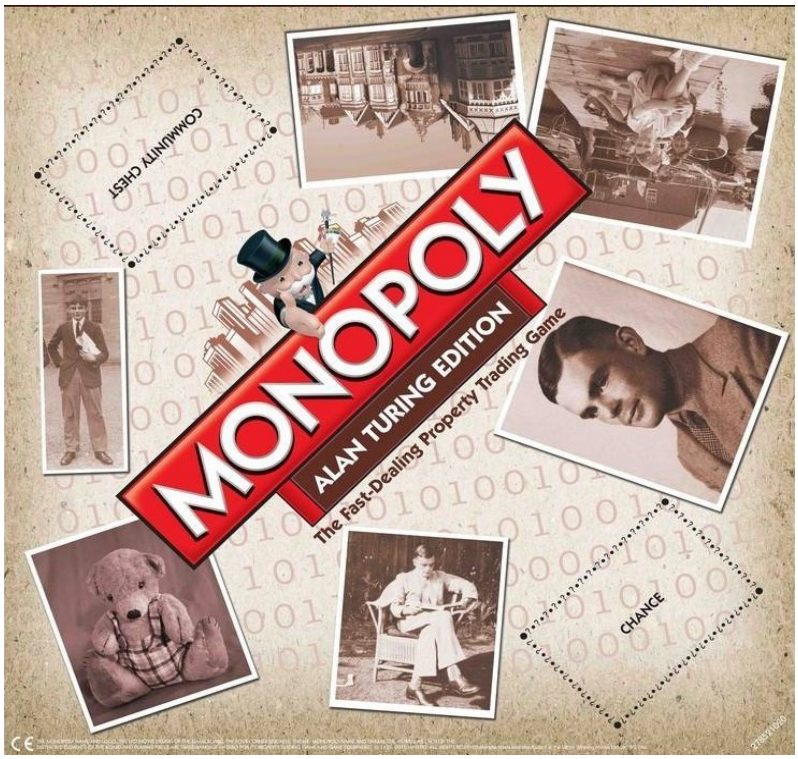
Objets et développement d'applications

MONOPOLY



Décembre 2012

Table des matières

Introduction

L'objectif de ce projet est de développer une application JAVA en utilisant quatre design patterns vus en cours. Pour ce faire, nous avons choisi de développer un Monopoly implémentant la majorité des règles du Monopoly original. Nous décrirons dans ce rapport le fonctionnement de celui-ci, puis détaillerons les quatre design patterns utilisés.

I) Mode d'emploi

A) Lancement du programme

Au lancement du programme, les élements du jeu sont instanciés afin de créer toutes les cases du plateau, le plateau lui-même et toutes les cartes chance et caisse de communauté. Il demande ensuite aux joueurs participant d'entrer leur nom et de choisir un des quatre pions à leur disposition. La partie peut alors commencer, les joueurs joueront dans l'ordre dans lequel ils se sont inscrits.

Différence avec le Monopoly original : seuls quatre joueurs peuvent participer simultanément, quatre pions étant disponibles.

B) Déroulement d'un tour

Au début d'un tour, le joueur actif est invité à lancer les dés en cliquant sur le bouton Jouer, l'action de la case sur laquelle il atterrit est ensuite automatiquement effectuée (payer un loyer, proposer d'acheter une propriété libre, tirer une carte, etc...). S'il a alors la possibilité de construire des maisons sur une couleur qu'il possède, le bouton correspondant devient accessible. S'il a fait un double, le bouton Jouer restera accessible afin qu'il puisse relancer les dés. Mais attention, trois doubles de suite l'entraîneront en prison. Lorsque le joueur ne peut plus entreprendre d'action, il peut alors cliquer sur le bouton « Passer la main » pour finir son tour. A tout moment, il peut cliquer sur son nom pour afficher une liste de toutes les propriétés qu'il possède, avec leur couleur et leurs loyers.

Si un joueur est en prison, il peut dès son deuxième tour derrière les barreaux lancer les dés, qui le libèreront s'il fait un double, payer une caution pour se libérer ou utiliser une carte « Vous êtes libéré de prison ».

