Rapport Final Projet Monopoly

Rim GHERMAOUI — Badmavasan KIROUCHENASSAMY

June 2021

1 Introduction

Dans le cadre de ce projet en langage orienté objet Java, nous avons été amenés à implémenter la simulation d'un jeu inspiré du Monopoly. Ainsi, nous avons fourni en début de projet un rapport contenant le diagramme UML représentant la modélisation que nous avions imaginé pour cette implémentation. Etant donné que nous n'avions pas encore commencé à coder le contenu des classes, notre diagramme UML a évolué tout au long de la durée du projet. En effet, il y a plusieurs paramètres que nous n'avions pas pris en compte au début et dont nous nous sommes aperçus au fur et à mesure. De nouvelles méthodes ainsi que de nouveaux attributs se sont ajoutés tandis que d'autres ont disparu pour laisser place à d'autres qui nous ont semblé plus pertinentes. Nous allons donc parcourir toutes les modifications qui ont été effectuées pour justifier nos modifications.

2 Ancien et Nouveau Diagramme UML

Les deux version sont dans le directory diagram uml.

3 Choix Arbitaire

Durant la réalisation de ce projet, nous avons du faire des choix arbitraires. Tout d'abord, nous avons choisi de déclarer plusieurs valeurs de la variable en dur ce qui rend le programme non entièrement dynamique. Comme nous l'expliquerons plus tard dans le rapport, plusieurs modifications ont été faites et ont été dus justement à ces choix réalisés par nous.

4 Modifications

Tout d'abord, la principale différence que nous pouvons remarquer directement est la présence de packages dans le nouveau modèle. En effet, vu la quantité de fichiers java, nous avons vite vu que nous allions avoir besoin de packages afin de se retrouver plus facilement dans toutes ses lignes de codes. Nous avons

donc créé 8 packages à savoir casePackage, configurationPackage, etatPackage, exceptionPackage, jeuPackage, joueurPackage, plateauPackage et testPackage. Certains d'entre eux ne contiennent qu'un seul fichier java mais nous avons quand même créé un package pour le mettre dedans pour que ce soit plus propre. A présent, parcourons package par package pour parler des changements que nous avons opéré.

Le premier package auquel nous allons nous intéresser est le package **joueurPackage**. Celui-ci regroupe les fichiers qui implémentent les différentes classes des joueurs à savoir la classe abstraite Joueur, les classes JoueurPrudent et JoueurAgressif qui sont les deux types de joueurs que nous avons dans le jeu et la classe joueurs qui fait la création des joueurs. Nous avons ajouté des getters à la classe abstraite afin de pouvoir récupérer toutes les valeurs de ses attributs, l'attribut position qui nous permet de récupérer l'indice de la case où se trouve le joueur et la fonction deduct qui est à l'opposé de créditer(). Parallèlement, nous avons mis une fonction removeInvestissement qui permet de supprimer une CaseInvestissement à la liste des investissements des joueurs si celui-ci est amené à en laisser. De plus, la fonction movePlayerTo a vu le jour et celle-ci permet de déplacer le joueur sur la prochaine case en modifiant son attribut position tout en tenant compte du caractère rond du plateau. Enfin, il y'a les deux fonctions getMinInvestissement et getMaxInvestissement qui permettent de retourner la case Investissement qui a respectivement soit la valeur minimale ou maximale, qui seront utilisées lors des passages sur les cases loi antitrust. En ce qui concerne les classes JoueurPrudent et JoueurAgressif, celles-ci ont connu les mêmes changements puisque nous avons supprimé les fonctions actionInvestissement et actionAntitrust car nous avons changé notre manière d'implémenter et nous avons décidé de créer une nouvelle classe configurationJeu qui contient toutes les informations relatives à la réaction des joueurs selon leur type face aux cases dans lesquelles ils se trouvent.

