Oligopoly7

Rapport troisème itération

Leo Flamencourt, Téo Pisenti, Frederic Camail ,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 deuxième itération, une version plus poussée de notre Monopoly, avec l'ajout de fonctionnalité et une première version de l'interface graphique. Le but étant de préciser certains aspect du projet, ainsi que d'améliorer les classes et fonctionnalités, en s'approchant le plus possible des "véritables" règles du Monopoly. Ces ajouts se sont organisés sur le git, sous forme d'issues, que nous nous sommes répartis lors qu'une réunion.

Nous avons donc réparti le travail suivant ces nouveaux ajouts ; chaque personne/groupe de personne a géré un aspect particulier (interface graphique, menu, amélioration des fonctionnalités, adaptation/évolution des classes initiales).

3 Fonctionnalités traitées

3.1 Fonctionnalités traitées durant l'itération 1

Le but de cette itération a été de rendre une première version du Monopoly jouable uniquement en ligne de commande.

Toutes ces fonctionnalités de l'itération 1 correspondent au package "logique-Monopoly" dans le code final.

1. Gestion des tours : (Arbitre.java)
    - Lancer le dé (Joueur.java)
    - Avancer le joueur (Joueur.java)
    - Action de la case (CaseFonctionnelle.java)
        - Achat maison si libre et joueur veut (CaseLibre.java)
        - Ne rien faire si veut pas acheter (CaseLibre.java)
        - Payer joueur si maison appartient à autre joueur (CaseHotel.java)
        - Ne rien faire si la maison nous appartient (CaseHotel.java)
    - Fin du tour (Arbitre.java)

2. Changer de joueur automatiquement (Arbitre.java)
3. Arrêter jeu si un joueur n'a plus d'argent (BanquerouteException.java)
4. Afficher les règles (Jouer.java)
5. Joueur la partie (Jouer.java)
6. Choisir le nom des joueurs avec les arguments de la ligne de commande (Jouer.java)
7. Afficher les joueurs (Plateau.java)
8. Afficher le plateau (Plateau.java)
    - Afficher une case (CaseGraphique.java)
    - Afficher l'appartenance d'un hotel (CaseHotel.java)
    - Afficher qu'une case est libre (CaseLibre.java)

4 Fonctionnalités traitées durant l'itération 2

Le but de cette itération a été la création de l'interface graphique en parallèle de l'ajout de fonctionnalités dans le package "logiquemonopoly" (séparation du groupe en 2 équipes différentes).

Le résultat a été 2 exécutables différents pour lancer le Monopoly non graphique (avec les fonctionnalités supplémentaires) et l'interface graphique seule (pour voir les éléments d'interface et cliquer dessus).

1. Interface graphique (package interfaceGraphique)
    - Afficher le plateau au centre de l'écran (PlateauGraphique.java)
    - Afficher une pop-up au centre de l'écran avec un choix "oui/non" pour acheter un terrain (PopupAchat.java)
    - Afficher une pop-up avec un seul choix possible "ok" pour afficher des messages généraux (cartes chances par exemple) (Popup.java)
    - Afficher un dé en bas à gauche de l'écran (BoutonDe.java)
    - Afficher un bouton pour ouvrir le menu d'amélioration des propriétés à droite de l'écran (BoutonPropriétés.java)
    - Afficher un scoreboard intéractif (joueurs triés par argent décroissant, mise à jour des scores) au haut à droite de l'écran (JoueurScore.java et InterfaceGraphique.java)
    - Voir le joueur en train de jouer son tour sur le scoreboard (couleur différente) (JoueurScore.java)
    - Afficher un bouton "fin du tour" en bas à droite de l'écran (BoutonFinTour.java)

- Repositionner tous les éléments de l'interface quand on redimensionne la fenêtre du jeu (réaliser l'interface WindowListener)
- Afficher un bouton pour ouvrir les paramètres et activer ou non le son (partiellement traité)

