Rapport première itération TOB

April 2023

Leo Flamencourt, Téo Pisenti, Frederique,Cylia Yangour, Alexis Courboin,
Emile Devos.

Contents

1 Présentation générale du projet

L'objectif de ce projet est d'appliquer les notions de Technologie Objet vues en cours grâce à la mise en oeuvre notre vision du jeu du Monopoly, Oligopol7, qui se joue directement depuis un ordinateur. Il permettra à l'utilisateur de jouer contre d'autres utilisateurs en multijoueurs sur un même écran, au sein d'un terrain 2D isométrique et une interface graphique vivante, surprenante et dynamique. Il mettra notamment en scène la ville de Toulouse et fera références à certains lieux emblématiques du centre ville et de l'ENSEEIHT.

2 Application des méthodes agiles

En suivant les notions vues en méthode agiles, nous avons pris le choix de rendre, pour cette première itération, une version très simplifiée de notre Monopoly, sans interface graphique. Le but étant de se concentrer d'abord sur le "fond" du projet, en répartissant les fonctionnalités sur différentes classes et implémantations élémentaires, afin de poser les bases et la structure initiale du jeu.

Nous avons donc réparti le travail suivant les fonctionnalités ; chaque personne/groupe a géré un aspect particulier (joueur, plateau, arbitre) en gérant si possible les tests associés.

— en plus d'avoir simplifié le problème, on a séparé les tâches avec la méthode agile: on a séparé le travail en plusieurs users-stories que chacun a implémenté.

3 Explication des fonctionnalités

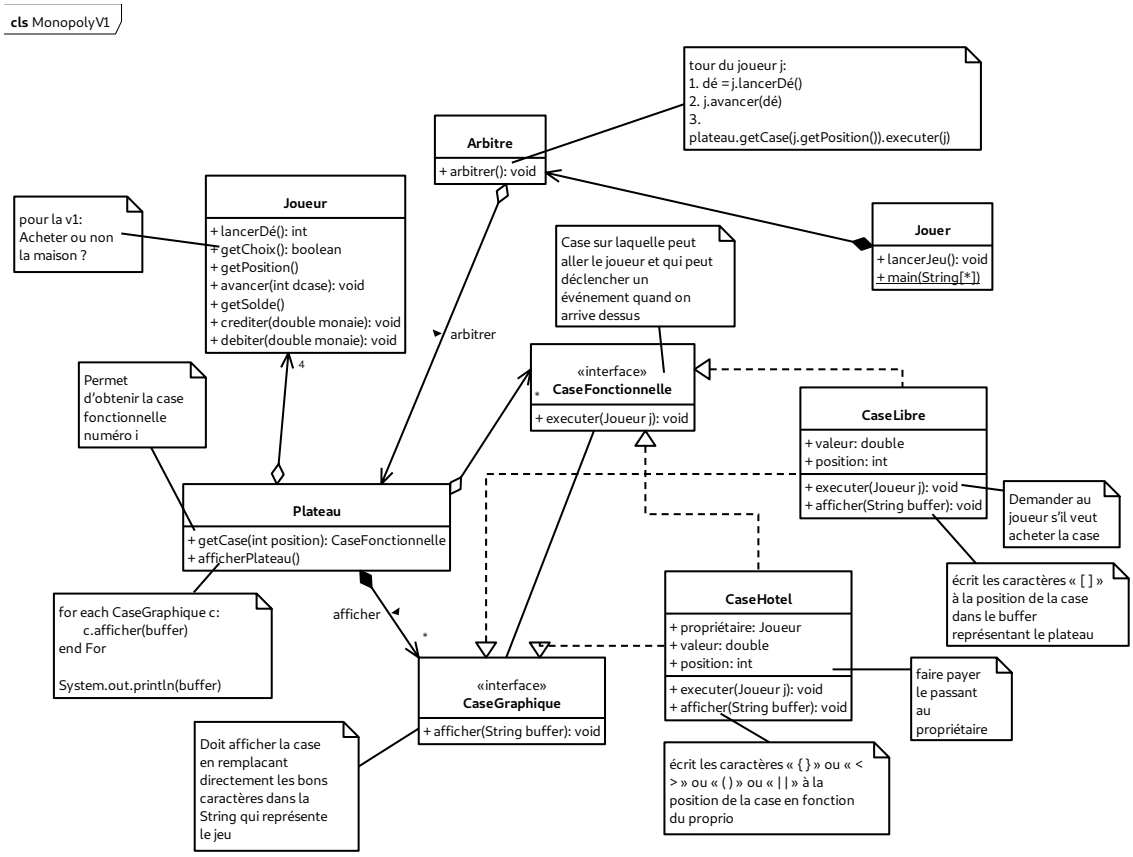
getchoix getposition afficher ajoutercase getcase

