**Exercice Les patterns du GoF**

**Evolution du modèle du Monopoly mettant en œuvre des patterns du GoF**

**( Application de patterns du GoF sur le FS )**

Partie I : L’application jeu de Monopoly doit être organisé de façon à proposer des composants réutilisables.



## Exercice 1 : Déplacement des joueurs sur le plateau de Monopoly

### Problème :

Chaque joueur doit pouvoir se déplacer sur le plateau de jeu. A partir de la case courante sur laquelle le joueur se trouve, celui-ci doit pouvoir accéder à la case suivante et ainsi de suite pour parcourir le plateau jusqu’à arriver à sa case de destination, en fonction de la valeur des dés qu’il a lancés. Les règles suivantes sont à prendre en compte.

1. Lorsqu’un joueur arrive sur sa case de destination, une règle de jeu dite « d’arrivée » doit être exécutée en fonction du type de case concerné, par exemple :
   1. Case « Taxe de Luxe » : le joueur doit payer la taxe indiquée sur la case.
   2. Case « Aller en Prison » : le joueur va directement sur la case Prison.
2. Le fait de passer sur certaines cases peut également entraîner l’exécution d’une règle dite « de passage ». C’est le cas de la case « Départ », qui permet à un joueur de recevoir 200 € lorsque celui-ci passe dessus.

### Résolution :

Proposez une solution pour que chaque joueur n’ait pas à se préoccuper de la structure du plateau et du type de jeu auquel il joue pour effectuer ses déplacements. Le déroulement du jeu sera sous la responsabilité de l’objet Monopoly. Le tirage de dés sera un service fourni par la classe Monopoly pour la classe Player.

Elaborez le diagramme de classe correspondant et identifiez les méthodes nécessaires sur chaque classe.

Elaborez un diagramme de collaboration pour illustrer le déplacement d’un joueur sur 3 cases.

## Exercice 2 : Contrainte sur le déplacement d’un joueur

### Problème :

En fonction de la case sur laquelle le joueur se trouve, son déplacement au tour suivant peut être contraint. Les règles du Monopoly nous permettent d’identifier au moins deux situations différentes :

1. Etat dit « Libre » : au tour suivant le joueur lance les dés et avance du nombre de cases indiqué par les dés.
2. Etat dit « Prisonnier » : lorsque le joueur se trouve en Prison, celui-ci doit faire un double avec les dés pour pouvoir sortir. Au bout de trois tentatives sans double, il paye 50 € et avance et avance du nombre de cases indiqué par les dés.

L’automate suivant résume le comportement d’un joueur en fonction de son état:



### Résolution :

Proposez une solution pour que chaque joueur n’ait pas à se préoccuper de son état lors de son tour de jeu.

Complétez le diagramme de classe élaboré précédemment et identifiez les classes et méthodes nécessaires.

Elaborez des diagrammes de collaboration pour illustrer le fonctionnement d’un tour de jeu.

Les scénarios sont les suivants :

* + Déplacement normal sur trois cases.
    - + Variante : Déplacement normal sur trois cases, avec passage par la case « départ ».
  + Déplacement sur deux cases, la deuxième case étant « Allez en Prison ».
  + Sortie de prison d’un joueur après obtention d’un double avec les dés.
  + Déplacement d’un joueur libre avec un double d’as (suivi d’un jet de dé quelconque).
    - + Variante : le double d’as est le troisième double joué dans le tour (le joueur va en prison).

## Exercice 3 : Généralisation de la gestion du déplacement

Problème :

Dans l’optique de généraliser le simulateur à d’autres jeux de plateau, pour lesquels des « bifurcations » existent (ex : Cluedo, Trivial Pursuit), on décide de permettre au joueur, lors de son déplacement, de choisir son chemin.

Pour cela IsquareIterator a été modifiée de la manière suivante (code fourni pour MonopolySquareIterator).

* next()  devient « nextPossibles() », qui ne renvoie plus une case, mais un ensemble de cases successeurs possibles.

