Rapport Associé
Projet Long : Monopoly

Groupe E31

Avril 2020

Table des matières

Introduction

Dans ce rapport, nous présentons l'ensemble des tâches réalisées pour la création du Jeu de Monopoly. Dans ce cadre, nous avons commencé par établir une vue générale de l'application en réalisant le diagramme

UML (figure 1). Ce jeu que l'on va concevoir est un jeu graphique. Pour ce faire, nous avons défini un modèle qui sera utilisé pour l'implantation de type MVC (modèle vue contrôleur) passif.

1 Présentation de l'architecture du projet

Le modèle que l'on a conçu a pour élément principal le jeu. Ce jeu a pour attribut la liste des joueurs, les deux dés, les cartes et le plateau. L'idée est d'avoir en données l'ensemble des éléments susceptible d'évoluer au cours d'une partie.

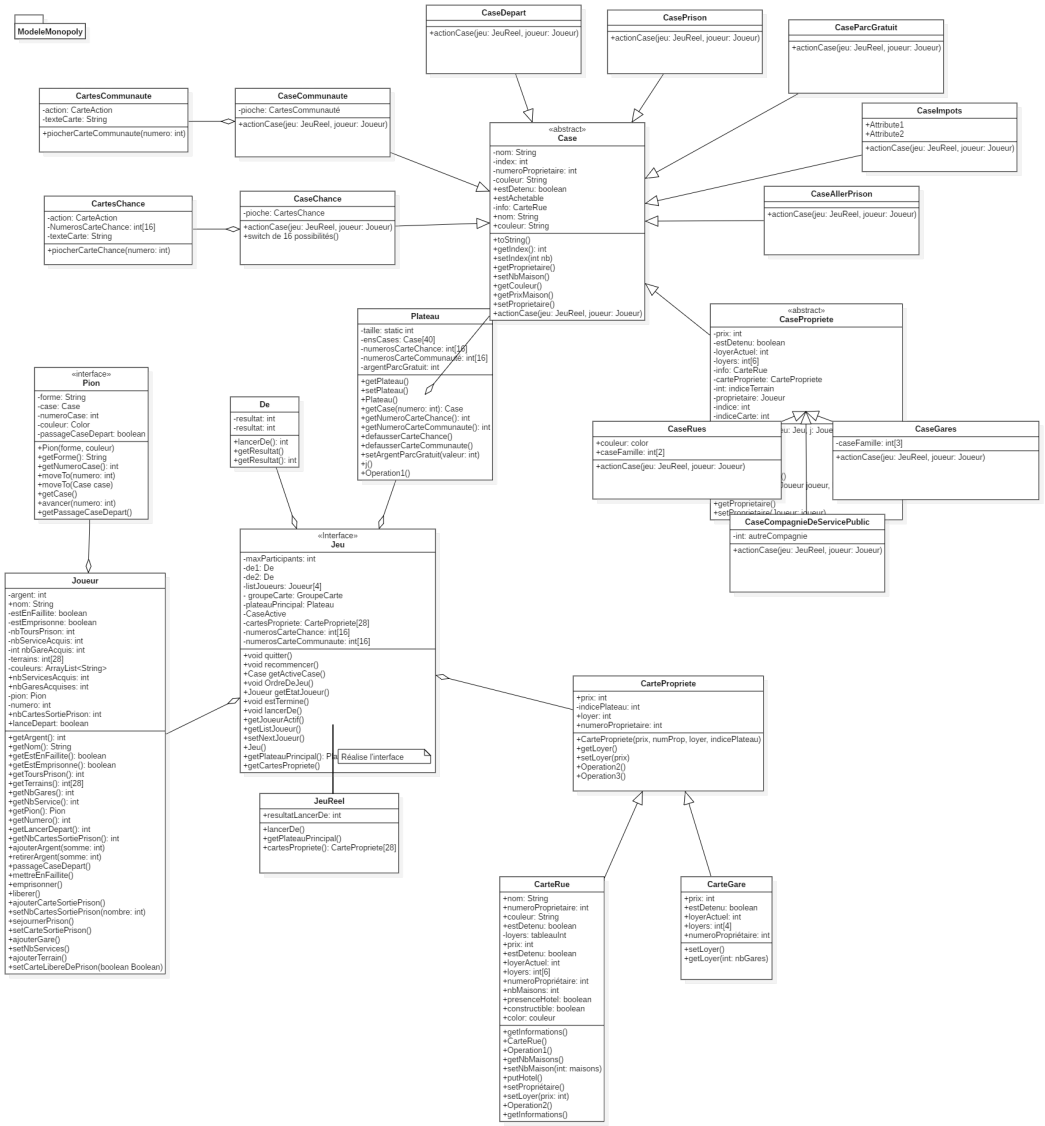

FIGURE 1: Diagramme UML

1.1 Explication du rôle des classes principales

Les principaux éléments sont donc : le Plateau, les Joueurs et les cartes.
Le plateau que l'on va utiliser sera composé de cases différentes. Ainsi, pour factoriser le plus d'éléments possibles, nous avons créé une classe abstraite *Case*. Ces différentes cases ont pour méthode principale : *actionCase()*. Cette procédure, lorsqu'elle est appelée modifie le modèle en fonction de l'action qu'elle doit effectuer.
Par exemple, lorsque l'utilisateur fait appel à *actionCase()* d'une case chance, elle fait piocher une carte chance qui pourrait faire avancer le joueur.
