# Rapport

# MONOPOLY Toulouse

Groupe n°3

## Table des matières



## 1. Introduction et sujet

Ce projet nommé *"Monopoly Toulouse"* propose une réinterprétation du jeu classique, intégrant les rues et les sites emblématiques de la ville de Toulouse. Cette version offre une nouvelle perspective sur la "ville rose", invitant les joueurs à acquérir et à négocier des propriétés familières.

Les membres de l'équipe se sont réparti les tâches pour englober les différents aspects du projet, de la structure du jeu jusqu'à l'interface utilisateur. Ce rapport actuel résume les progrès accomplis au cours des différentes itérations du projet.

# 2. Fonctionnalités

## Gestion du jeu

Cette partie contient les différends du jeu et les interactions possibles entre eux. On s'est d'abord concentré sur cette partie pour avoir les ressources nécessaires pour faire tourner le jeu :

- ☑ Gestion des joueurs : Itération 1
- ☑ Gestion des propriétés et des maisons : Itération 1
- ☑ Types de cartes : Itération 2
- ☑ Gestion du plateau : Itération 1 et 2
- ☑ Gares : Itération 1
- ☑ Compagnies : Itération 2
- ☑ Case événements : Itération 2 et 3
- ☑ Dés : Itération 2
- ☑ Gestion d'une partie et des tours : Itération 3
- ☐ Initialisation d'une partie (Pas commencé)

## Interface Graphique

Cette partie fait référence à tout l'aspect graphique du Jeu (les fenêtres, panels …). On s'est d'abord concentré sur le plateau sur la 1re itération, puis sur les autres éléments par la suite :

- ☑ Création plateau : Itération 1
- ☑ ATH plateau : Itération 1 (et correction bug itération 2)
- ☑ Déplacement pions : Itération 1
- ☑ Fenêtre pour messages simples : Itération 3
- ☑ Fenêtre achat propriété : Itération 3
- ☑ Fenêtre Case Prison : Itération 3
- ☐ Fenêtre Carte communauté et carte chances : Itération 3 (Pas terminé)
- ☐ Écran de fin de partie (Pas commencé)
- ☐ Écran de début de partie (Pas commencé)

**Bilan :** On a implémenté toutes les règles classiques du Monopoly (sauf les cartes communautés pas terminées) permettant de jouer une partie normalement.

Il reste à faire l'initialisation de partie et la fin de partie pour avoir un Monopoly complet et ensuite peut-être mettre en place le système de négociation proposé.

# 3. Architecture

## a. Les sous-systèmes

On a découpé l'application en deux paquetages principaux :

- **GestionMonopoly** : Le paquetage qui gère le "back", c'est-à-dire toutes les informations du jeu et leurs manipulations.
- **InterfaceGraphique** : Le paquetage qui gère l'interface graphique.

C'est deux paquetages sont eux-mêmes décomposés en sous-paquetages :

Pour **GestionMonopoly**, on retrouve les paquetages :

- **Cartes** : Contient tous les types de cartes pouvant être créé
- **Cases** : Contient tous les types de cases pouvant être créé

Pour **InterfaceGraphique**, on retrouve les paquetages :

- **Utilitaires** : Contient des éléments de bases en swing (utiliser dans les autres paquetages)
- **FenêtreCase** : Contient toutes les fenêtres ouvertes lors de l'arrivée sur une case.

Voici un schéma récapitulatif :

