Intro

Dans le cadre du 1er semestre de la deuxième année de licence d’informatique, il est demandé de faire un projet dans la l’UE programmation orientée objet. Ce projet consiste en la réalisation de deux jeux : un domino et un jeu de Carcassonne. Ceux-ci ayant une structure propice à utiliser l’ensemble du programme de la matière, à savoir, l’utilisation de classes, de l’héritage ou encore des interfaces graphiques.

1. Dominos

La création du domino se divise en trois étapes majeures :

1. Conceptualisation

Dans la conception du projet cette étape a été essentielle, elle consiste en la création conceptuelle du domino et des différentes classes qui le compose. La mise sur papier de ces idées permet de faire apparaitre une première vision du code et d’organiser le travail. Ce travail de conceptualisation est aussi utile dans la création du Carcassonne étant donné les similitudes des deux jeux notamment concernant les tuiles.

/\*insérer photo de faux papier qui nous aurait supposément servis\*/

On peut voir que l’organisation du code du dominos est déjà plus ou moins complète, il « suffit » maintenant de l’écrire.

1. Ecriture du code

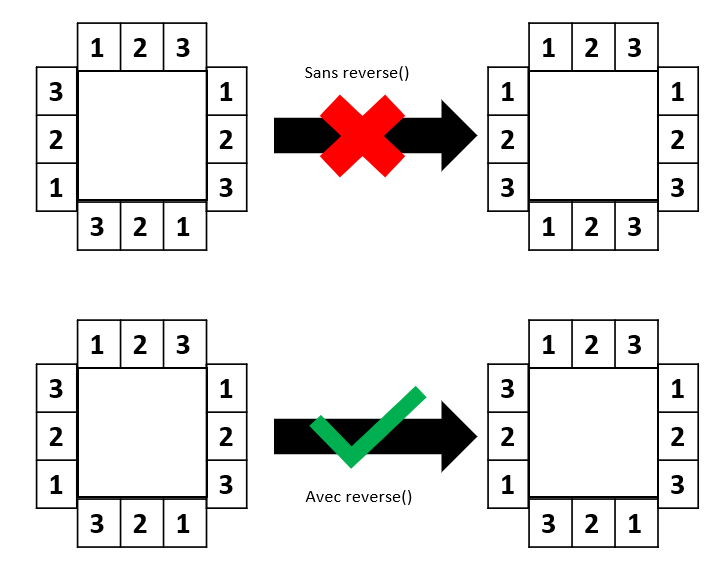
Ayant maintenant une assez bonne idée de la structure du code, l’écriture de celui-ci est divisée selon trois classes :

1. Trio

Le premier problème dans la création du domino a été de choisir si oui ou non une classe auxiliaire était utile pour définir les nombres contenus dans les tuiles du domino. On utilise la classe Trio pour définir les côtés des tuiles du domino. Elles possèdent une fonction **booelan estEgal()** qui renvoie true si deux Trio sont égaux, la méthode reverse qui échange la 1ère et la dernière case du Trio et la fonction **int sum()** qui renvoie la somme des trois nombres du Trio. Cette dernière méthode sera utilisée pour tourner les tuiles. L’implémentation de cette classe permet de ne pas avoir à manipuler les tableaux à chaque fois, elle a toutefois pour conséquence de charger un peu le code.

1. TuileDominos

Cette classe hérite de Tuile.java qui est une classe vide qui permet l’utilisation de certaines classes pour les deux jeux.

Elle possède 4 attributs Trio : haut, bas, gauche et droite. Ceux-ci représentent les quatre faces du domino. Les constructeurs sont assez simples et il y a une méthode **String toString()** qui permet de jouer dans le terminal. La méthode  **void tourneTuile()** permet de tourner le domino dans le sens des aiguilles d’une montre, c’est ici que la méthode reverse de Trio est très utile, pour que la Tuile tourne proprement il faut que les trios se comportent comme dans le schéma ci-dessous :

La fonction **boolean estCompatible()** est liée à la fonction **Tuile[] tuileAdjacentes()** du plateau. Celle-ci crée un tableau de tuiles qui contient les tuiles adjacentes et chacune d’entre elle a un index attribué. Dans **boolean estCompatible(),** les index servent à savoir quelle partie de la Tuile doit être vérifiée (haut, bas, gauche ou droite). Elle retourne un booléen qui vérifie, à l’aide de **boolean estEgal(),** si la tuile est compatible ou non.