Différences avec le Monopoly original :

- Les maisons se construisent simultanément sur toutes les propriétés d'une couleur. Par exemple, sur le quartier jaune, comportant 3 propriétés, le joueur ne peut construire que s'il dispose d'assez d'argent pour construire trois maisons.
- Les accords entre joueurs ne sont pas possibles, leur interactivité étant considérablement diminuée par le partage du même écran.
- L'hypothèque ne peut pas être entreprise à tout moment, elle est n'est effectuée que lorsqu'un joueur ne peut pas s'acquitter d'une dette, et cela automatiquement.

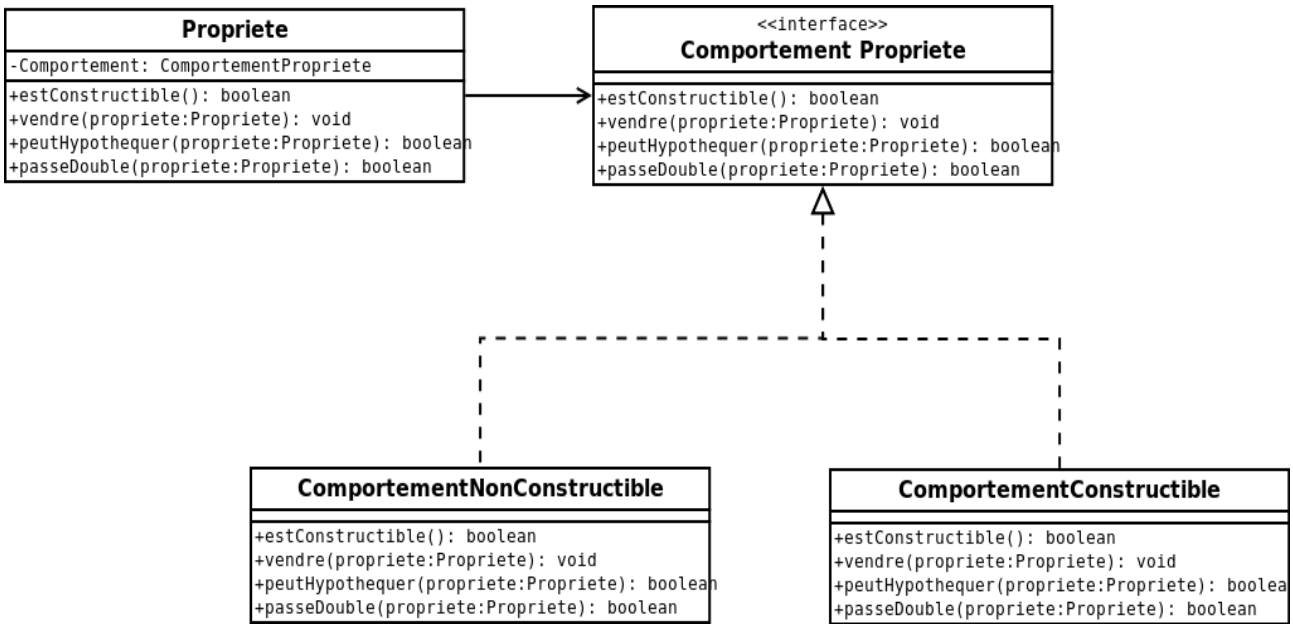
C) Fin d'une partie

La partie s'achève lorsqu'il ne reste qu'un seul joueur en jeu, c'est-à-dire lorsque tous les autres ont été éliminés. L'élimination d'un joueur peut se produire lorsqu'il doit payer une taxe ou un loyer qu'il n'est pas en mesure de fournir. Le programme appelle alors la methode actionDesesperee pour tenter de sauver ce joueur. Cette méthode vend automatiquement les maisons construites par le joueur jusqu'à ce qu'il ait assez d'argent. Si ce n'est pas le cas et que le joueur n'a pas de maison à vendre, ses propriétés sont successivement hypothéquées pour lui faire gagner de l'argent supplémentaire. Si, même au terme de ces actions, le joueur ne peut pas s'acquitter de sa dette, il est éliminé, et toutes les propriétés en sa possession reviennent sur le marché dans l'état libre qu'elles avaient au début de la partie, devenant ainsi disponible à l'achat pour les joueurs restants.