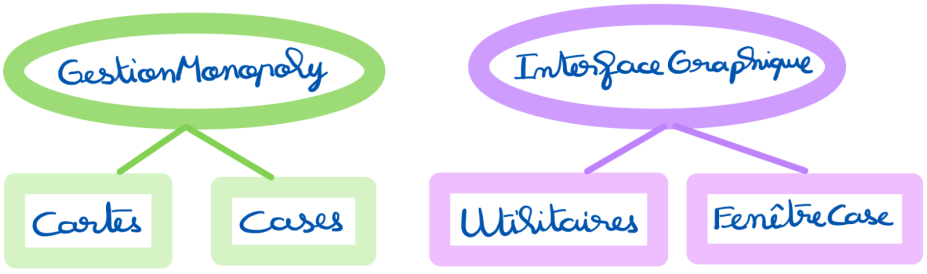
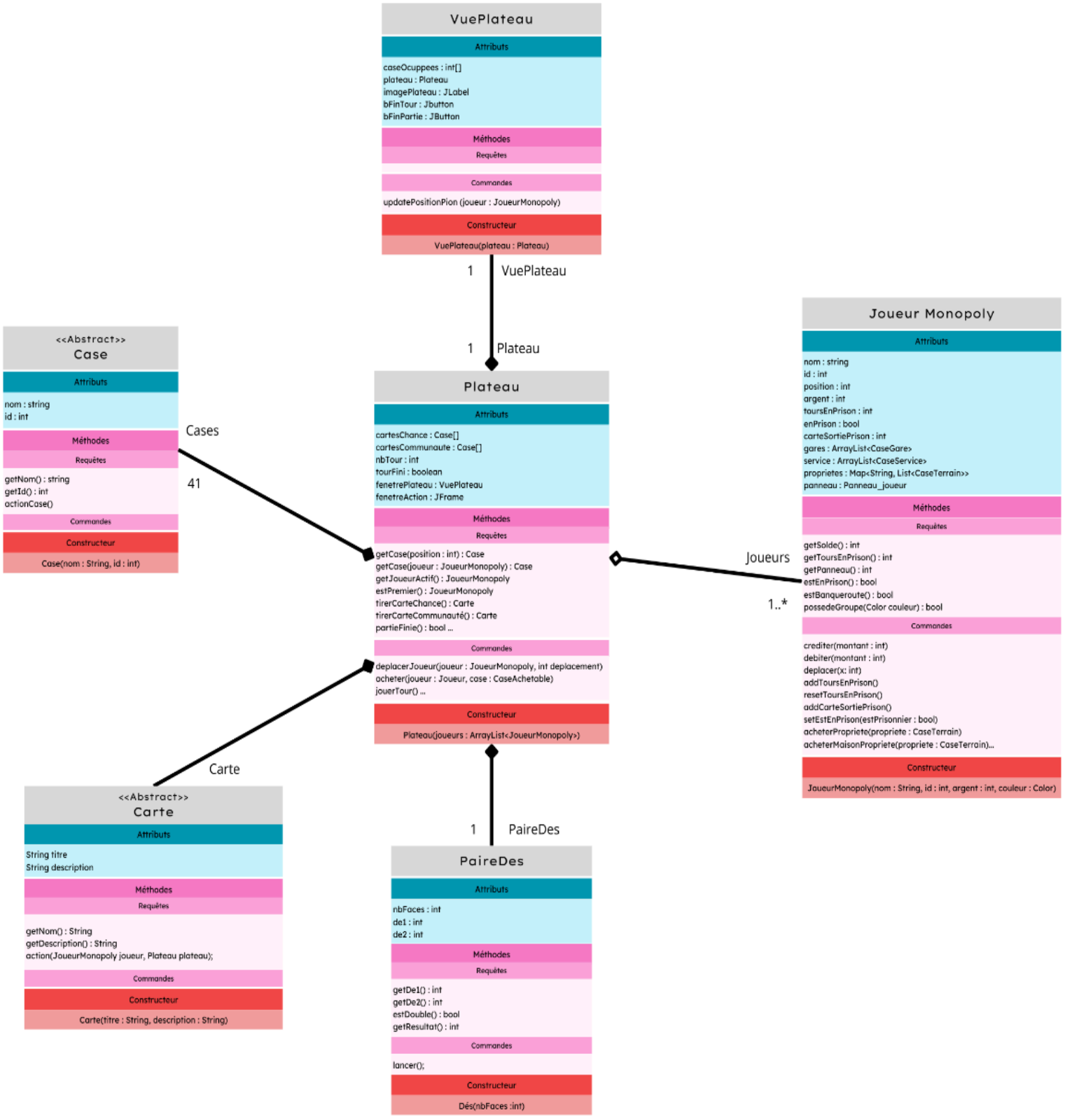


