Eren-Mustafa Ceylan | Ihsane Errami | Louis Saly

Rapport de décision

Plateau de jeu

Le plateau de jeu sera un tableau 2d de 5x5. Ce tableau sera composé par des éléments de l'interface *Positionnable* (le château et les terrains).
Les Positionnable ont tous une couleur qui représente leur terrain (ex: Gris pour le château, Vert pour la prairie..)

Chaque case de ce tableau aura une fonction *isPosable()* qui nous indiquera si oui ou non la case respecte la contrainte de 5x5.

Le test pour le type du terrain, une fois la case choisie, sera effectué de sorte à ce que chaque terrain posé (les 2 de la tuile) soit testé, au moins une des deux terrains doit correspondre à un de leurs voisins.

Tuiles

Les tuiles sont composées de deux terrains (gauche et droite). Pour connaître leur placement dans le plateau, nous avons opté pour choisir un côté principal (qui est le gauche dans notre cas), puis lors du placement le joueur devra choisir une direction à sa tuile (nord, sud, est, ouest) ce qui nous permet de connaître la position du terrain de droite par la suite.

Modes de jeu

Les différents modes de jeu ont tout d'abord été modélisés avec le pattern Strategy.
Les modes de jeu devront implémenter GameStrategy qui aurait eu pour but d'implémenter le comptage des points selon le mode de jeu choisi.

Cependant nous nous sommes rendu compte par la suite, que pour combiner les modes de jeu, ce n'était pas la bonne solution et après réflexion, le pattern "Decorator" est celui adapté.

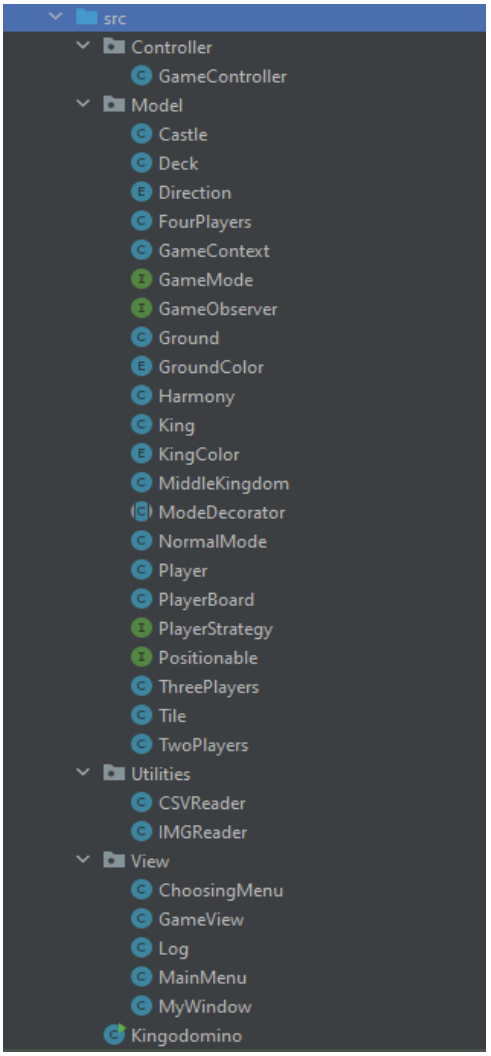
"ModeDecorator" possède un attribut pour référencer un objet emballé. L'attribut doit être déclaré avec le type de l'interface du composant afin de contenir à la fois le composant concret ("NormalMode") et les décorateurs. Le décorateur de base délègue toutes les opérations à l'objet emballé.
Cette méthode nous permet d'implémenter facilement de nouveaux modes de jeu et de les combiner entre elles.

Modèle MVC

Nous avons modélisé notre application en modèle MVC, c'est-à-dire un controller (GameController) qui va modifier le modèle lorsqu'un joueur va faire une action, une vue qui va transmettre au controller les demandes utilisateurs et afficher continuellement l'état du modèle.

Les vues ordonnent les créations des objets métiers au controller, qui lui, va ordonner à la classe maître (GameContext) la création des objets métiers.
Le controller ne crée pas lui-même les objets métiers car il y aurait une trop forte dépendance entre le modèle et le controller.