1. Lanceur (il faut que je le finisse)
2. Interface graphique
3. Carcassonne
4. Ecriture du code
5. Interface graphique
6. Classes communes
7. Class Joueur

La classe joueur est commune aux deux jeux, elle contient 6 attributs. Il y a les attributs couleur et nom qui comme leur nom indique servent a rendre l’interface lisible en attribuant à chaque joueur un nom et une couleur. Cette classe sert essentiellement a enregistré des infos et a séparé la tour. Les quelques fonction qu’elles contient sont **piocher()** et **placerTuile()** dans lesquels on appelle simplement les fonctions du même nom de la classe plateau. Il y a aussi un attribut pion propre au carcassonne. L’attribut booléen bot permet, comme son nom l’indique, de choisir si oui ou non le joueur est joué par l’ordinateur. Enfin l’int score stock le score du joueur et la Tuile tuileCourante représente la tuile dans la main du joueur.

1. Class Plateau

La classe Plateau soulève plusieurs problèmes : comment faire pour qu’elle fonctionne avec le domino et avec le Carcassonne ? Comment faire la pioche du domino ? Comment faire un plateau dynamique ?

La classe Tuile permet de faire le lien entre le domino et le carcassonne, elle permet d’avoir une seule class plateau pour les deux types de tuile. De plus, l’utilisation de instanceof permet de faire des fonctions communes au deux jeux. La classe plateau possède dix attributs :

Une image contenant texte

Description générée automatiquement

Le constructeur de plateau prend deux arguments : un booléen isCarcassonne et un entier nbTuiles. L’entier nbCasesCote est initialisée a afin que dans le cas où toutes les tuiles sont placé en ligne, il n’y est pas de soucis. Il y a deux cas : Si isCarcassonne est true le tableau pioche se rempli des tuiles du carcassonne qui ont été manuellement entré. Dans l’autre cas, la pioche se remplit des tuiles du domino qui sont générée via la class Générateur dont le fonctionnement sera explicité plus tard. Le tableau des pions placé est rempli de -1 signifiant qu’aucun pion placée. Enfin, la fonction **void centrePlateau**() est appelée et celle-ci défini le point centrale du plateau.

Le tableau tuilesRestantes est un tableau de taille nbTuiles qui contient les nombres de 1 à nbTuiles. Dès que la i-ème tuile de la pioche est jouée ou défaussée, le i-ème élément de tuilesRestantes est remplacé par un -1. Lorsque le tableau n’est plus composé que de -1 la partie s’arrête, cela est vérifié par la fonction **boolean resteTuile().**

La fonction **Tuile[] tuilesAdjacentes()** prend des coordonnées x,y et regarde les quatres case adjacentes. Pour chaque cas non nul, le tableau se rempli de l’index 1 à 4 comme expliqué précédemment. La fonction **boolean placer()** prend en argument des coordonnées x,y, un Joueur j ainsi qu’une Tuile t qu’on veut placer. Le tableau de tuilAdj est de taille quatre et contient a chaque index soit une tuile soit null. Pour chaque case vide du tableau on incrémente cntVide qui sera utile dans la suite de la fonction. Si la case n’est pas vide, il y a deux cas : la tuile est une instanceof TuileDominos ou une instanceof Tuiles Carcassonne. Cette dichotomie permet de cast le type de tuile dans les deux cas afin que le programme sache quelle fonction **boolean estCompatible()** utilisé car les deux types de tuiles n’ont pas la même. Chaque test est stocké dans la variable compatibleAvecAll. Il y a deux cas où le placement d’une tuile est possible si le plateau est vide ou si il y a au moins une tuile adjacente au coordonnées x,y et que celle(s) est(sont) compatible(nt). Dans le cas où ce sont des TuileDominos, le score est compté et stocké dans le score du joueur en argument. Enfin, si les tests sont passé, la tuile est placé sur plateauActuel et la fonction renvoie true ; sinon rien ne se passe et la fonction renvoie false.

/\*Placer pion\*/

La fonction **Tuile piocher()** initialise un nombre aléatoire nb, si le nombre l’index nb de tuileRestantes est -1, elle return **piocher()**  et ce jusqu’à ce que le nombre a l’index nb soit différent de -1 (il n’y a jamais que des -1 car le jeu est arrêté avant si c’est la cas). Lorsque le test est passé, une Tuile temp est initialisée et il y a encore deux cas, carcassonne ou domino (avec instanceof).Une fois de plus, les deux cas ne diffère que dans le type des tuiles. On stock dans une variable p, du type voulu, le nb-ème élément de la pioche qui est cast a ce même type. Une nouvelle tuile, qui est une copie en profondeur de p, du type désiré est affecté temp. Enfin, le nb-ème élément de pioche devient null et celui de tuileRestantes est maintenant -1. Et on retourne temp.

Il reste les fonctions **void defausser()** et **String toString().** La première ajoute une tuile a la défausse et la seconde affiche le tableau dans le terminal de la manière suivante :

/\*Insérer image\*/

1. Lanceur (partie terminal)
2. Générateur
3. Piste d’extension et représentation graphique du modèle des classe

Conclusion