Monopoly

Rapport de projet
VR1B

PÉRIS Jules
CISSE Sekou
FERNEZ Dimitri
MAISSOURADZE Nicolas
AGRANE-BENLALAM Adam-Christophe

Projet de Programmation
License Informatique
Université Paris Cité
2024

Sommaire

CHAPITRE 1

Introduction

Notre projet de Monopoly consiste en la réinvention du célèbre jeu de société avec des éléments innovants et des mécanismes de jeu modernisés : des événements aléatoires, et des mécaniques de jeu nouvelles pour offrir une expérience de jeu plus riche et dynamique. Ce projet a été développé en utilisant Java et JavaFX, en adoptant une architecture MVC (Modèle-Vue-Contrôleur) pour une meilleure gestion et organisation du code.

Pour ce projet nous nous sommes inspirés d'un City Builder. En effet, les joueurs devront acquérir différents bâtiments et les améliorer au fil de la partie. Lorsqu'un joueur adverse atterri sur une case contrôlée par un building dont vous êtes propriétaire, il devra vous payer un loyer pour ce round. Le plateau de jeu est un carré de 11x11 cases contenant plusieurs chemins que les joueurs peuvent emprunter, à la manière d'une ville.

Une partie peut contenir 2 à 5 joueurs. Les joueurs ont la possibilité de choisir un nom et chaque partie lancée est sur une carte différente, générée aléatoirement.

Les objectifs principaux.
-Développer une version numérique et moderne du jeu Monopoly -Mettre en place une architecture logicielle robuste -Créer une interface utilisateur intuitive et attrayante -Intégrer des mécanismes de jeu innovants -Sauvegarde et chargement de partie -Génération de map aléatoire

CHAPITRE 2

Structure et implémentation

2.1 Description de l'architecture générale du projet

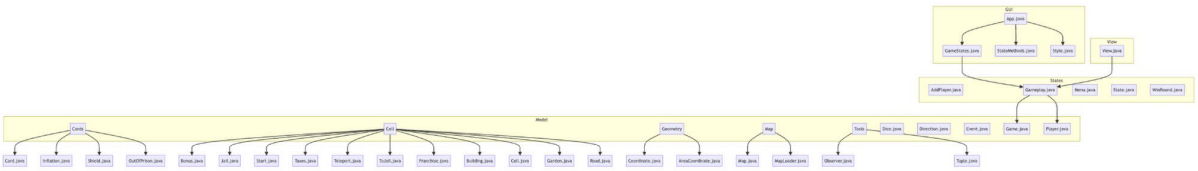
L'architecture générale de notre projet repose sur le modèle MVC (Modèle-Vue-Contrôleur) implémenté à l'aide d'un Observer pattern (Observer/Observable) et d'une machine à états (State pattern). Cela permet une séparation claire des responsabilités entre les données, la logique de traitement et l'interface utilisateur. Le projet est divisé en plusieurs packages correspondant aux différents composants :

Model : Gère les données du jeu, les règles et la logique de base. View : Responsable de l'affichage graphique et des interactions utilisateur. Controller : Intermédiaire entre le modèle et la vue, gère les entrées utilisateur et met à jour le modèle et la vue en conséquence.