Différences avec le Monopoly original :
- L'appel à actionDesesperee est entièrement automatique et c'est l'unique moyen d'hypothequer ses propriétés et de vendre ses maisons. Par conséquent, le joueur ne choisit pas quelles maisons il vend ou propriété il hypothèque, cela est effectué en commençant par le moins lucratif et jusqu'au plus avantageux. Le joueur ne peut donc pas se prémunir contre la faillite en engageant manuellement ces actions, et doit faire confiance au programme pour vendre si cela est nécessaire.
- Les propriétés ayant été hypothéquées par actionDesesperee peuvent être rachetées par le joueur qui les possède pour leur prix d'hypothèque majoré de 10%. Cela sera proposé au joueur s'il atterrit lors de son parcours sur une propriété hypothéquée lui appartenant.

II) Patterns

A) Strategy



Lors de la création des propriétés, nous avons dès le début remarqué une différence démarquant trois catégories par leur loyer :

-Une rue est catégorisée comme un terrain constructible et son loyer varie en fonction de l'appartenance de toutes les rues de sa couleur à un même propriétaire. Son loyer peut aussi augmenter si le propriétaire y construit des maisons ou un hôtel.

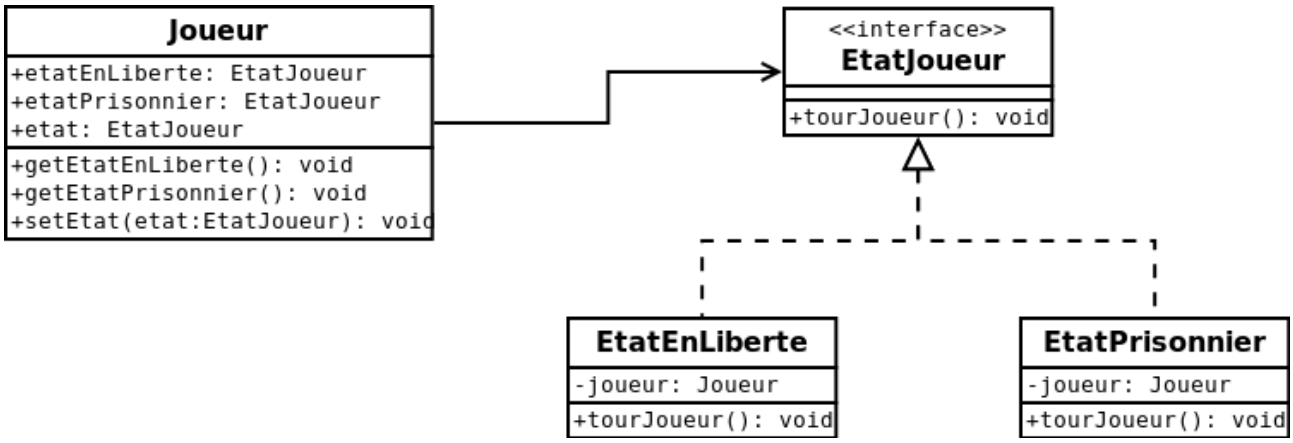
-Une gare voit son loyer augmenter proportionellement au nombre de gares possédées par son propriétaire.

-Une compagnie fixe son loyer en utilisant le lancé de dé du joueur qui atterrit dessus. Sa valeur est multipliée par 4 si son propriétaire ne possède qu'une compagnie, ou par 10 si son propriétaire possède les deux.

Afin d'éviter les structures conditionnelles interminables, faciliter la modification du code et mieux structurer notre application, le choix d'un pattern Strategy s'est imposé pour permettre de différencier les terrains constructibles des terrains non constructibles. Ainsi, les gares et compagnies ont été rassemblées par un ComportementNonConstructible et les rues ont été isolées avec un ComportementConstructible.

Un souci demeurait cependant : les maisons des propriétés constructibles. Un pattern State est donc utilisé pour le résoudre. En revanche, les deux patterns étant étroitement liés, Strategy est également utilisé comme test lorsqu'on cherche à vérifier qu'une Propriété est une Rue, afin de savoir si l'on peut y construire une maison.