En ce qui concerne la classe *Joueur*, elle dispose en attribut d'un *Pion* qui va être l'élément qui va être déplacé d'une case à une autre (et aura notamment une position, une couleur et une forme). La classe Joueur dispose aussi de l'attribut argent qui va définir son fond de compte et d'un tableau de boolean qui va indiquer si le joueur est propriétaire ou non d'une case. En outre, il est aussi possible de savoir s'il est en faillite ou s'il dispose de certaines cartes.
Pour ce qui est des cartes, elles permettent aussi de faire évoluer le modèle en fonction de leurs caractéristiques (notamment les cartes communauté et les cartes chance).

1.2 Principaux choix réalisés

1.2.1 Le modèle des cartes

Le modèle de départ choisi pour les cartes était celui de la figure 2. Ce modèle s'appuyant sur le patron de conception *Stratégie* permettait une meilleure liaison dynamique car toutes les cartes héritaient de la classe abstraite *Carte*. Cependant, puisque certaines cartes sont utiles lorsqu'on est dans des cases spécifiques, on a choisi de mettre les cartes Chance et cartes Communauté dans les cases associées.

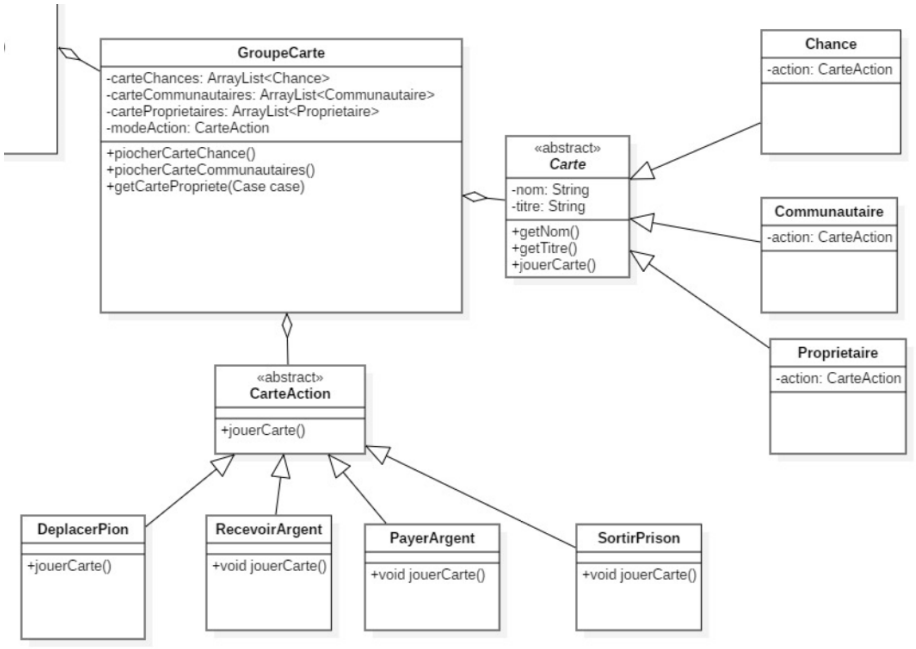

FIGURE 2: Modèle de départ des cartes

1.2.2 Le modèle MVC

Dans notre architecture, le modèle est complètement indépendant de la vue (ce même modèle pourrait être utilisé pour la création d'un jeu en console). Ainsi, la classe *VueMonopoly* permet d'établir la vue mais dispose aussi de tous les contrôleurs nécessaires à la gestion du modèle.