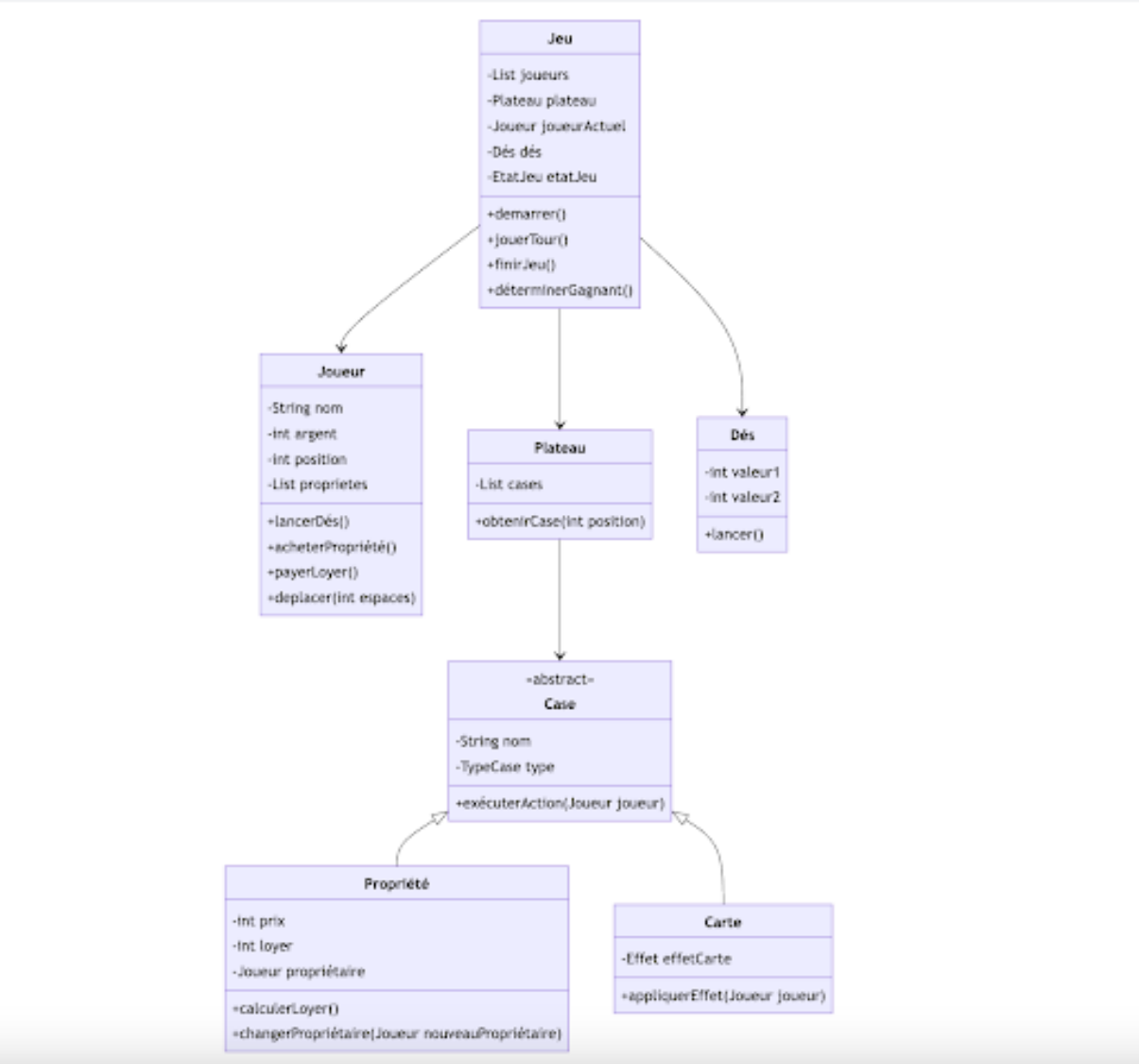
Nous avons également appliqué la POO pour beaucoup de nos classes dans notre code afin de garantir la modularité, et ainsi l'extensibilité et la maintenance de notre projet dans le futur.

Enfin, nous avons fait le choix de ne pas utiliser de boucle de jeu, pour des raisons de gain de temps et de simplicité. De ce fait, la classe TranslateTransition de JavaFX est chargée d'animer les déplacements des pions sur le plateau en fonction des coordonnées.

2.2 Diagramme des classes



2.3 Diagramme du Model



CHAPITRE 3

Fonctionnalités principales

Nous avons opté pour un design simple et intuitif avec des informations contextuelles pour chaque case (propriétés, bonus, etc.) lorsqu'on a la souris dessus. Ainsi les utilisateurs peuvent facilement naviguer entre les différentes sections du jeu sans confusion.

