INTRODUCTION:

Ce projet de POOIG3, basé sur le jeu de société Azul et ces variantes, permet de mettre en pratique certaines notions vu en cours, TD et TP, ce qui m'aide à améliorer mes compétences et acquérir de nouvelles connaissance.

MODELISATION:

La modélisation de ce projet à été créer en structure de Packages Java, avec des classes parents et subclasse, le tout dans un même package appelé projet.

Liste des classes:

-Piece
-Jeux(interface)
-Jeu (implémentant Jeux)
-Joueur
-Plateau
-PlateauIncolor (héritant de Plateau)
-Case
-CaseMur (héritant de Case)
-VueGeneral
-Launcher (pour lancer le programme)
-TuileJoker (héritant de Piece)

Les classes se trouvant dans des sous package possède la ligne «import Projet.*;» ce qui leur permet d'appeler des fonctions et variable de classes se trouvant dans le package parent.

Les subclasses ont toutes un constructeur, celui-ci fait appelle au construteur de la classe parent et apporte quelque modification au variable de la subclasse.
L'algorthime de mon modèle ce base sur l'utilisation d'ArrayList et de Tableaux:

Pour toutes les variantes du jeu, chaque joueur possède un attribut Plateau et Pioche, Pioche étant une ArrayList<Piece> afin d'avoir un moyen d'augmenter ça taille et la réduire sans limite, et Plateau étant une classe ayant comme attribut les différentes partie d'un Plateau Azul:

-Un tableau de tableau de CaseMur[5][5] étant remplit avec des motif afin de représenter le mur.

-Un tableau de tableau de case[5][i] où i < 5(de sorte que case[0][0] 1 élément sur cette ligne, case[1][0] case[1][1] 2 élément sur cette ligne, etc..., voir image ci-dessous) représentant les lignes à remplir à chaque tour de manche.

-Un tableau de case[7] représentant le plancher.

Où Case est une cellule pouvant être null ou contenant une pièce, et CaseMur étant une case possédant une couleur.

J'ai séparer l'algorithme en plusieurs partie, la plus grande ce trouvant dans la classe Jeu, où y sont écrite les fonctions qui interagirent avec les différent éléments du jeu(Plateau des joueurs, sac, défausse etc)

En version textuel, la classe VueGeneral interagit avec les joueurs en utilisant un scanner, puis appelle les différentes fonctions de la classe Jeu avec les données fournit pas le joueur, tout en montrant l'état de la partie dans le terminal.

La fonction qui sera appeler par VueGeneral dépends toujours de ce que le joueur à écrit, VueGeneral vérifie aussi que les informations ne sont pas erronées, par
exemple, si le terminal demande au joueur de choisir sur quel lignes placé ces tuiles et que le joueur répond «Hello!» le programme le redemandera de fournir des données qui coordonnent avec la question demandé.

Pour facilité le lancement, la classe VueGeneral initialise le jeu en demandant via le terminal à quel jeu le joueur veut jouer, et créer le Jeu en fonction de la réponse.

Le jeu ce passe en trois étapes, la classe VueGeneral gère ces étapes les détails via le terminal.

Au début, on lance la fonction preparation, qui va remplir les fabriques avec les pièces du sac, si celui-ci est vide, on remplit le sac avec les pièces disponible dans la défausse jusqu'à c'que les fabriques soient remplit.

Si la défausse et le sac sont vide on arrête immédiatement de remplir les fabriques remplit ou non et on lance l'étape offre.

L'étape offre est la plus grosse partie du code, où les joueurs choisissent des pièces au centre ou dans les fabriques afin de remplir leurs lignes, la Vue demande au joueur de choisir et l'algorithme remplit sa pioche en fonction de son choix, le joueur ensuite choisie une lignes ou son plancher et l'algorithme vérifie si les pièces peuvent être posé sur les lignes en fonction des règles.

Lorsque le centre et les fabriquent sont tous vides on passe à l'étape de décoration, la Vue appelle une fonction deco de la classe Jeu tout en montrant les différentes étapes sur le

terminal, la fonction deco vérifie si une ligne est complète en utilisant des fonction de la classe Plateau, si c'est le cas on vérifie à l'aide des fonctions de Plateau si on est toujours dans les règles, et on décor le mur et ajoute au score du joueur les points correspondant à la situation courante de son mur.

Lorsque qu'un joueur a une ligne de mur complet, on arrête la phase de décoration et on applique les fonctions qui compte les points bonus, et on lance une autre fonction qui désignera le joueur gagnant, terminant la partie.