B) State



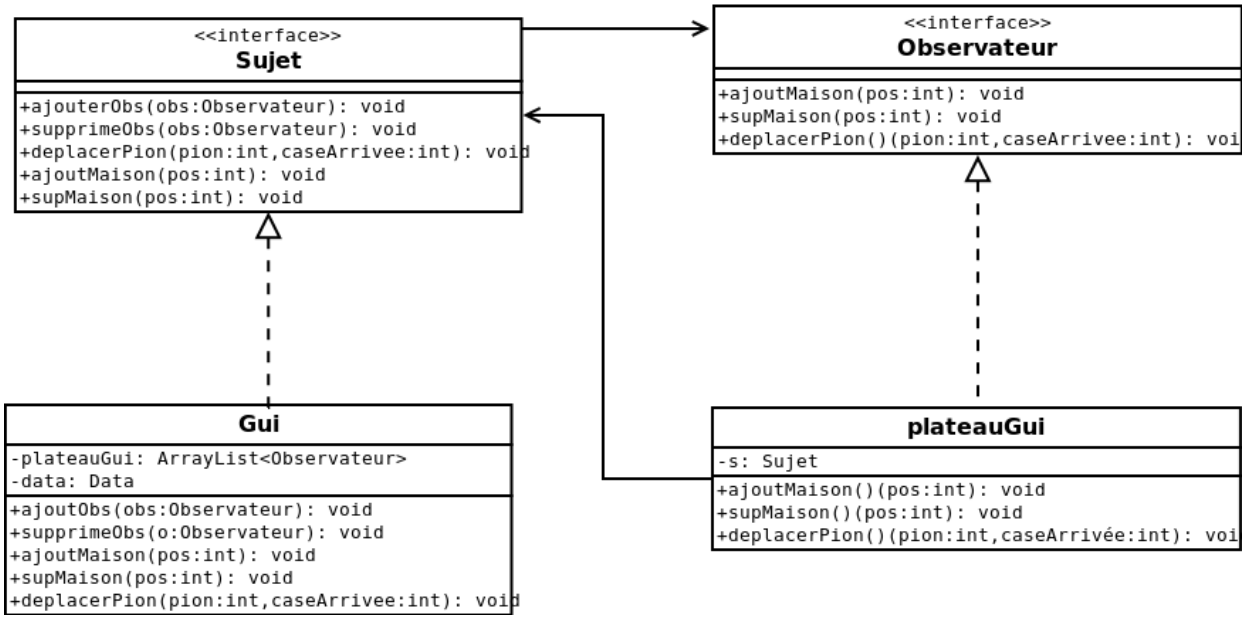
Ce programme implémente trois instances du pattern State vu en cours. L'exemple choisi ci-dessus est celui de la gestion de l'état d'un joueur, à savoir s'il est en prison ou non. En effet, le déroulement de son tour en sera grandement affecté. Durant la partie, un Joueur aura donc un EtatJoueur, qui sera un EtatPrisonnier s'il est en prison et un EtatEnLiberte s'il est libre. A chaque fois qu'un joueur est envoyé en prison par l'effet de la case « Allez en prison », l'effet d'une carte ou trois doubles d'affilée aux dés, son EtatJoueur change, et vice-versa lorsqu'il est libéré.

Chacun de ces EtatJoueur comprend une méthode tourJoueur() qui définit les actions à effectuer lorsque vient le tour du joueur, c'est-à-dire s'il est libre continuer son parcours et effectuer l'action de la case où il atterrit, et s'il est en prison essayer d'en sortir en payant une amende, attendant trois tours ou utilisant une carte « Vous êtes libéré de prison ».

Ce pattern est également utilisé afin de gérer l'état d'une propriété, à savoir si elle est libre, achetée ou hypothéquée. Cela affecte l'action effectuée par un joueur qui atterrirait dessus à la suite de son déplacement.

Ce pattern apparaît également afin de déterminer si une rue achetée est simple, si la couleur à laquelle elle appartient est entièrement possédée par un unique propriétaire ou si il y a une, deux, trois, quatre maisons ou un hôtel dessus. Cela influe sur le loyer qu'un joueur concurrent paiera en atterrissant dessus et également sur les actions autorisées ou interdites à son propriétaire (notamment à savoir s'il a le droit d'y construire une maison supplémentaire ou un hôtel, et si elle peut être hypothéquée).