3.1 Interface utilisateur et plateau de jeu

3.2 fonctionnalité de base

- Achat de building et loyer perçu quand un autre joueur est sur une case contrôlée par votre building.
- Possibilité d'améliorer votre building pour augmenter le loyer perçu.
- Mise en vente de building lorsqu'un joueur n'a plus d'argent.
- Certains building ont la possibilité d'être étendus en ajoutant un jardin ce qui augmente le loyer perçu. (`Garden.java`)
- D'autres types de buildings ayant un prix plus élevé sont également implémentés (restaurant et musée). Ces infrastructures permettent au joueur qui les possède de recevoir un revenu tous les 3 rounds. (`package Franchise  Franchise.java` et ses classes filles)

3.3 Case Bonus/Malus

Sur le plateau de jeu, des cases aux effets spécifiques ont été créées (package Bonus Bonus.java et ses classes filles). Trois exemples :

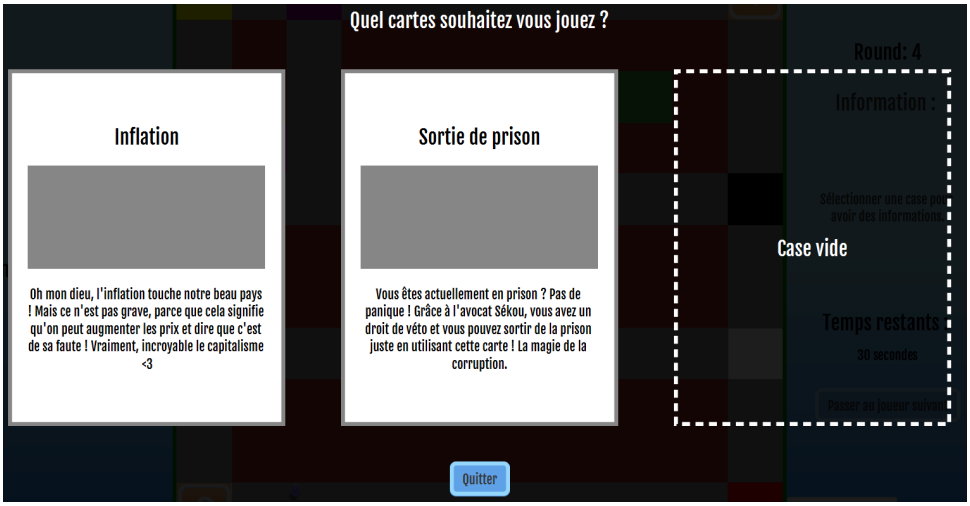
(a) **Taxe :** Le joueur doit payer une certaine somme d'argent

(b) **Teleport :** Le joueur paye pour se téléporter où il le souhaite sinon est téléporté aléatoirement

(c) **Chance :** Le joueur pioche une carte

3.4 Cartes

Le joueur peut collecter des cartes aux capacités diverses lorsqu'il atterrit sur une case Chance, modifiant ainsi l'état du jeu ou des joueurs en conséquence. (package Cards Cards.java et ses classes filles).



3.5 Multiples chemins et sens interdit

Dans notre Monopoly, le joueur à la possibilité de se déplacer librement sur le plateau de jeu parmis les différents chemin possibles. Toutefois, pour renforcer la difficulté et l'aspect stratégique, il y a certain pourcentage de chance pour que des panneaux stop bloquant l'accès apparaissent sur certains chemins selon la direction (nord,sud...).

3.6 Catastrophes/événements et réparation

Différents événements tels que des catastrophes naturelles peuvent se produire lors d'un round. Cela aura pour effets de détruire certains buildings que les joueurs devront réparer en payant une certaine somme d'argent.

