C++ : Travaux Pratiques
Compte Rendu de TP6: Monopoly

Liam SMALL, Fatoumata SEYE, Milo SOULARD, Nirmine KORTAM

26 janvier 2025

Introduction

Le but de ce TP est de recréer un jeu de Monopoly. Il se passera dans le terminal, pour 2 à 4 joueurs. Les fonctions principales du jeu y seront présentes, sur le terrain habituel composé de 40 cases.

Objectifs du TP :

Les principaux objectifs de ce TP sont les suivants :

- Concevoir des classes en C++ pour modéliser un jeu de Monopoly, en représentant les joueurs et les différentes cases du plateau.
- Appliquer les principes de l’héritage en programmation orientée objet pour spécialiser les types de cases (achetées, non achetables, terrains, gares, services publics, etc.).

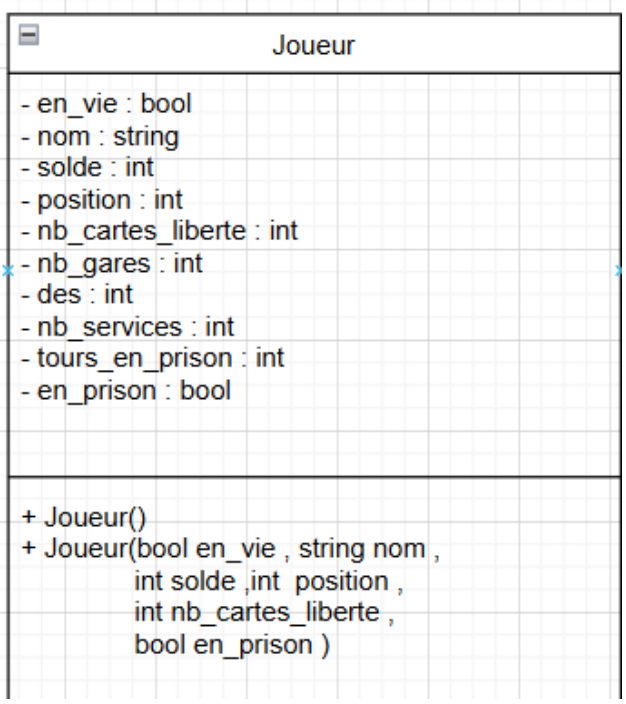
- Gérer les interactions entre les joueurs et les cases, comme l'achat de propriétés, le paiement des loyers et l'utilisation des cartes de liberté.
- Simuler les actions et événements du jeu (mouvement des joueurs, lancers de dés, gestion de la prison) en fonction des règles du Monopoly.
- Travailler avec des structures de données comme des vecteurs pour gérer les joueurs et les cases.

Conception et diagramme de classe :

Pour notre conception, nous avons choisi de créer les huit classes suivantes :

Joueur :

La classe `Joueur` représente un joueur dans le jeu de Monopoly. Elle gère des informations comme le nom, le solde, la position sur le plateau, le statut (en vie ou en prison), et des éléments spécifiques du jeu comme le nombre de cartes de liberté, de gares et de services publics possédés. Elle possède également des attributs pour gérer le lancer de dés et le nombre de tours passés en prison. Le constructeur permet d'initialiser un joueur avec des valeurs par défaut ou spécifiques.

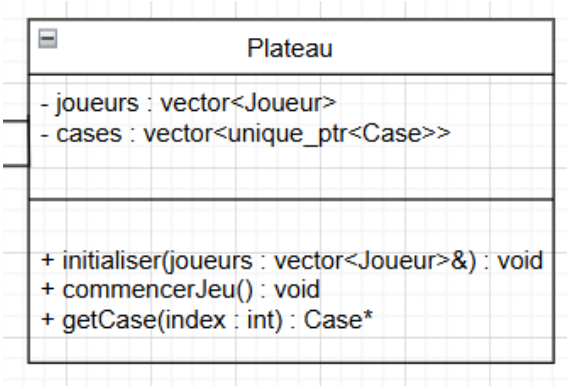


Plateau :

La classe `Plateau` gère le fonctionnement du plateau de jeu. Elle :

- ❖ Initialise les joueurs et les cases avec la méthode `initialiser`.
- ❖ Lance la partie et coordonne les tours des joueurs via `commencerJeu`.

