Programmation orientée objet

Patchwork

BASTOS Antoine

ROUX—DAUGROIS Yann

02/12/22

Sommaire

Introduction....3
Contrôleur....3
Modèle....3
    Game....3
    GameBoard....3
    Les patchs....4
    Bonus de boutons, patchs spéciaux et tuiles spéciales....4
    Gestion de l’argent du jeu (buttons)....4
    XML....4
Vue....5
Schéma synthétique de l’architecture....6

Introduction

Nous avons décidé de mettre en place une architecture MVC de façon à séparer le code en modules répondant chacun à un des trois besoins du jeu pour reprendre la suggestion d'architecture, de l'exemple utilisant le bibliothèque Zen5, indiquée dans le sujet.

Contrôleur

Pour cette première partie du sujet l'interface sera seulement en ligne de commande, cependant on doit aussi anticiper les prochaines améliorations. Ainsi dans la mesure du possible on devrait pouvoir éviter de dupliquer du code en créant un contrôleur capable de prendre une quelconque interface. Ce n'est donc pas le contrôleur qui s'occupe véritablement des interactions utilisateurs mais il les récupères depuis l'interface donnée.

Modèle

Game

Une partie est représenté par son mode de jeu et son plateau de jeu.

Le plateau est créé à partir d'un fichier de configuration au format XML contenant les données utiles. Un simple fichier fait différer l'implantation de la version de base (phase 1) et la version complète (phase 2), car c'est seulement le contenu du plateau et non le fonctionnement qui est différent. De cette façon il n'est pas nécessaire de recompiler le projet pour changer les paramètres de jeu. On peut donc ajouter plusieurs joueurs, des événements ou des patchs.

Cette classe pourra dans le futur accueillir des méthodes pour sauvegarder une partie ou des scores. En attendant une méthode statique permet de créer une nouvelle partie pour un mode de jeu donné.

GameBoard

La classe *GameBoard* chapeaute les composants du jeu patchwork. On y réalise la gestion du tour des joueurs comme une pile car c'est le premier joueur à la position la plus la plus en retard qui joue. Pour clarifier la gestion des patchs du plateau (autour du plateau dans le jeu de société) on utilise une classe *PatchManager.*

Les patchs

Un patch est représenté par ensemble de coordonnées relatives et possède aussi une origine absolue. Ce qui permet de gérer sa position dans l'espace et ses rotations. L'ensemble de ses rotations est calculé à sa création.

Bonus de boutons, patchs spéciaux et tuiles spéciales

Les bonus de boutons, les patchs spéciaux et les tuiles spéciales sont attribués au joueur par le biais d'actions déclenchables par un événement. L'événement peut être positionné sur une case ou s'effectuer comme une routine à chaque tour, et être désactivé s'il est à usage unique. Ses actions sont stockées dans leur événement sous forme de lambda de type *Predicate* indiquant si l'action a été déclenchée ou non.

Gestion de l'argent du jeu (buttons)

Au départ dans notre implémentation, les objets qui le nécessitaient, possédaient un champs buttons. Mais afin de factoriser le code, nous avons utilisé de l'héritage. Les classes *GameBoard* et *Player* héritaient d'une classe abstraite *ButtonOwner*. Car on souhaitait avoir une API pour définir une circuit fermé de la monnaie en jeu. Cependant en raison des désavantages de l'héritage, évoqué dans le cours, nous avons utilisé une Interface et de la délégation. Les objets pouvant s'échanger des boutons implantent l'interface *ButtonOwner* et possèdent une *ButtonBank* qui de la même manière implante *ButtonOwner*. Autrement, les objets ayant une 'valeur' en boutons implantent l'interface ButtonValued. C'est de cette manière que ce caractérise la circulation de l'argent en jeu ; par des propriétaires de boutons pouvant s'échanger de l'argent et acheter des objets ayant une valeur.

XML

Dans le doute que l'utilisation du parseur XML fournit par Java soit autorisé dans le projet et pour des raison de simplicité, un XMLParser et un XMLElement ont été implanté pour accéder aux données des fichiers de paramètres. Le plateau de jeu est donc construit depuis un objet XML récursivement en utilisant le même fonctionnement sur les composants utiles.

Vue

Pour cette phase 2 seule l'interface CommandeLineInterface à été implantée. Les objets affichables du jeu implantent *Drawable* et DrawableOnCLI. De cette manière depuis le contrôleur on peut demander d'afficher l'objet pour une quelconque interface utilisateur, et c'est au final cette dernière qui utilisera la méthode adéquate, *drawOnCLI* pour *CommandeLineInterface* ou bien, lors de la phase suivante, *drawOnGUI*.

Comme pour reprendre le fonctionnement d'un affichage graphique la vue en ligne de commande possède un *StringBuilder* faisant office de fenêtre. C'est seulement lorsqu'on appelle la fonction d'affichage *display ou clear* qu'on affiche les éléments ou que l'on efface la fenêtre. Les interfaces utilisateur auront donc la même API. L'API de l'interface utilisateur permet aussi de récupérer des choix utilisateurs indépendamment de l'implantation choisie (UI/GUI).

Schéma synthétique de l’architecture

Voici un diagramme UML assez synthétique, représentant dans les grandes lignes notre implantation.