2 Principales classes

Les composantes principales du modèle Monopoly sont les Cases, les Cartes, le Plateau de jeu et les Joueurs. De nombreuses relations de composition ont été nécessaires. La classe Jeu fait le lien entre toutes les composantes du modèle.

La classe VueMonopoly constitue pour le moment la vue et le contrôleur. Elle comporte tous les composants graphiques nécessaires au jeu du Monopoly. Les actions associées aux composants graphiques ne sont pour la plupart pas encore implantées.

La classe Partie est en cours d'implantation. Elle utilise le modèle MVC en entier pour gérer une partie de Monopoly. Par exemple, durant un tour de jeu, elle affiche une fenêtre qui demande à un joueur s'il veut acheter une propriété (vue), composée de 2 boutons OUI/NON qui modifient par la suite (contrôleur) le modèle.

3 Listing des programmes par classes

Liste non-exhaustive des principales procédures des différentes classes

3.1 Joueur

Un joueur est construit à partir d'un pion, un numéro qui constitue son identifiant et le nom du joueur grâce au constructeur défini par :

public Joueur(**Pion** *pion*, **int** *numero*, **String** *nom*)

On peut réaliser des opérations sur la quantité d'argent d'un joueur grâce aux procédures suivantes :

public void ajouterArgent(**int** *somme*)

public void retirerArgent(**int** *somme*)

D'autres procédures servent à gérer la liberté d'un joueur, qui peut être ou non en prison :

public void emprisonner()

public void liberer()

public void ajouterCarteSortiePrison()

public void setNbCartesSortiePrison(**int** *nombre*)

public void sejournerPrison()

Un dernier paquet de procédures permet de gérer les cases ou cartes en possession du joueur :

public void ajouterService()

public void ajouterGare()

public void ajouterTerrain(**Case** *terrain*)

public void retirerTerrain(**Case** *terrain*)

public String getInfoTerrains()

3.2 Case et Carte

La classe case est une classe abstraite de laquelle héritent tous les types de case et ses procédures permettent d'obtenir l'indice d'une case ou d'activer l'action reliée à la case :

public int getIndex()

public abstract void actionCase(**JeuReel** *jeu*, **Joueur** *joueur*)

En particulier parmi les cases on retrouve la classe case propriété qui a en plus des procédures pour gérer le propriétaire de la case :

public Joueur getProprietaire()

public void setProprietaire(**Joueur** *joueur*)

En outre, dans la carte propriété liée à une case propriété on retrouve des méthodes gérant son prix et le loyer à payer si un joueur tombe sur une telle carte et n'est pas son propriétaire :

public int getLoyer()

public void setLoyer(**int** *montant*)

public int getPrix()

public void setPrix(**int** *prix*)

D'ailleurs certaines cartes propriété ont la particularité d'être des cartes rue, et on peut mettre des maisons ou des hôtels sur les rues donc on a besoin de procédures permettant de gérer ces entités :

public int getNbMaisons()

public void setNbMaisons(**int** *maisons*)

public void putHotel()

public void setConstructible()

Dans les classes Carte communauté et Carte chance on retrouve deux procédures permettant d'appliquer l'action d'une carte chance ou communauté piochée par la classe plateau. Pour appliquer son action la procédure a besoin de connaître le numéro identifiant la carte, le joueur qui l'a piochée et l'état du jeu pour produire des modifications :

public void piocherCarteCommunaute(**int** *numeroCarte*, **Joueur** *joueur*, **JeuReel** *jeu*)

public void piocherCarteChance(**int** *numeroCarte*, **Joueur** *joueur*, **JeuReel** *jeu*)

3.3 Pion

Un pion est construit à partir d'une forme et d'une couleur :

public Pion(String forme, Color couleur)

Il contient l'information de la position d'un joueur donc on peut déplacer sa position dans le plateau, soit en avançant d'un certain nombre de cases soit en se déplaçant directement par l'effet d'une carte chance ou communauté. Dans les déplacements d'un certain nombre de cases il faut vérifier si le pion est passé par la case départ :

public void avancer(**int** *nbCases*)

public void moveTo(**int** *numeroCase*)

public boolean getPassageCaseDepart()

3.4 Plateau

Le rôle principal de cette classe est d'identifier chaque case à un identifiant par le stockage des cases dans des tableaux. Mais aussi de piocher une carte chance ou communauté parmi un ensemble pour la communiquer à la procédure dans CarteChance et carte communauté :

public Case getCase(**int** *numero*)

public int getNumeroCarteChance()

public void defausserCarteChance()

public void defausserCarteCommunaute()