Quant au package **jeuPackage** qui contient la classe Simulation, celle-ci a connu un grand nombre de modifications aussi. Notons que cette fonction est la classe principale du projet, c'est elle qui contient le main. Nous avons supprimé les attributs listJoueurs et plateau puisque la liste des joueurs est à présent dans la classe qui est dédiée à sa création à savoir joueurs et le plateau est initialisé à l'intérieur du main. De même, l'action initjoueurlist a été supprimée pour les mêmes raisons. L'action configurationJeu a disparu pour se transformer en une classe à part entière qui contient tout ce qui est relatif à l'état du jeu à l'instant t comme le nombre de joueurs agressifs/prudents avec les getters associées à ses attributs. Ainsi, nous avons ajouté les fonctions tocontinue et checkEndOfGame qui renvoient des booléens et qui sont réutilisés dans le main de la même classe. Aussi, il y'a la fonction menu qui permet de faire l'interaction avec la personne qui lance le jeu afin de lui demander quelle est la configuration souhaitée et renvoie donc configurationJeu. De même, nous avons une fonction qui permet l'affichage de l'état du jeu au moment où elle est appelée, il s'agit de printjeu. Nous avons aussi eu besoin de rajouter getRandom pour le lancer de dés et removeJoueurPerdu pour retirer un joueur de la liste de joueurs et le mettre dans celle qui regroupe ceux qui ont perdu.

En ce qui concerne le package **casePackage**, celui-ci n'a pas connu beaucoup de modifications au niveau du squelette mis à part la suppression des setters qui ont été remplacés par des constructeurs et l'ajout de getters et l'action investissementBackToEtat dans la classe CaseInvestissement.

Dans le package **plateauPackage**, son unique classe n'a connu aucune modification.

Dans le package **exceptionPackage**, nous avons ajouté de nouvelles exceptions qui sont générés dans les cas leur correspondant.

Dans le package **etatPackage**, la notion de créditer et donc déduire de l'argent liquide et des investissements a été ajoutée grâce aux fonctions créditer, deduct, addToInvestissement et removeInvestissement.

La classe et le package **configurationJeu** et **configurationPackage** n'existaient pas. La classe a été créée afin de regrouper le constructeur ainsi que les getters de tout ce qui est relatif à comment est configuré le jeu contenant le nombre de joueurs agressifs et prudents, le capital de départ des joueurs, le capital de l'état, le profil ainsi que la liste contenant le max d'investissements de chaque joueur prudent qui sera utile pour l'implémentation de la création des joueurs prudents.

5 Mode d'utilisation

Le fichier Monopoly.java contient le main donc il suffit de lancer ce fichier sur eclipse pour tester le jeu. Tous les tests sont dans le package testPackage qui se divise en plusieurs parties en fonction de leur nature.

6 Améliorations Possibles

En guise de pistes d'amélioration, la création du plateau aurait pu être plus dynamique, puisqu'elle l'est pas entièrement. En effet, nous avions pris la décision de laisser des variables déclarées en dur puisque le fait de le dynamiser mènera à la nécessité de plusieurs variables en paramètre lors de la création de celui-ci. Une possibilité serait de fournir les minmums et les maximums en dur et faire en sorte que le programme calcule les pourcentages et les valeurs par lui-même tout en construisant le plateau. Egalement, nous aurions pu éviter d'utiliser 3 listes dans la classe Simulation pour n'en utiliser que deux mais nous avons opté pour une lisibilité du code qui est meilleure avec 3 listes plutôt que deux. Néanmoins, il s'agit d'une possibilité.

On n'a pas fait beaucoup de tets. Comme on devait faire des changements au niveau de la simulation a la derniere minute, on n'a pas eu le temps de le faire.

Le facon dont on a construit la classe simulation complique le facon de faire les tests. DOnc on n'a pas fait le test d'action de chaque case.

7 Conclusion

Pour conclure, nous pouvons commencer par dire que ce projet nous a beaucoup apporté. En effet, il est beaucoup plus évident de voir l'utilité de ce que nous apprenons en programmation quand nous l'utilisons sur des sujets concrets. Le langage Java étant un langage très utilisé dans le monde de l'IT, nous avons trouvé le projet super intéressant pour nous permettre de voir ce que nous pouvons faire concrètement avec nos connaissances. D'autant plus que le fait que coder un jeu est toujours plus sympathique que coder une gestion de banque ou encore de bibliothèque comme nous avons pu le faire en TP. Nous avons trouvé quelques difficultés quand il a fallu supprimer tous les "instance of" comme demandé par les encadrants.