Manipuler autant d'ArrayLists et de Tableaux de tableau a créer énormément d'erreur de programme pendant le test des premières version du jeu, ce qui a rendu la correction de ces erreurs difficiles car le programme étant séparer en plusieurs fonctions, les messages d'erreur de terminal n'était pas évident à comprendre.

Pour implémenter les variantes, il aura suffit de modifier une ou deux lignes dans la classe Jeu, et créer des sous classes (PlateauIncolor et TuileJoker) afin de changer le comportement de des fonctions concerner en fonction de la classe sur la quel elles sont appelé.

Comment compiler et exécuter le code:

Lancer un terminal, placez celui-ci dans le Fichier "ProjetPOOIG3"

Lancer la ligne de commande: javac -d MesClass Projet/*.java

puis lancer la ligne de commande: Projet/Launcher puis suivez les instruction à l'écran, le programme vous demandera de choisir le jeu au quel vous voulez jouer.

(Un test rapide est écrit en commentaire dans le fichier Launcher si vous ne souhaitez pas testé avec 100 tuiles)

Liste des fonctionnalités demandées et réalisées:
-Jeu Azul
-Jeu Azul Mur Incolore

Liste des fonctionnalités demandées et non réalisées:
-Jeu Azul Joker
-Version graphique du Jeu Azul

Points fort du code:

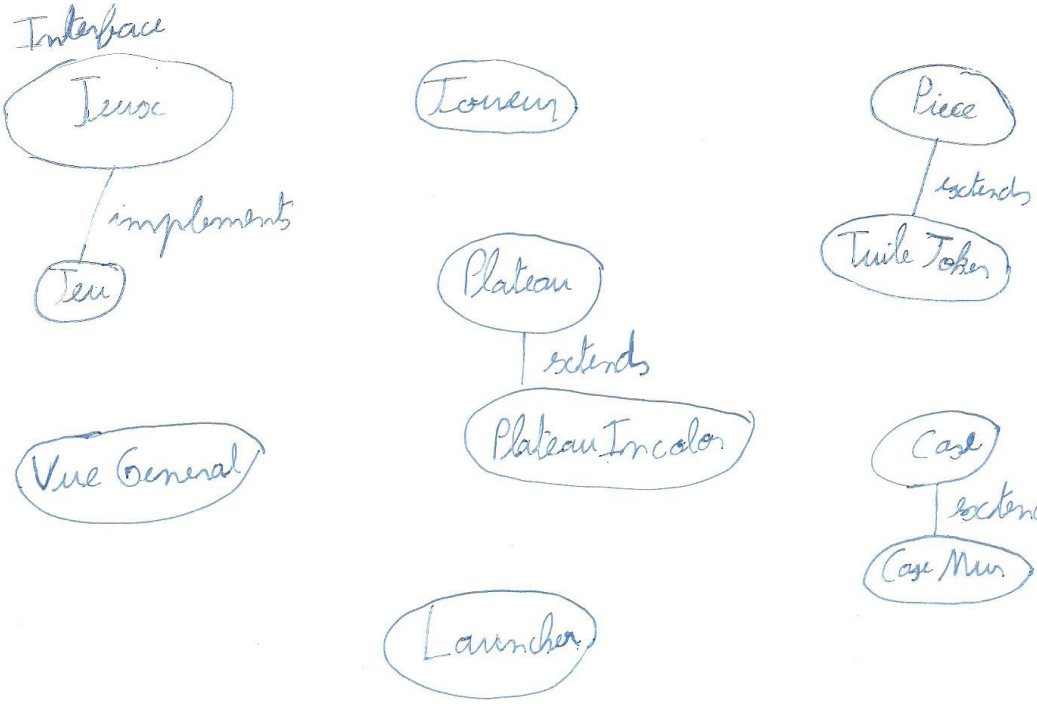
-Implémentation simple
-Code séparé en plusieurs partie, toute ayant des fonctions simplifiant l'utilisation des variables de ces classes
-Facilement réutilisable et modifiable
-Utilisation simplifié pour l'utilisateur, à l'aide de la Vue
-Prend en compte les mauvaise saisies afin d'évité toute sorte d'erreur de type «IndexOutOfBounds, NullPointerException etc»

Points faible:

-Code très long, avec des classes dépassant les 200 lignes

Précision sur la répartition du travail:

Moi (Fellou) et mon Binôme (Omar) devait se partager le travail, mais celui-ci ne m'a plus donnée de nouvelle après que lui avoir demandé par où il voulait commencer, j'ai donc fais ce projet tout seul.


Interface
Jeuse
implements
Jeu
Joueur
Piece
extends
Tuile Jaker
Plateau
extends
PlateauInCalor
Vue General
Case
extends
Case Mur
Launcher