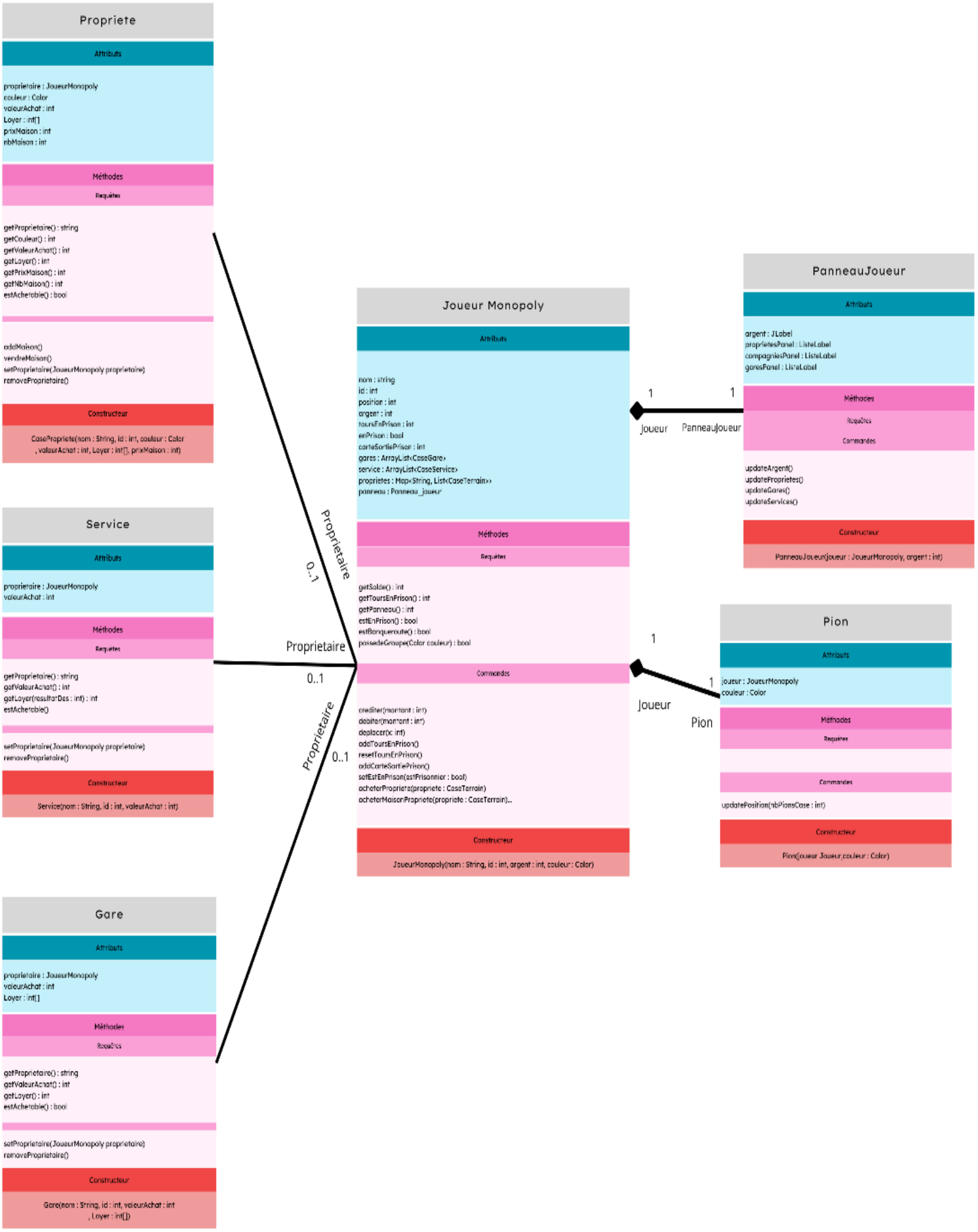

*Figure 1 : Schéma des paquetages*

# b. Diagramme de classe