Voici comment sont disposées les classes en appliquant le modèle MVC :



Modèle maître

GameContext est notre modèle maître, il est donc chargé de réunir toutes les informations utiles à une partie, de créer les différents éléments du jeu sur demande du contrôleur.
Il a donc des relations de composition avec les différents éléments tels que la strategy du nombre de joueur, du mode de jeu ou de Deck. Cependant, Tile a une relation d'agrégation car la classe ne gère pas son cycle de vie, tout comme les observers qui sont donnés en paramètre dans la fonction addObserver().

Vues

Les vues sont toutes créées à partir d'une vue principale nommée MyWindow, qui va créer les différentes fenêtres comme le menu principal, le paramétrage de la partie ou encore le jeu. Cela permet une facilité de navigation dans les vues et laisse la possibilité d'implémenter d'autres vues dans l'application facilement.

MyWindow connait le controller et le modèle, elle retransmet l'état du modèle et transmet les demandes utilisateurs au controller.

Notre modèle GameContext contient des GameObserver (Interface), en utilisant ce patron de comportement, cela permet d'avoir des vues qui vont être en permanence notifiées lors d'un changement dans le modèle et qui vont mettre à jour leur rendu.

Contrôleur

Le contrôleur fait le lien entre le modèle et la vue, il récolte les demandes utilisateurs à partir de la vue, et les transmets en modifiant les modèles.
Le contrôleur connaît donc le modèle, et est connu par la vue.

Stratégie par nombre de joueur

La stratégie "PlayerStrategy" varie selon le nombre de joueurs pour une partie. Cette stratégie ne sera pas responsable de la création des Rois mais seulement de l'assignation de leurs couleurs et des tuiles en fonction du nombre de joueurs.

Cette décision a été prise car la stratégie aurait trop de responsabilités si c'était elle-même qui crée les Rois. Une autre solution a été discutée, celle que la stratégie devrait renvoyer une liste de couleurs, mais cela ne ressemblait pas à un comportement de stratégie (pas vraiment d'algorithme derrière).

Le fait d'utiliser une stratégie nous permet d'implémenter rapidement un nouveau nombre de joueurs qui auraient un comportement légèrement différent.

Tests Unitaires

Nous avons décidé de faire des tests unitaires afin de s'assurer le bon fonctionnement de notre application dès le début mais aussi plus tard lorsque nous aurons potentiellement changer quelques fonctions. Ces tests là imite différents scénarios de partie où l'on pose des tuiles sur un plateau.

Autres décisions

La fonction getColor() de l'enum GroundColor renvoie, à partir d'un String, une variable de type GroundColor. Cela a été validé que cette manière de faire était assez générique par le prof.

Nous avons voulu laisser au contrôleur de créer les éléments du jeu c'est-à-dire joueurs, plateau, tuiles, etc… par exemple une classe Deck qui créera les tuiles.
Cependant, après maintes réflexions et discussions avec le prof, si le contrôleur connaît tous les objets métiers alors il y aura un grand nombre de dépendances. Il faudrait alors qu'on crée une ou plusieurs classes ordonnées par le contrôleur (patron de structure - *Façade).*

Nous avons créé aussi des classes utilitaires comme CSVReader et IMGReader qui permettent de lire des fichiers ressources. Nous avons décidé de les mettre en tant que Modèle dans le MVC, préféré à l'idée de la séparer du MVC et d'en créer un nouveau package.

Exécutable

Nous avons un exécutable au nom de a31_kingdomino.jar qui se trouve dans le dossier out/artifacts/a31_kingdomino_jar.

Pour l’instant, notre exécutable affiche :

- le menu principal du jeu
- une fenêtre permettant à l’utilisateur de choisir son mode de jeu et le nombre de joueurs. (Cette partie n’est pas encore très complète car le choix des couleurs n’est pas réalisé encore)
- une dernière fenêtre affichant le jeu (cette partie affiche seulement les plateaux des joueurs et la première manche sans réel action possible)

Amélioration futur

- Ne pas laisser le Deck créer les tuiles mais plutôt la classe maître GameContext.
- Amélioration de l’interface de la page du mode de jeu.
- Modifier la fonction calculeScore() pour qu’elle puisse renvoyer la valeur juste du score du joueur.
- Ajouter le patron de structure facade pour décharger la classe maître et ne pas en avoir une avec trop de responsabilité.