Rapport de fin de projet : Monopoly Game

Badre Dorian Nicolas

April 2024

1 Introduction

Ce rapport présente le projet de développement d'un jeu de Monopoly en Java, ainsi que son état d'avancement, les itérations réalisées,le diagramme de classe, les métriques pertinentes et leur interprétation, ainsi qu'un bilan critique du déroulement du projet et de ses perspectives d'évolution.

2 État d'avancement du projet

Le projet a été réalisé dans son intégralité. Toutes les fonctionnalités du jeu de Monopoly ont été implémentées. Le jeu fonctionne comme prévu et peut être joué par plusieurs joueurs.

3 Itérations réalisées

Les itérations du projet se sont déroulées comme suit :

1. **Itération 1 :** Mise en place de l'architecture de base du jeu, création des classes principales (Player, Case, Board) et des fonctionnalités de base (déplacement du joueur).
2. **Itération 2 :** Ajout des fonctionnalités avancées telles que la gestion de l'argent, les cases propriétés.
3. **Itération 3 :** Implémentation des cases Chance et Caisse de communauté, de l'interface qui est appellés uniquement par le controller (classe MonopolyGame).
4. **Itération 4 :** Finalisation du jeu avec l'ajout des dernières fonctionnalités(Prison , maisons hotels), y compris la gestion du parc gratuit (Free Parking) et la résolution des problèmes.

4 Code final et diagramme de classe

Le code final du projet ainsi que le diagramme de classe sont présentés ci-dessous.

4.1 Diagramme de classe

Le diagramme de classe du jeu de Monopoly représente les principales entités et leurs relations dans le système. Il est conçu pour fournir une vue d'ensemble de la structure du jeu et des interactions entre ses composants.