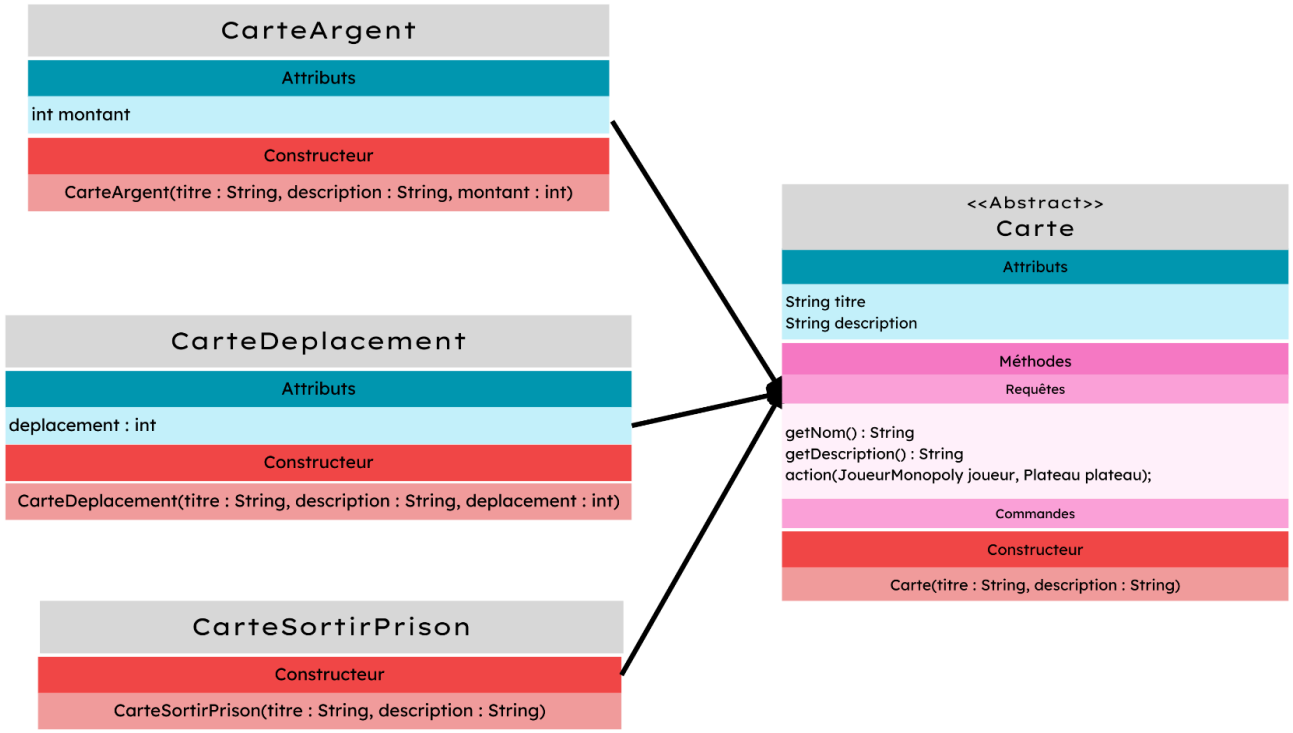
Diagramme de classe des principales classes qui constituent le cœur du Jeu de Monopoly.