3.5 Jeu Réel

Le jeu est chargé de lancer les dés à chaque tour et jouer le tour par les procédures appelées par le contrôleur suivant :

public void jouer()

public void lancerDe()

L'ordre de jeu et le joueur qui joue actuellement ainsi que la fin du jeu sont gérés par :

public void setNextJoueur()

public void setjeuTermine()

Au début de la partie le jeu doit aussi construire certains éléments du plateau, en l'occurrence les cartes :

public void InitialiserCartes()

3.6 Vue Monopoly

Vue monopoly lance d'abord une fenêtre de bienvenue et demande à l'utilisateur combien de joueurs vont participer à travers un JDialog :

public void afficherBienvenue()

public void saisirNbjoueur()

À partir de cette information il demande à l'administrateur d'initialiser chaque joueur avec une couleur et une forme de pion déterminée. L'ordre dans lequel les joueurs vont jouer est défini par un premier lancé de dés. Après avoir ordonné les joueurs il faut leur attribuer un identifiant correspondant à leur place dans la file d'attente des tours :

public void initialiserJoueurs()

private void definirOrdreJeu()

private void redefinirNumero()

Ensuite on présente l'ordre de passage. Il existe une procédure permettant d'afficher le résultat du lancé des dés dans jeu réel :

public void afficherDe()

public void presenterOrdreJeu()

Le jeu commence finalement et il faut redéfinir la fenêtre principale et présenter l'interface de jeu :

public void redefinirFenetrePrincipal()

Dans cette nouvelle interface on retrouve en particulier les commandes de jeu :

public void afficherCommandes()

4 Organisation du travail

Pour l'organisation du travail, nous nous sommes répartis fondamentalement en 3 groupes : Omar-Benoit , Ababacar-Lucas, Laurène-Arthur-Avy.

Chaque groupe devait s'occuper d'une des trois parties majeures du projet :
Omar et Benoit se sont occupés de la partie des cartes et des cases.
Ababacar et Lucas se sont occupés de la partie Jeu et Partie.
Laurène, Arthur et Avy se sont occupés de la partie Joueur et Pion.

L'organisation intra-binôme a différé d'un groupe à l'autre selon les parties demandées. Par exemple Omar et Benoit ont travaillé une grande partie de leur tâche en binôme, en SHARESCREEN sur Discord vu que ces deux parties étaient fortement liées et que le diagramme de classe de cette partie était modifié au fur et au mesure de leur avancée, et cette méthode s'est avérée efficace pour eux.

Et finalement, dès qu'un groupe finissait la partie qui lui était accordée, il commençait à aider le groupe en difficulté, ou en retard.

5 Difficultés rencontrées

Durant la conception de notre application, nous avons rencontré diverses difficultés.

La première difficulté est récurrente et commune à tout le groupe : nous ne pouvons pas tous travailler sur toutes les classes ; la cohérence des noms des méthodes, mais surtout la cohérence de conception globale est donc difficile mais absolument nécessaire. Pour que chacun puisse avancer efficacement et que chaque classe soit codée correctement, il faut que tous les membres du groupe aient parfaitement compris l'architecture globale de l'application pour être sûr de quelle classe gère quel détail.
Cependant il est irréaliste de penser que notre première proposition d'architecture était parfaite : par exemple, elle ne précisait pas quelles classes devaient gérer un passage par la case départ après lancé des dés.
Cette difficulté se surmonte donc finalement grâce à beaucoup de recul et des échanges réguliers entre les personnes qui codent des classes liées voire dépendantes.

La méthode actionCase() de la classe CaseRue était compliquée à implanter. En effet, Benoit s'est rendu compte qu'après l'achat d'une propriété, il fallait réactualiser les loyer de toutes les cartes de la famille correspondante si le joueur les possédait toutes après l'achat de la dernière. Et aussi donner la possibilité de placer des maisons sur les 2 ou 3 cartes de la même famille, d'un coup, lorsque le joueur achète la dernière. De ce fait, il a choisi d'ajouter un attribut de classe : un ou deux entiers correspondant pour chaque rue aux indices des autres rues de la même famille.

D'après Ababacar, la classe VueMonopoly est très longue du fait de la difficulté de réaliser différentes Classes de fenêtre et réaliser des Listener externes. Arthur le rejoint sur le fait qu'il existe beaucoup d'options à implémenter (construire sur ses terrains, hypothéquer, passer au tour suivant, etc) et comme les options et le texte sont les seules variantes, il serait intéressant de pouvoir construire les fenêtres en utilisant un modèle prédéfini.