❖ Accède aux cases spécifiques du plateau grâce à la méthode `getCase`.

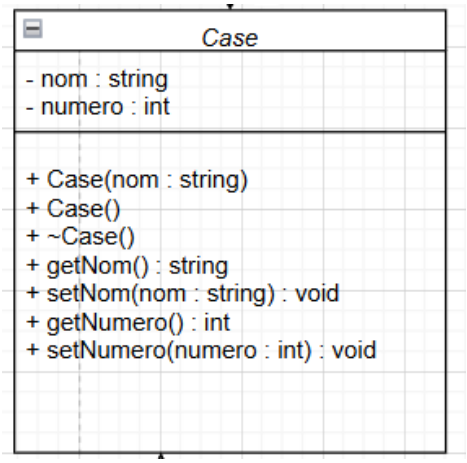


Case :

La classe `Case` représente une case générique sur le plateau de jeu . Elle contient deux attributs principaux : `nom`, qui stocke le nom de la case (par exemple, "Départ", "Chance", ou "Rue de la Paix"), et `numero`, qui indique la position de la case sur le plateau.

La classe dispose de deux constructeurs : un constructeur par défaut, qui crée une case sans nom ni numéros spécifiques, et un constructeur paramétré qui permet de créer une case avec un nom donné. Elle possède également un destructeur virtuel pour gérer correctement la destruction des objets dérivés.

Concernant les méthodes, la classe inclut des accesseurs et des mutateurs. Les accesseurs `getNom()` et `getNumero()` permettent de récupérer respectivement le nom et le numéro de la case, tandis que les mutateurs `setNom()` et `setNumero()` permettent de modifier ces valeurs.

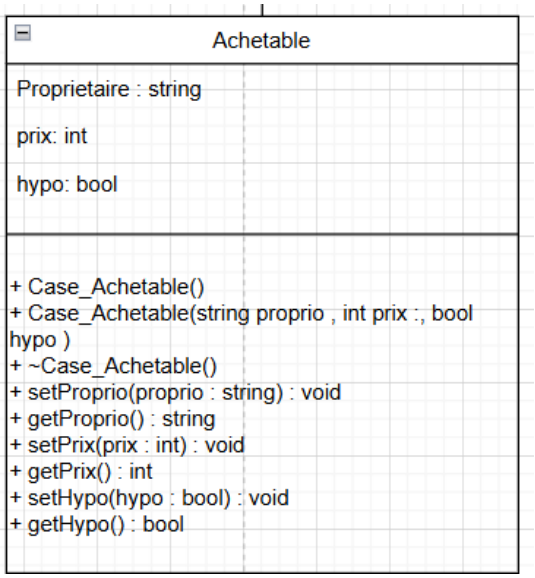


Case achetable :

La classe `Case_Achetable` dérive de la classe `Case` et représente les cases sur le plateau de Monopoly qui peuvent être achetées par un joueur. Elle ajoute des attributs et des méthodes spécifiques à ces cases, comme le propriétaire, le prix, et le statut hypothécaire de la case.

Elle dispose de deux constructeurs : un constructeur par défaut qui initialise la case avec des valeurs par défaut (pas de propriétaire, prix nul, non hypothéquée), et un constructeur paramétré qui permet d'initialiser la case avec un propriétaire, un prix et un statut hypothécaire spécifique.

Les méthodes de la classe permettent de gérer le propriétaire, le prix et le statut hypothécaire de la case.