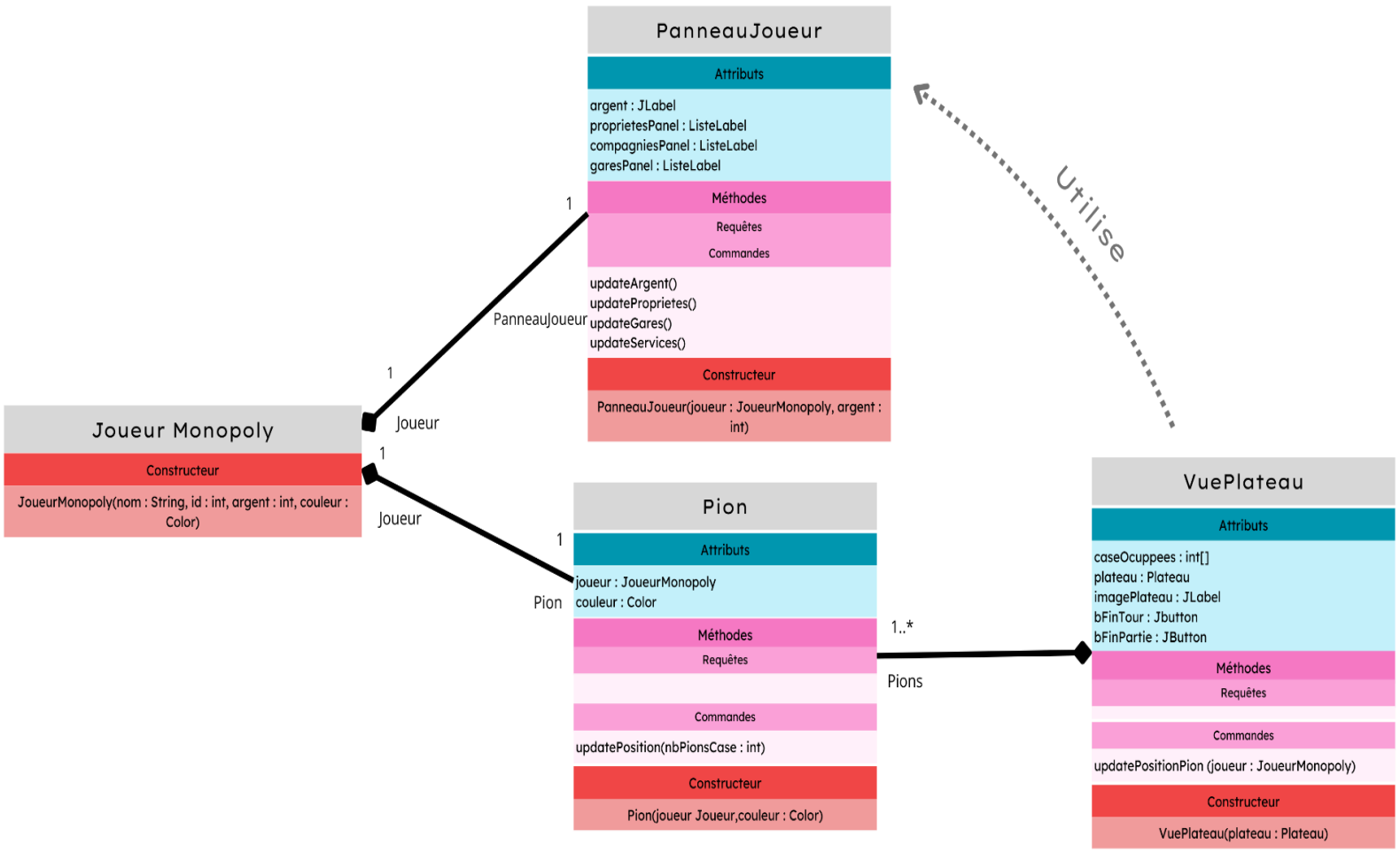
Maintenant voici le diagramme zoomé sur le Joueur :



Maintenant, nous allons détailler les cases
Pour les cartes, elles sont séparées en plusieurs types d'actions :




Maintenant, voici un focus sur la partie UI principale :



Il existe d'autres classes en dehors de celle présentée : Les classes dans le paquetage FenetreCase. Ces classes sont utilisées dans la méthode actionCase() des différentes cases pour avertir le joueur de l'action de la case.

### c. Fonctionnement

Une classe Partie (qui est en fait un main) crée les joueurs puis le plateau, puis joue des tours (méthode de plateau) jusqu'à ce qu'un joueur soit à sec.

Les tours sont gérés par le plateau qui, lors d'un tour, lance les dés pour le joueur courant, dès que le joueur à appuyer sur le bouton de la vue plateau, le déplace et lance l'action de la case sur laquelle est tombé le joueur (*). Après l'action, on attend que le joueur clique sur le bouton *Finir le tour* pour finir le tour et passer au Joueur suivant.
Pour ce qui est des informations des joueurs, celles-ci sont affichées à droite du plateau dans des panneaux qui sont interconnectés avec le joueur et sont mises à jour à la moindre action de celui-ci.