CHAPITRE 4

Sprints et Développement

Chaque semaine, nous avons introduit de nouvelles fonctionnalités. Voici un bref résumé de la contribution individuelle de chaque membre :

4.1 Adam

- Mise en place de la machine à états du projet.
- Développement du menu principal et des différentes vues du jeu.
- Ajout des nouvelles vues et des nouveaux états du jeu.
- Implémentation de la vue de choix des joueurs et ajout du sens interdit.
- Équilibrage du jeu.

4.2 Dimitri

- Recherche et proposition d'idées pour rendre le Monopoly innovant.
- Développement de la logique de la map et implémentation de la génération de maps aléatoires avec directions des bâtiments.
- Création des tests pour la map et implémentation des fonctions contraintes.
- Travail sur la case téléport et gestion des événements de destruction des bâtiments par catastrophes naturelles.

4.3 Jules

- Agencement des classes et création des classes Bonus, Building, Case, et Player.
- Développement de la logique du jeu, y compris la gestion des tours de jeu et des déplacements des joueurs.
- Création de nouvelles classes Coordinates et Directions.
- Amélioration des catastrophes.

4.4 Nicolas

- Mise en place du diagramme des classes et développement des contrôleurs.
- Intégration de la logique du jeu avec les contrôleurs et l'interface graphique.
- Bases pour la case Teleport.

4.5 Sekou

- Création du dépôt GitLab et mise en place de Gradle et de JavaFX.
- Développement de l'interface utilisateur et stylisation selon une charte graphique.
- Reliure de l'interface graphique avec la logique du jeu et les contrôleurs.
- Ajout de cases bonus, gestion des cartes et implémentation des sprites pour les bâtiments et les joueurs.
- Implémentation des observers pour la logique du jeu.

CHAPITRE 5

Problèmes rencontrés et solutions

L'atout principal de notre jeu, est également la principale difficulté que nous ayons rencontré.

Pour éviter de faire un jeu redondant nous avons eu l'idée de faire une carte générée aléatoirement à chaque partie, nous avons eu une première approche en mettant une carte remplis de building et en faisant des "trous" avec des routes, on part du centre puis on choisit une direction aléatoirement et on répète l'opération jusqu'à une valeur prédéfini (8 de distance avec 4 retour au centre par exemple). Cela a posé problème car nous retrouvions des impasses ce qui empêchait de faire une carte jouable et globalement le rendu n'était pas correcte deux tiers du temps.

Pendant que Dimitri programme cette fonctionnalité, lorsqu'il partageait ses difficultés avec le groupe, Jules travaillait simultanément sur la même fonctionnalité, mais avec une idée légèrement différente. Nous conservons le concept de base d'une carte remplie de bâtiments, mais cette fois, nous allons découper aléatoirement la carte avec des routes droites, horizontales et verticales. Ensuite, en imposant bien sûr des contraintes pour éviter des impasses et des blocs trop massifs, nous replaçons des bâtiments dans les segments de route afin de créer des chemins harmonieux.

CHAPITRE 6

Conclusion

Conclusion

Notre projet de Monopoly a permis de développer une version numérique du jeu de société classique avec de nombreuses améliorations et innovations. Les objectifs principaux ont été atteints.

— **Application des concepts appris en cours :** Le développement de ce projet a permis de mettre en pratique les concepts et techniques étudiés, tels que la programmation orientée objet, l'architecture MVC, la gestion de projet, et les principes de conception logicielle.

— **Développement de compétences techniques :** En utilisant Java et JavaFX, on a pu renforcer nos compétences en programmation, en conception d'interfaces utilisateur.

— **Travail d'équipe et gestion de projet :** La réalisation de ce projet exige une organisation efficace du travail d'équipe et une communication constante.

— **Innovation et créativité :** En revisitant le jeu Monopoly avec des éléments innovants et des mécaniques de jeu modernes, ce projet a encouragé la créativité et l'innovation.

Cependant, il existe des axes d'amélioration :

— multijoueur et réseaux

— effets sonores

— code plus modulaire et structuré

— plateau de jeu avec différents thèmes

— intégration de nouveaux thèmes et cartes

— ajout de mini-jeux