C) Observer



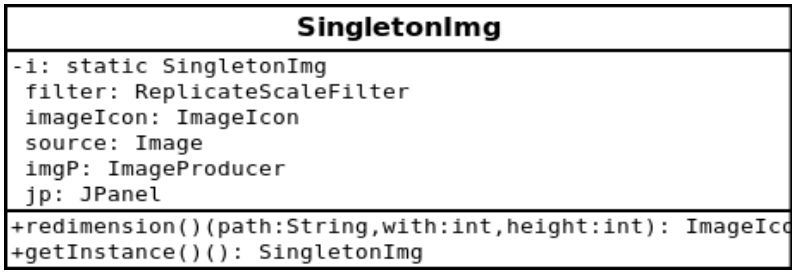
Le pattern Observer gère une partie de l'affichage graphique, il correspond au plateau de Monopoly avec les cases, les pions et les maisons. Le pattern Observer a été implémenté pour permettre des modifications simples du plateau telles qu'une autre édition, ou encore implémenter un affichage console, voire même un affichage 3D utilisant une nouvelle bibliothèque ou bien les trois en même temps !

La classe Gui implémente l'interface Sujet, elle contient des méthodes d'ajouts et de suppression d'observateurs ainsi qu'une méthode « actualiser », répartie sous la forme de trois méthodes différentes par souci d'optimisation de l'actualisation graphique. Ces méthodes sont le déplacement d'un pion et l'ajout/suppression d'une maison. En effet, le pattern Observer est de type "Pousser" pour les raisons évoquées ci-dessus. Il contient donc une ArrayList d'observateurs mais également de nombreuses autres données tels que les boutons, le menu et un pointeur sur la partie de Monopoly en cours.

La classe plateauGui implémente l'interface Observateur, elle définit les fonctions de déplacement de pion, d'ajout et de suppression de maisons ainsi que le sujet. La méthode employée consiste à pousser l'information dans l'observateur pour éviter qu'il ne réactualise l'ensemble du plateau inutilement.

Le plateauGui étant également l'affichage graphique, il contient l'ensemble des classes permettant l'affichage tel qu'un tableau de pions, l'image centrale du plateau et une liste de terrains. Ces terrains sont décomposés en deux parties : l'image de fond et une grille qui définira l'emplacement des maisons construites sur la case et des pions la visitant.

D) Singleton



L'affichage graphique nécessite un ensemble d'images qui peuvent être de tailles et types différents. Le traitement de ces images avec la bibliothèque Swing est complexe et lourd car il est nécessaire de posséder une image source, un filtre, une image modifiée et une image résultat. En supposant que la définition des images soit haute, la mémoire est fortement utilisée sans raison.

C'est pour cette raison que nous avons choisi de n'autoriser qu'une seule instance de la classe TransformationImage grace au pattern Singleton. De plus, celui-ci doit être unique car, dans le cas contraire, l'accès au fichier contenant l'image en lecture par plusieurs instances retournerait une erreur.

La classe est facilement extensible, actuellement elle permet de redimensionner une image en passant en paramètre le chemin de l'image, la hauteur et la largeur souhaitées et retourne une image modifiée.

Conclusion

Ce projet illustre l'importance des design patterns dans la programmation d'une application impliquant un grand nombre d'objets, qui serait chaotique et très laborieuse à modifier en leur absence. Il s'agit du premier projet universitaire constituant une véritable application que nous développons, la souplesse de l'énoncé et le temps à notre disposition nous ayant permis de concevoir un projet qui nous intéressait, et d'effectuer nos propres choix lors de sa programmation.

Nous aurions aimé offrir au joueur de plus nombreuses possibilités s'approchant davantage des règles du Monopoly original, notamment au niveau des interactions entre les joueurs, et lui permettre de jouer seul contre une intelligence artificielle optant pour des choix prédéfinis selon des stratégies basiques. Il aurait été également envisageable avec plus de temps et de connaissances d'adapter cette application pour un portage sur plateforme Android ou offrir la possibilité de jouer en réseau sur des écrans différents.

ANNEXE : Diagramme des classes

Ce diagramme n'a pas vocation à être un UML, il constitue plutôt une carte du programme, afin de clarifier les relations entre les nombreuses classes.