(*) Les actions des cases ouvrent toujours une fenêtre pour avertir le joueur de l'action. Celle-ci est créée grâce à une classe dédiée dans le paquetage FenetreCase, puis elle vient remplacer la fenetreAction de la classe plateau.

## 4. Les difficultés et les choix

**Plateau en image ?**

Nous nous sommes interrogés sur la possibilité de créer le plateau directement en Swing. Toutefois, afin de progresser rapidement tout en obtenant une interface utilisateur plaisante, nous avons opté pour une image créée avec Canva.

Cependant, cela a soulevé un problème concernant l'adaptation de la taille de l'image à l'écran. Pour que le plateau reste entièrement visible sur tous les écrans, il devait s'ajuster automatiquement à la taille de la fenêtre. Pour ce faire, nous avons mis en place un système de positionnement flexible qui réagit au redimensionnement de la

fenêtre, redimensionnant l'image et repositionnant les éléments aux nouvelles coordonnées.

**Les fenêtres pop-up pour les cases**

Nous avons rencontré un problème avec les fenêtres qui devaient apparaître puis disparaître (se fermer et non simplement être réduites). Une première solution consistait à utiliser les fenêtres de messages de Swing, mais celles-ci ne permettaient pas d'interagir avec elles, alors que nous avions besoin d'interaction, notamment pour la case Prison.

Nous avons donc opté pour la solution suivante : le plateau possède une fenêtreAction, qui est la fenêtre pop-up courante. Ainsi, les fenêtres se ferment automatiquement dès qu'une nouvelle apparaît.

**Problème de conventions**

Au début, nous avons rencontré des problèmes avec ce système, ce qui a considérablement ralenti la production lors de la deuxième itération, car nous avons dû tout réorganiser et reconnecter les classes.

Par conséquent, nous avons établi des conventions et des règles pour la soumission des classes (par exemple : "soumettre une classe qui compile", ce qui pouvait sembler évident pour certains mais pas pour d'autres) afin d'éviter de tels problèmes à l'avenir.

**Problème de communication**

La première itération, qui coïncidait avec les vacances, a été un peu compliquée, notamment en termes de communication entre tous les membres du groupe.

Nous avons donc profité de la transition entre la première et la deuxième itération pour discuter ensemble, permettant à chacun d'exprimer ce qu'il n'aimait pas dans le travail de groupe et de réfléchir collectivement à des solutions.

Le début de la deuxième itération s'est déroulé beaucoup mieux : la communication s'est rétablie, le positionnement sur les différentes user stories a été optimisé, et le travail a progressé plus rapidement que lors de la première itération.

## 5. Méthode agile et organisation de l'équipe

Nous avons utilisé la méthode agile Scrum, avec des **sprints documentés directement sur notre GitHub**, et nous nous sommes chacun positionnés sur des User-stories spécifiques.

Notre équipe a appliqué le principe PDCA (Plan-Do-Check-Act). Nous avons planifié nos tâches en créant plusieurs user stories couvrant l'ensemble du projet. Nous avons exécuté la majorité de ces tâches au cours de trois sprints, **vérifié les résultats dans un espace dédié "In Review" sur notre GitHub ou directement par message via Discord selon la personne**, puis apporté les ajustements nécessaires avant de les fusionner avec la branche principale (main) sur Git.

Pour la répartition des tâches, il y a eu deux principaux acteurs : Baptiste et Hamza

- ➔ Baptiste a géré l'outil de gestion avec les différentes user stories et Epics et vérifier les rendus des fonctionnalités. Il a également géré le dépôt GitHub, les problèmes de collisions, les demandes de merges…
  C'est celui qui gardait toujours un point de vue global sur le projet réalisé et signaler si quelqu'un partait dans une mauvaise direction.

- ➔ Hamza a géré l'équipe en demandant aux membres de se positionner sur les différentes user stories.

Assala, Mohammed, Ayman et Clément étaient principalement l'équipe de développement.