ChooseCurrent(Isquare s) est ajoutée, pour positionner l’itérateur sur la case finalement choisie. ChooseCurrent(Isquare s) vérifie que s appartient bien à l’ensemble des successeurs possibles.

Remarque : la classe MonopolySquare ne change pas.

**Résolution :**

En vous basant sur le pattern de la template method, modifiez et spécialisez la classe Player de manière à permettre, dans un autre contexte, un déplacement offrant des choix de direction multiples.

Complétez/Modifiez les diagrammes de classe impactés par cette nouvelle fonctionnalité.

Modifiez le diagramme de collaboration montrant un déplacement normal sur trois cases pour prendre en compte les modifications effectuées.

## Exercice 4 : Gestion des propriétés

### Problème :

Les cases Propriété (Terrain, Compagnie, Gare) ont un comportement différent selon qu’elles sont libres ou achetées.

### Résolution :

Proposez une solution pour gérer facilement les changements de comportement des cases de propriété. Nous considérons qu’une case Propriété libre est achetée automatiquement par le joueur arrivant dessus si celui-ci possède l’argent nécessaire.

* Elaborez le diagramme d’état des cases Propriété.
* Complétez le diagramme de classe et identifiez les méthodes nécessaires sur chaque classe.
* Elaborez deux diagrammes de collaboration selon les deux scénarios suivants :
  + Un joueur arrive sur une propriété libre et l’achète.
  + Un joueur arrive sur une propriété possédée par un autre joueur.

## Exercice 5 : Achat des propriétés

### Problème :

Afin de rendre le simulateur plus démonstratif, on souhaite pouvoir « paramétrer » le comportement d’achat des différents joueurs.

Des profils de joueurs sont identifiés , par rapport à leur stratégie d’achat :

* Le « raider » : il achète tout ce qu’il trouve, sans limitation.
* L’ « épargnant » : il n’achète que lorsque son cash est supérieur à un seuil donné, paramétrable pour chaque joueur adoptant ce comportement.
* Il n’est pas interdit d’en imaginer d’autres…

**Résolution :**

Proposez une solution permettant à chaque joueur d’acheter suivant son profil (voire de changer de profil en cours de jeu) , sachant que l’on souhaite pouvoir ajouter de nouveaux comportements d’achat à l’avenir sans impacter la définition de la classe MonopolyPlayer :

* Elaborez le diagramme de classe correspondant et identifiez les méthodes nécessaires sur chaque classe.
* Elaborez deux diagrammes de collaboration selon les deux scénarios suivants :
  + Un « raider » arrive sur une propriété libre et l’achète.
  + Un « épargnant » arrive sur une propriété libre mais ne l’achète pas.

## STOP, à faire plus tard ….

## Exercice 6 : Suivi d’un joueur

### Problème :

Pour réaliser l’interface homme-machine du simulateur, il est nécessaire d’être prévenu de la fin du tour d’un joueur pour afficher ses mises à jour :

* valeur du crédit
* case sur laquelle il se trouve

### Résolution :

Proposez une solution pour mettre en place l’IHM de manière à ce que les classes du monopoly soient indépendantes de celle-ci (plus aucun « println » dans les classes, sauf trace des exceptions).

* Complétez le diagramme de classe et identifiez les méthodes nécessaires sur chaque classe.
* Reprenez le diagramme de collaboration de déplacement sur trois cases et complétez-le pour prendre en compte les modifications effectuées.

--------------------------------------------------------------------------------------------------------------------------

(Partie II : Construire l’application FS )

* L’organisation des Dossiers-Fichiers-Liens doit être simple, cohérente, sans redondance ;
* Le formatage des Unités Logiques doit être fait de façon souple, pour que l’on puisse ajouter de nouveaux formats sans impact sur la partie présentation de l’application ;
* Il faut pouvoir ajouter de nouveaux traitements sur les Dossiers-Fichiers-Liens sans modifier cette hiérarchie.

**Modélisation et codage afin de percevoir l’apport de l’application des patterns.**