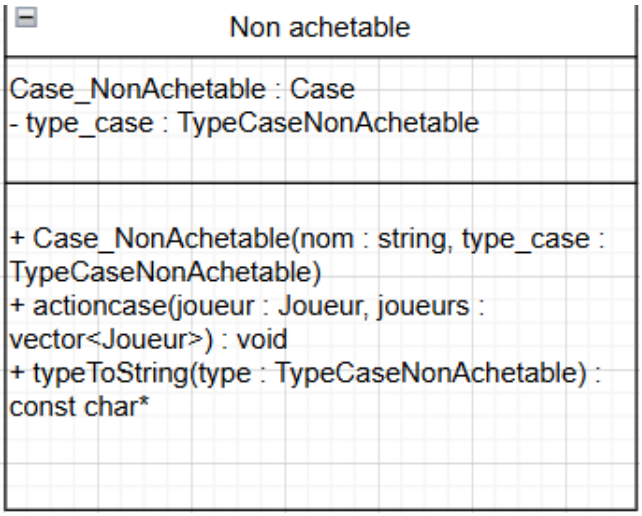
Case non_achetable :

La classe `Case_NonAchetable` hérite de la classe `Case` et représente les cases du plateau de Monopoly qui ne peuvent pas être achetées par les joueurs. Ces cases incluent des éléments comme "Départ", "Prison", "Chance", "Caisse de Communauté", et d'autres cases spéciales.0

Elle contient une énumération `TypeCaseNonAchetable` qui définit différents types de cases non achetables. Le constructeur permet d'initialiser la case avec un nom et un type spécifique (parmi les types définis dans l'énumération).

La méthode `actioncase` déclenche l'action associée à cette case. Selon le type de la case (par exemple, "Chance" ou "Impôt"), cette méthode applique les effets correspondants sur le joueur actif et, parfois, sur d'autres joueurs.

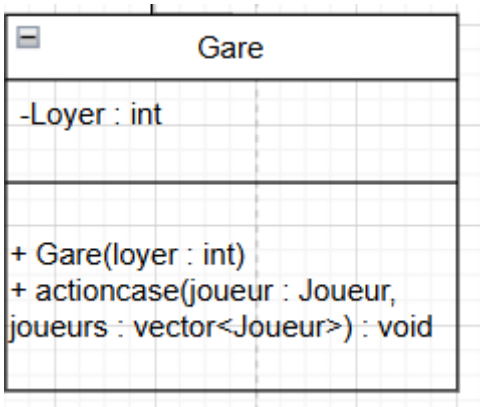
Enfin, la méthode `typeToString` est utilisée pour convertir le type de la case en une chaîne de caractères afin de faciliter l'affichage ou les logs.



Gare :

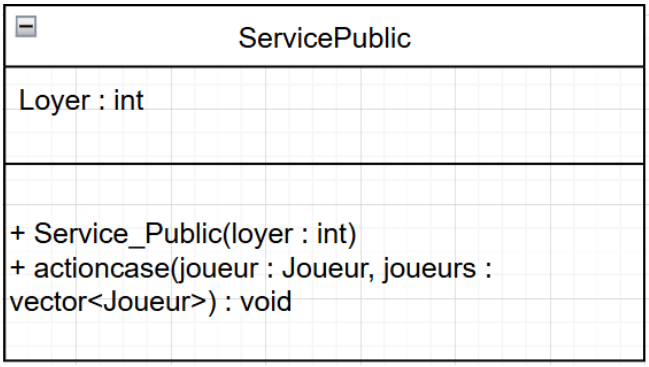
La classe `Gare` dérive de `Case_Achetable` et représente une case spécifique du plateau, où les joueurs peuvent acheter des gares. Elle possède un attribut supplémentaire, `loyer`, qui définit le montant du loyer que le joueur doit payer s'il atterrit sur une gare déjà possédée.