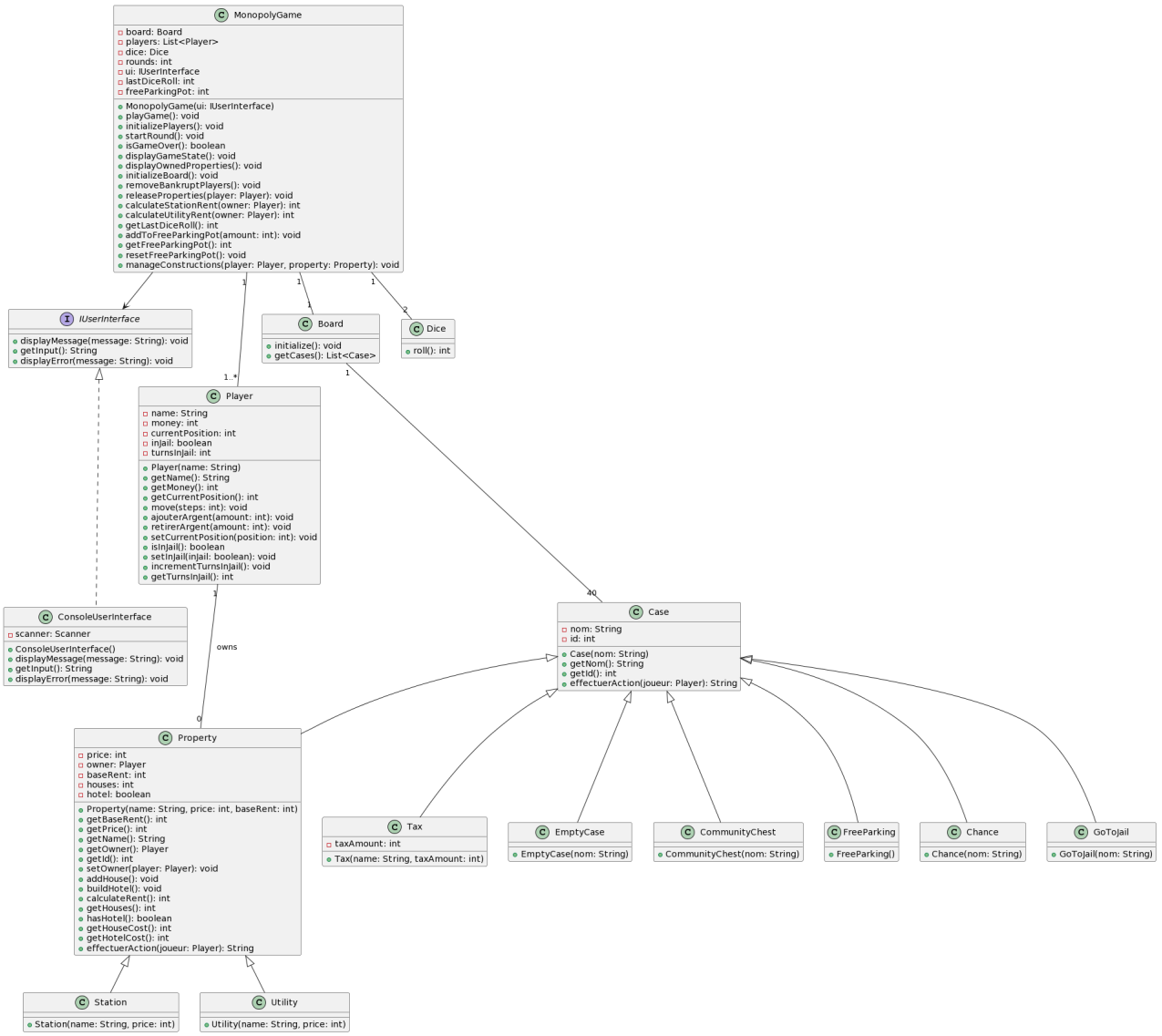

Figure 1: Diagramme de classe du jeu de Monopoly

Ce diagramme inclut les classes principales telles que Player (Joueur), Case (Case sur le plateau de jeu), Property (Propriété), Station (Gare), Utility (Services publics), ainsi que d'autres types de cases spéciales comme Tax (Impôts), EmptyCase (Case vide), CommunityChest (Caisse de Communauté), Chance (Chance), GoToJail (Aller en Prison) et FreeParking (Parc Gratuit).

On y trouve également la classe principale du jeu, MonopolyGame, qui coordonne le déroulement du jeu, ainsi que l'interface IUserInterface et la classe Dice (Dés) pour la gestion des interactions utilisateur et le lancer des dés.

Chaque classe possède ses attributs et méthodes propres, permettant de modéliser les actions et les interactions entre les différents éléments du jeu. Les relations entre les classes, telles que l'héritage, la composition et l'agrégation, sont également représentées pour refléter la structure et le fonctionnement du jeu de Monopoly.

4.2 Code final

Le code final du projet est disponible dans le dépôt GitHub à l'adresse suivante : `https://github.com/NicolasRbls/Jeux_Monopoly.git`.

Toutes les versions sont disponibles en changeant simplement de branche et en sélectionnant celle portant le nom approprié.

Comment jouer à notre Monopoly :

Le Monopoly que nous avons crée est une version informatisée du célèbre jeu de société Monopoly. Voici comment jouer :

1. **Préparation du jeu :** Lancez le code MonopolyGame sur VS Code.
2. **Sélection des joueurs :** Vous pouvez Sélectionner le nombre de joueurs que vous souhaiter dans la partie.
3. **Déplacement :** À tour de rôle, chaque joueur lance les dés automatiquement et se déplace en fonction du nombre indiqué par les dés.
4. **Actions spéciales :** Lorsque vous atterrissez sur une case spéciale comme "Chance", "Caisse de Communauté" ou autres une action spéciale est déclenchée automatiquement. Pour les propriétés, une action est déclenchée il faut donc suivre les instructions affichées à l'écran(Acheter ou non).
5. **Constructions :** Si vous tomber sur une propriété que vous possédez, vous pouvez construire des maisons et des hôtels pour augmenter les loyers.
6. **Gestion de l'argent :** Vous gagnez de l'argent en recevant des loyers des autres joueurs. Vous dépenserez de l'argent pour acheter des propriétés, payer des loyers, des impôts ou des amendes.

