aimé que Piece.Pioche soit un champ statique pour nos jeux car nous ne jouons qu'un seul jeu à la fois, mais nous n'avons pas réussi possible à cause de la généricité utilisé dans Pioche.

La généricité est aussi une des raisons importantes pour lesquelles nous n'avons pas fait hérité tous les jeux de Domino. Si on l'avait fait en gardant Jeu comme classe générique, on aurait perdu l'avantage de déclarer à la création du jeu le type de notre LinkedList Piece.Pioche et surtout de faire correspondre les pièces de la pioche avec le jeu. Ainsi, l'héritage de Domino par DominoGomette n'est pas dérangeant parce que PieceDominoGomette hérite de PieceDomino aussi: l'héritage entre jeu correspond à l'héritage entre pièces.

b) L'affichage des jeux et le choix du jeu sur lequel séparer le modèle de la vue

Nous avons du mal avec l'espace textuel qu'une case prend, étant donné les positions que prend prendre 1 seul domino. Alors, nous avons pensé qu'une case, c'est-à-dire un domino, prend la place d'un carré ou bien que l'affichage prendrait un tableau de tableaux de cases; mais l'implémentation d'un affichage comme cela aurait pris plus de temps que l'affichage final que nous avons choisi.

Au final, nous avons utilisé un tableau simple et collé les dominaux à la suite en mettant de côté l'affichage vertical des dominos.

Concernant les cartes galeries du saboteur, nous ne savions pas comment encoder le fait qu'un passage était ouvert et vers quelle direction. Finalement, nous avons choisis d'utiliser un tableau de taille 4 rempli de 1 (si le passage est ouvert) ou de 0. L'indice dans le tableau correspond au côté qui est concerné: le premier indice indique une ouverture vers le haut, le deuxième une ouverture vers la droite etc. dans le sens des aiguilles d'une montre. Par exemple, si un tableau représentant les passages ouverts pour une carte est comme ceci [1,0,0,1], il se traduit comme "cette carte a un passage ouvert qui mène vers le haut, et un qui mène vers la gauche".

Pour ce qui est du puzzle, on pourrait penser que l'affichage textuel relève un peu de la triche parce qu'il donne les réponses au joueur. En effet, la représentation textuelle d'une pièce est l'affichage de sa position correcte dans le puzzle, ce qui dit au joueur où mettre correctement la pièce. Afin de compenser cette triche nous avons choisi de faire l'interface graphique sur le puzzle pour qu'une partie de ce jeu soit plus réaliste.

c) L'interface graphique

Alors, nous avons créé une classe VueGenerale qui a entre autre un objet puzzle pour attribut, et qui est elle-même un attribut "vue" du puzzle.Cependant, nous n'avons pas su comment exécuter le déroulement du jeu textuel en même temps que le déroulement du jeu dans l'interface. En effet, les phénomènes qui déclenchent des actions (rentrer la position de la pièce à poser pour la représentation textuelle contre cliquer sur la pièce qui nous intéresse pour la représentation graphique) n'étaient pas compatibles pour jouer simultanément avec la représentation textuelle et la représentation graphique. Nous avons donc séparé les deux déroulements possible pour le puzzle, le déroulement choisi par le programme se fait en fonction de la représentation choisie par le joueur.

III - Pistes d'extensions

Nous aurions pu inclure les dominaux verticaux pour la représentation textuelle des jeux dominos; et faire des interfaces graphiques pour tous les jeux, ainsi qu'un menu au début permettant au joueur de choisir quel jeu jouer. Un autre aspect graphique à développer aurait été de permettre au joueur de choisir d'autres puzzle à jouer (en lui proposant d'autres images s'il aime pas notre puzzle chat), et plus généralement des thèmes pour sa fenêtre ainsi que l'affichage du chronomètre dans l'interface et l'affichage de l'image complète du puzzle dans un coin de l'interface afin que le jeu ne soit pas trop difficile.

Pour ce qui est des améliorations concernants les jeux, nous aurions peut-être pu exploiter l'identité des joueurs jouant aux saboteurs en essayant de faire en sorte que les joueurs qui sabotent puissent être découverts par les autres et ainsi être pénalisés, soit en étant éliminés, soit en donnant des avantages aux joueurs non-saboteurs. Aussi, on aurait pu créer une sorte de tournois regroupant tous les jeux, chaque jeu ayant des niveaux de difficultés qui rapportent plus ou moins de points selon le type de jeu et la difficulté choisie.