Le constructeur de la classe permet d'initialiser une gare avec un montant de loyer spécifique. La méthode `actioncase` définit le comportement lorsqu'un joueur atterrit sur la gare. Si la gare est déjà possédée, le joueur doit payer un loyer au propriétaire. Si la gare est encore libre, le joueur peut décider de l'acheter.



Service public :

Comme pour la classe gare, la classe `Service_Public` est dérivée de `Case_Achetable` et représente une case où les joueurs peuvent acheter des services publics. Elle contient un attribut `loyer`, qui définit le montant du loyer à payer par un joueur lorsqu'il atterrit sur cette case.



Case_Terrain:

`CaseTerrain` dérive de `Case_Achetable` et représente des terrains achetables tels que des propriétés ou des rues. Elle possède des attributs supplémentaires pour gérer les spécificités des terrains, comme `loyer`, `groupe` (la couleur), et `construction` (le nombre de constructions sur le terrain, telles que des maisons ou hôtels).

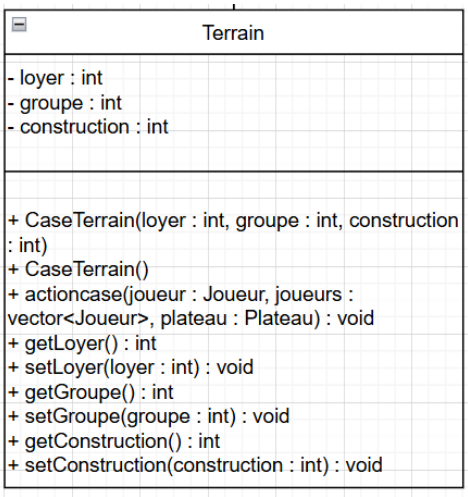
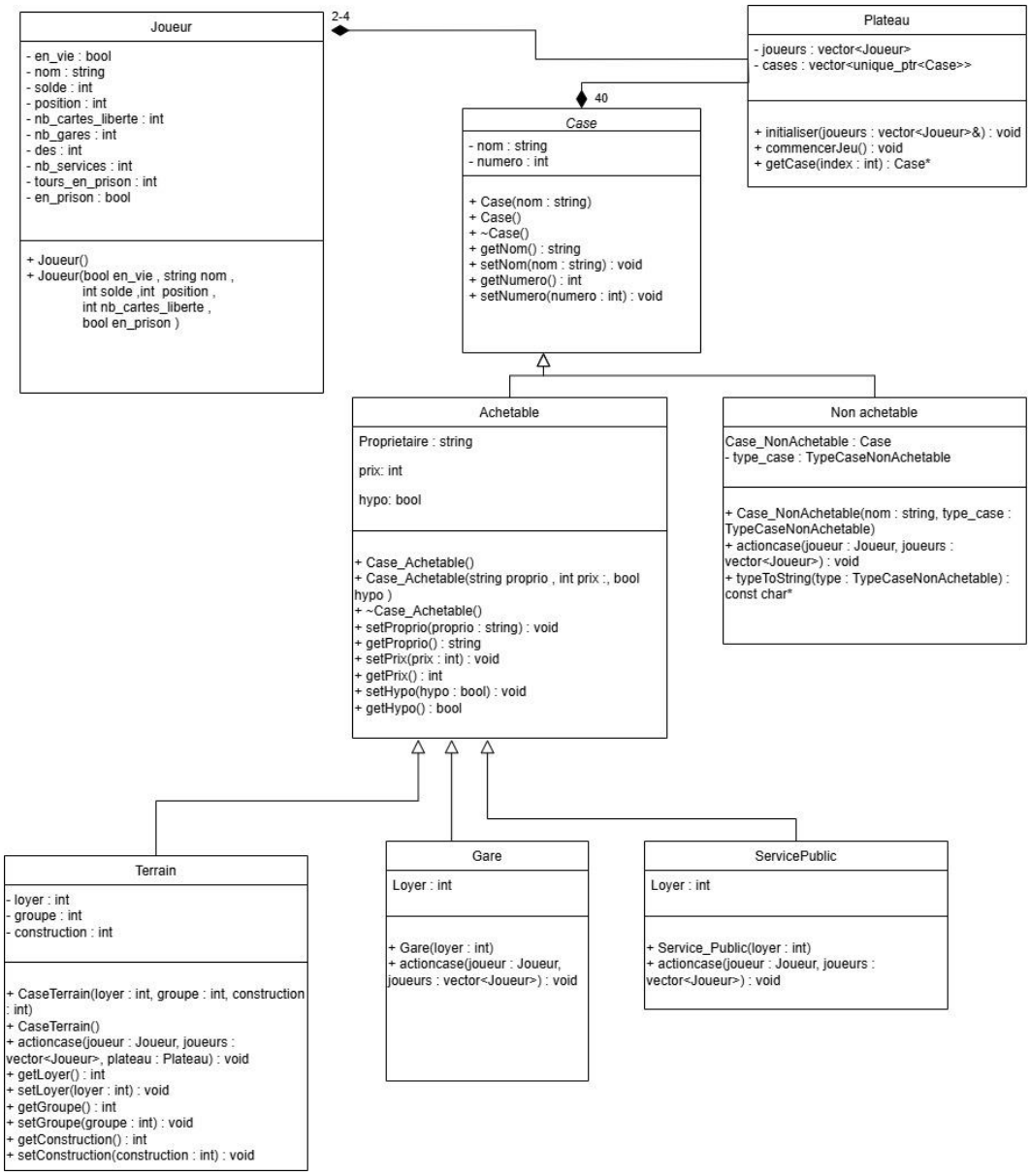