7. **Objectif :** Le but du jeu est d'être le dernier joueur en jeu. Vous pouvez éliminer les autres joueurs en les mettant en faillite en les forçant à payer des loyers qu'ils ne peuvent pas se permettre.

8. **Fin de partie :** La partie se termine lorsque tous les autres joueurs ont fait faillite et que vous êtes le seul joueur en jeu. Vous êtes alors déclaré vainqueur !

- Nombre total de lignes de code : 832 lignes
- Nombre de classes et d'interfaces : 16
- Nombre moyen de méthodes par classe : Le nombre moyen de méthodes par classe est de 7.5.

Quant à l'évaluation de ces nombres :

Nombre total de lignes de code : Le nombre total de lignes de code que nous avons, 832, semble être dans une fourchette raisonnable pour un projet comme celui-ci. C'est un signe que nous avons écrit suffisamment de code pour créer le jeu.

Nombre de classes et d'interfaces : Avec 16 classes et interfaces, notre projet semble avoir une structure raisonnablement modulaire et bien divisée. Cela suggère que nous avons réussi à organiser notre code en différents composants distincts, ce qui est un bon signe pour la maintenabilité et l'extensibilité du projet.

Nombre moyen de méthodes par classe :Un nombre moyen de 7.5 méthodes par classe est généralement considéré comme raisonnable dans la plupart des projets

5 Conception des tests et résultats

Nous avons conçu une classe de test selon les bonnes pratiques enseignées lors de nos cours de qualité logicielle. Cette classe de test, nommée "PlayerTest", est dédiée à la vérification de toutes les méthodes de la classe "Player". Tous les tests de cette classe ont été un succès. Voici trois exemples de méthodes de test :

```
@Test
void testAjouterArgent() {
Player player = new Player("TestPlayer");
player.ajouterArgent(500);
assertEquals(2000, player.getMoney()); // Vérifie si l'argent est correctement ajouté
}

@Test
void testGetCurrentPosition() {
Player player = new Player("TestPlayer");
assertEquals(0, player.getCurrentPosition()); // La position initiale doit être 0
}

@Test
void testGetMoney() {
Player player = new Player("TestPlayer");
assertEquals(1500, player.getMoney()); // Vérifie si le montant initial est correct
}
```

6 Bilan critique et perspectives d'évolution

Le projet s'est bien déroulé dans l'ensemble. On a bien planifié nos étapes (itérations) et on a bien travaillé ensemble en se repartissant le travail. Mais nous avons eu quelques difficultés a certain moments. Notamment, le code devenait de plus en plus compliqué et on a dû résoudre certain problèmes. Une chose qui nous a pris plus de temps que le reste, c'est de s'assurer qu'aucune classe ne parle directement à une interface, mais plutôt que tout passe par le controller, la classe MonopolyGame. Ça a été un peu compliqué de garder notre code organisé et clair.

En ce qui concerne les perspectives d'évolution, plusieurs pistes peuvent être envisagées :

- Amélioration de l'interface utilisateur en mettant en place une interface graphique.
- Ajout de fonctionnalités multijoueurs en ligne pour permettre aux joueurs de s'affronter à distance.
- Intégration de l'intelligence artificielle pour jouer contre des adversaires virtuels.
- Extension du jeu avec de nouvelles règles et variantes du Monopoly.

En conclusion, le projet de développement du jeu de Monopoly a été une très bonne expérience qui nous a permis de mieux comprendre ce qu'est la qualité logicielle et la complexité de relever certains défis pour obtenir un bon projet.