fonctionnalités:

- gestion des tours (arbitre) -¿ Epics – lancer le dé (joueur) – avancer le joueur (joueur) – action de la case (caseFonctionnelle) — achat maison si libre et joueur veut (caseLibre) -¿ users story — ne rien faire si veut pas acheter (caseLibre) — payer joueur si maison appartient à autre joueur (caseHotel) — ne rien faire si la maison nous appartient (caseHotel) – fin du tour (arbitre) - changer de joueur automatiquement (arbitre) - arrêter jeu si un joueur n'a plus d'argent (banquerouteException) - afficher les règles (jouer) - lancer la partie (jouer) - afficher les joueurs (plateau) - afficher le plateau (plateau) – afficher une case (caseGraphique) – afficher l'appartenance d'un hotel (caseHotel) – afficher qu'une case est libre (caseLibre)

4 Diagramme de classe (UML)

Ci-dessous notre diagramme de classe UML (premier "brouillon" puis UML final) :

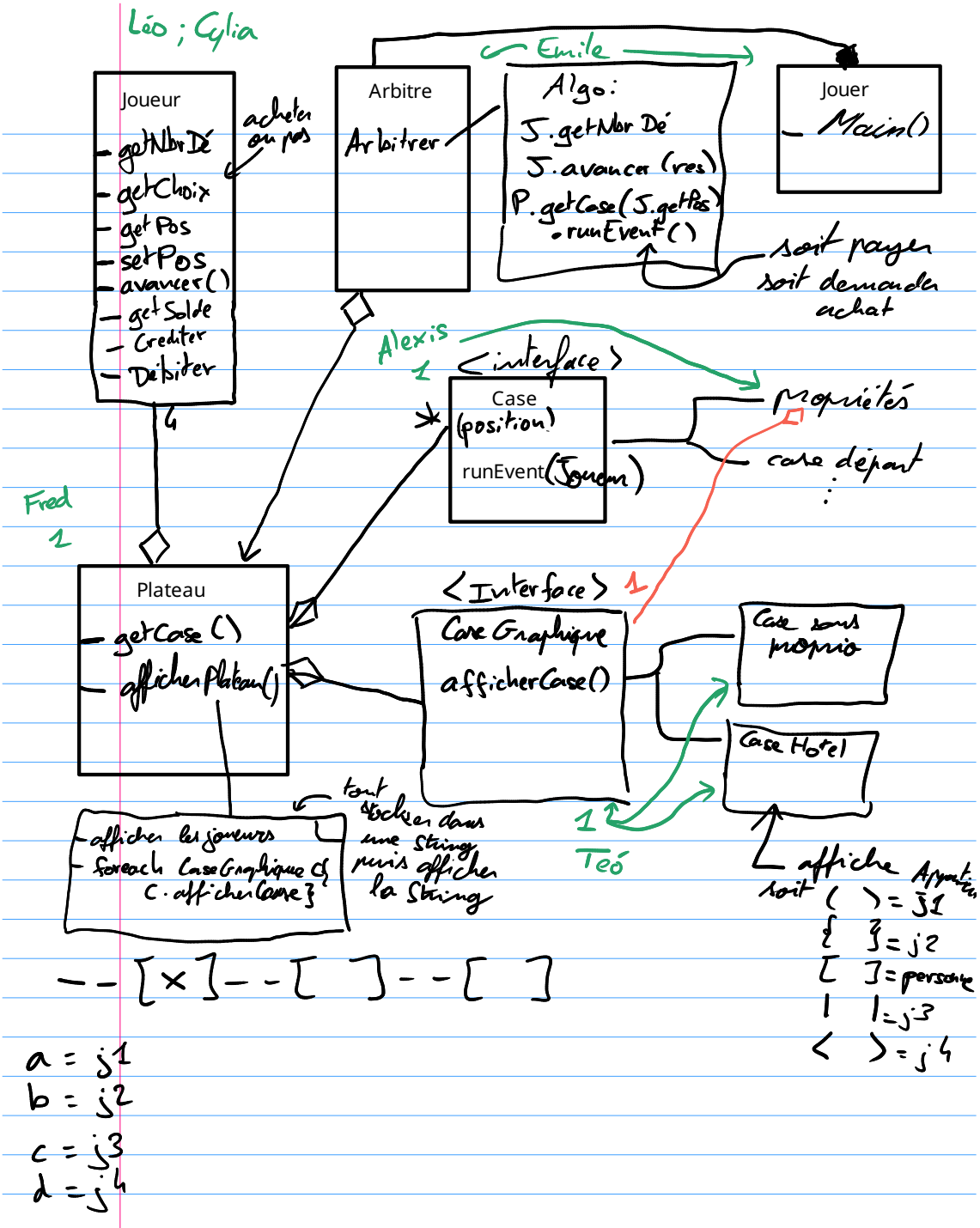


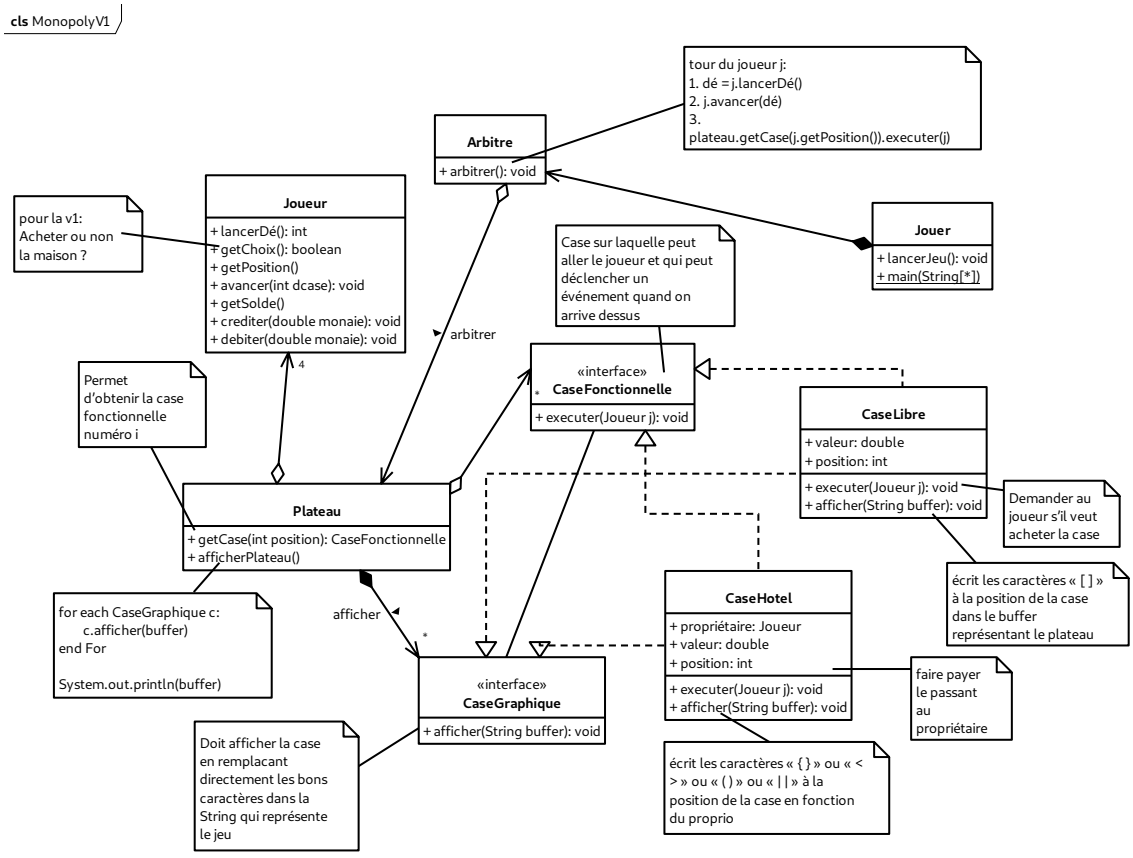
5 Principaux choix de conception

description succinte des classes (chacun décrit sa classe) -¿ voir la javadoc en en-tête des classes