Diagramme de classe complet :



Explication des codes

Main

Le code débute dans le fichier main.cpp. On y trouve les premiers dialogues avec l'utilisateur. Le nombre de joueurs, leurs noms sont demandés et les instances des joueurs sont créées. L'ordre des joueurs y est décidé aléatoirement. Le plateau est instancié, et les fonctions initialiser et commencerJeu sont appelées.

Plateau

Le tableau a pour attributs le tableau de joueurs et le tableau de cases qui composent la partie. Dans initialiser, ce dernier est créé avec les différents types de cases. Les

paramètres des cases y sont entrés : toutes ont un nom et un numéro, et le reste varie en fonction de leur type. Dans commencerjeu, toute la partie se déroule. Tant que la variable booléenne jeuTerminé ne vaut pas 1, une boucle fait jouer tous les jours dans l'ordre défini, avec la fonction joueur.jouerTour. Cette fonction gère aussi l'affichage, et permet à l'utilisateur de rester informé des différentes fortunes et propriétés achetées des joueurs.

Cases

Les cases énoncées précédemment, comme expliqué dans le diagramme de classes, vont se scinder en plusieurs catégories : les non achetables, où seront contenues depart, prison, allez_en_prison, chance, caisse_de_communaute, impot, taxe_de_luxe et parc_gratuit. Elles sont dans un enum, et un switchcase s'occupe de gérer les effets sur les joueurs. La plupart sont des cases sans effets ou de simples ajouts ou retraits sur le solde du joueur. Pour les cartes chances et caisse de communauté, on tire une carte au hasard dans un vecteur de string où se trouvent les énoncés des cartes, et les effets sur les joueurs sont appliqués en fonction de ce qui en retourne.

La classe case_achetable est virtuelle pure, et a trois filles différentes ; les terrains, les gares et les services publics. Ils ont en commun d'avoir un prix d'achat, un propriétaire et un statut d'hypothèque. La case terrain a un loyer, qui varie en fonction du nombre de maisons construites, allant de 0 pour terrain nu et 5 pour l'hôtel. Ces constructions ne sont possibles que lorsque toutes les maisons du groupe (couleur) sont achetées. Malheureusement, faute de temps, l'achat de maison ne sera implémenté. Les gares et services publics fonctionnent similairement, mais eux n'ont qu'un loyer. Ce dernier est calculé dans la fonction actionCase que toutes 3 possèdent en exemplaires différents. Pour le terrain, on vérifie le propriétaire : s'il est vide, on donne le choix au joueur de l'acheter si son solde le permet. Si c'est quelqu'un d'autre, il paye le loyer en vérifiant si le propriétaire ne possède pas tous les terrains de la couleur. Sinon, il paye le double. Pour la gare, le loyer est calculé en fonction du nombre de gares du propriétaire. Dans le cas du service public, le nombre renvoyé par les dés est aussi pris en compte pour le loyer. Il est multiplié par 4 si le propriétaire possède 1 service et 10 s'il a les deux.

Joueurs

Ceux qui vont s'affronter sur ce plateau seront les joueurs. Ils ont 9 attributs : leur état, encore dans la course ou en faillite ; leur nom ; leur position ; leur solde ; leur nombre de cartes de remise en liberté, de gares, de services ; s'ils sont en prison ; depuis combien de temps et le montant du lancer de leur deux dés. En plus des getters, setters et incrémenteurs, les joueurs ont la fonction JouerTour où la majorité des actions de la partie se dérouleront. Tout d'abord, elle vérifie si le joueur est en prison. S'il s'y trouve, il tente de sortir en utilisant une carte, en payant une amende, ou en obtenant un double

au lancer de dés, sinon il y reste jusqu'à ce que le compteur de tours en prison atteigne 3. Hors de prison, le joueur lance deux dés pour avancer sur le plateau, avec des règles spéciales pour les doubles : Ils permettent de rejouer, mais un troisième double consécutif envoie le joueur en prison. La nouvelle position du joueur est calculée, en tenant compte du passage par la case départ pour récupérer les 200 monos. Ensuite, des actions spécifiques à la case où le joueur atterrit sont exécutées en fonction de son type : terrains, gares, services publics ou cases non achetables. Ces tours sont joués dans l'ordre aléatoire décidé dans le main jusqu'à ce qu'un seul joueur ait le monopole de la partie.

Jeux d’essais

Jeu d’essai classe Plateau

Input	Résultat attendu	OK/NOK
Initialiser un plateau avec 4 joueurs (Nirmine, Fatou, Milo, Liam)	Plateau contient 4 joueurs, chaque joueur commence avec son solde initial, et le plateau est correctement configuré avec 40 cases.	OK
Nirmine achète "Rue Lecourbe"	Propriétaire de "Rue Lecourbe" devient Nirmine. Solde de Nirmine diminue de 60 unités (le prix de la case).	OK
Fatou arrive sur "Rue Lecourbe" (propriété d'Alice) et doit payer un loyer	Solde de Fatou diminue du loyer (calculé selon les règles). Solde de Nirmine augmente du montant du l	OK
Milo arrive sur "Taxe de Luxe"	Solde de Milo diminue de 75 unités	OK
Liam atterrit sur "Allez en Prison"	Liam est envoyé à la case Prison, et son prochain tour est affecté en conséquence.	OK
Un joueur (Nirmine) fait faillite et est retiré du jeu	Nirmine n'a plus de solde, est marquée comme "en faillite". Ses propriétés deviennent disponibles pour les autres joueurs	OK
Tous les joueurs sauf un sont en faillite	La partie se termine, et un message affiche le joueur restant (gagnant).	OK
Vérifier l'état des propriétés après chaque tour	Le programme affiche les propriétaires actuels de chaque case. Les données correspondent aux modifications dues à l'achat ou faillite.	OK
Tirer une carte "Caisse de Communauté" ou "Chance"	La carte affecte le joueur (gains, pertes ou autre action). Les résultats	OK

	suivent le texte de la carte tirée.	
Tous les joueurs passent par la case "Départ" au moins une fois	Le solde de chaque joueur augmente de la somme allouée au passage de cette case.	OK
Exécuter la méthode commencerJeu() jusqu'à la fin	La méthode se termine lorsque les conditions de victoire sont remplies, sans boucle infinie ni erreur. Le gagnant final est déterminé correctement.	OK