Arbitre = gestion du tour

Joueur = toutes les méthodes qui s'appliquent sur un joueur


Léo ; Cylia
Joueur
getNbrDé
getChoix
getPos
setPos
avancer()
getSolde
Créditer
Débiter
acheter ou pas
Arbitre
Arbitrer
Emile
Algo:
J.getNbrDé
J.avancer(res)
P.getCase(J.getPos)
.runEvent()
Jouer
Main()
soit payer
soit demander achat
Alexis
1
<interface>
Case
(position)
runEvent(Joueur)
propriétés
case départ
Fred
1
Plateau
getCase()
afficherPlateau()
<Interface>
1
CaseGraphique
afficherCase()
Case sans propio
Case Hotel
tout stocker dans une String puis afficher la String
afficher les joueurs
foreach CaseGraphique cG
C.afficherCase()
1
Teó
affiche
soit ( ) = j1
{ } = j2
[ ] = personne
| | = j3
< > = j4
- - [x] - - [ ] - - [ ]
a = j1
b = j2
c = j3
d = j4


cls MonopolyV1
tour du joueur j:
1. dé = j.lancerDé()
2. j.avancer(dé)
3.
plateau.getCase(j.getPosition()).executer(j)
Arbitre
+ arbitrer(): void
Joueur
+ lancerDé(): int
+ getChoix(): boolean
+ getPosition()
+ avancer(int dcase): void
+ getSolde()
+ crediter(double monaie): void
+ debiter(double monaie): void
pour la v1:
Acheter ou non
la maison ?
Jouer
+ lancerJeu(): void
+ main(String[*])
Case sur laquelle peut
aller le joueur et qui peut
déclencher un
événement quand on
arrive dessus
arbitrer
4
«interface»
CaseFonctionnelle
+ executer(Joueur j): void
Permet
d'obtenir la case
fonctionnelle
numéro i
CaseLibre
+ valeur: double
+ position: int
+ executer(Joueur j): void
+ afficher(String buffer): void
Demander au
joueur s'il veut
acheter la case
Plateau
+ getCase(int position): CaseFonctionnelle
+ afficherPlateau()
écrit les caractères « [ ] »
à la position de la case
dans le buffer
représentant le plateau
for each CaseGraphique c:
c.afficher(buffer)
end For
System.out.println(buffer)
afficher
*
CaseHotel
+ propriétaire: Joueur
+ valeur: double
+ position: int
+ executer(Joueur j): void
+ afficher(String buffer): void
faire payer
le passant
au
propriétaire
«interface»
CaseGraphique
+ afficher(String buffer): void
Doit afficher la case
en remplacant
directement les bons
caractères dans la
String qui représente
le jeu
écrit les caractères « { } » ou « <
> » ou « ( ) » ou « | | » à la
position de la case en fonction
du proprio