Jeu d'essai classe Case

Input	Résultat attendu	OK/NOK
Joueur Nirmine arrive sur une case générique	Message : *"Nirmine arrive sur une case générique."*	OK
Joueur solde = 500, Case achetable (prix=200), réponse = o	Joueur devient propriétaire, solde = 300 monos.	OK
Joueur solde = 500, Case achetable (prix=200), réponse = n	Aucun changement, propriété reste libre.	OK
Joueur Fatou arrive sur une case non achetable	Message : *"Fatou visite une case spéciale."*	OK
Joueur solde = 400, Terrain propriétaire = "Milo", loyer = 50	Joueur solde = 350 monos, Milo reçoit 50 monos.	OK
Joueur solde = 300, Gare propriétaire = "Nirmine", loyer = 100	Joueur solde = 200 monos, Nirmine reçoit 100 monos.	OK
Joueur solde = 600, Gare libre (prix = 200), réponse = o	Joueur devient propriétaire, solde = 400 monos.	OK
Joueur solde = 600, Service public libre (prix = 150), réponse = o	Joueur devient propriétaire, solde = 450 monos.	OK
Joueur solde = 500, Service public appartient à "Fatou", dés = 6, loyer calculé = 40	Joueur solde = 460 monos, Fatou reçoit 40 monos.	OK

Jeux d’essai classe Joueur

Input	Résultat attendu	OK/NOK
Instantiation d'un joueur avec `Joueur(false, "Liam", 1000, 0, 1, false)`	Joueur avec `nom=Liam`, `solde=1000`, `position=0`, `en_prison=false`.	OK
Lancement des dés : `dé1 = 3`, `dé2 = 5`. Position initiale = 0	Position finale = 8. Pas de double, tour terminé.	OK
Lancement des dés avec double 3 fois consécutives : `dé1 = 4`, `dé2 = 4`, `dé1 = 2`, `dé2 = 2`, `dé1 = 5`, `dé2 = 5`	Joueur va en prison après le troisième double consécutif.	OK
Arrivée sur la case `CaseTerrain` appartenant à un autre joueur (propriétaire = `"Fatou"`, loyer = 50). Solde initial = 200.	Solde = 150, `"Fatou"` reçoit 50.	OK
Joueur en prison et avec 50 de solde, choisit de payer l'amende.	Joueur quitte la prison. Solde = Solde initial - 50.	OK
Arrivée sur la case départ (position initiale 39, somme des dés = 3).	Solde augmente de 200 monos pour avoir franchi la case départ.	OK
Arrivée sur la case `Gare`, libre. Prix d'achat 200. Joueur décide d'acheter. Solde initial = 500.	Joueur devient propriétaire de la gare. Solde = 300.	OK
Arrivée sur la case `Service_Public` appartenant à `"Milo"`. Joueur réalise un lancer de dés donnant 4 + 6 et le coefficient de loyer est de 10. Solde initial = 600.	Loyer = 10 × 10 = 100. Joueur solde = 500, `"Milo"` reçoit 100.	NOK

Conclusion

En conclusion, ce TP a permis de modéliser différentes classes complexes pour simuler un jeu de Monopoly, en utilisant des concepts de la programmation orientée objet comme l'héritage, l'encapsulation et la gestion des interactions entre objets. Au cours de la conception, nous avons également eu à faire des choix entre créer plusieurs classes ou faire une énumération pour gérer toutes les cases non achetables.

Le travail collaboratif via GitHub nous a à nouveau entraînés à mieux organiser le code en équipe, à gérer les versions et à intégrer les contributions de chacun efficacement.