2. Ajout d'événements dans le jeu (package logiqueMonopoly)

   - Implémenter un code pour l'enchère d'une propriété : si un joueur refuse d'acheter une propriété, elle est mise aux enchères. N'importe quel autre joueur peut alors enchérir les uns après les autres et celui qui met le tarif le plus élevé gagne la propriété. (CaseLibre.java, partiellement traité)
   - On peut tirer des "Cartes Chances" qui réalisent un événement aléatoire (CarteChance.java)
   - Ajout de "cases chances" qui font tirer au joueur une carte chance s'il s'arrête dessus (CaseChance.java)

4.1 Fonctionnalités traitées durant l'itération 3

Le but de cette itération a été l'ajout du "liant" entre les packages "logiquemonopoly" et "interfaceGraphique" pour rendre le jeu totalement fonctionnel en mode graphique (et quelques animations).

Ces fonctionnalités sont à la fois traitées dans le package "logiquemonopoly" et "interfaceGraphique".

1. Pouvoir modéliser et ajouter dans le plateau graphique différents types de cases (CaseGraphique.java)
2. Afficher sur le plateau graphique les cases de propriétés des joueurs et leur état (libre ou achetée) (CaseProprieteGraphique.java)
3. Afficher sur le plateau graphique les cases chances et jouer une animation quand un joueur tombe dessus (CaseChanceGraphique.java)
4. Afficher sur le plateau le pion de chaque joueur (Pion.java)
5. Pouvoir faire avancer le pion d'un joueur graphique de n cases (Pion.java)
6. Suivre en temps réel le pion d'un joueur pendant son déplacement avec la caméra de la grille isométrique (Pion.java)
7. Jouer une animation quand on clique sur le dé (secoue le dé et affiche le numéro tiré) (BoutonDe.java)
8. Bloquer le clic du joueur s'il a déjà tiré le dé, où s'il ne peut pas encore finir son tour et afficher l'état des boutons (actif/inactif) en changeant l'image ou la couleur associée au bouton (BoutonDe.java, BoutonFinTour.java)

9. Attendre que l'utilisateur clique sur le dé, réponde à une pop-up, clique sur le bouton de fin du tour dans la boucle du jeu (Arbitre.java)
10. Récupérer la réponse de l'utilisateur à la Popup d'achat (PopupAchat.java)
11. Lier une case fonctionnelle du modèle à une case graphique de la vue (CaseFonctionelle.java, CaseGraphique.java)

4.2 Fonctionnalités non traitées, laissées à une itération future

- Ajouter un menu pour choisir le nom des joueurs graphiquement et la somme de départ (et éventuellement le nombre d'IA)
- Laisser le choix à chaque joueur entre plusieurs styles graphique de pion
- Afficher d'une couleur différente les propriétés de chaque joueur
- Afficher un menu principal : un menu est montré au lancement de l'application. Il permet de lancer une partie, reprendre une partie en cours, voir les crédits.
- Sauvegarder une partie pour la reprendre ultérieurement
- Améliorer les propriétés : une propriété peut être améliorée à tout moment depuis le menu des propriétés grâce à son nom (ajouts d'hôtels, etc)
- Hypothéquer une propriété : une propriété peut être hypothéquée à tout moment (pendant son tour) depuis le menu des propriétés. Le joueur récupère alors la somme de l'hypothèque mais ne gagne plus d'argent si d'autres joueurs tombent sur sa propriété hypothéquée. Il peut racheter cette hypothèque pour 10% de plus depuis le même menu pour revenir à l'état antérieur.
- Aller en prison : ajout d'une case qui envoie un joueur en prison. Celui-ci ne passe pas par la case départ. Ajout d'une carte chance qui envoie en prison et une carte chance "sortir de prison" qui est la seule carte que le joueur peut garder et le rend "immunisé" à la prison.
- ajouter une IA avec des stratégies basiques (tout acheter, réagir aléatoirement. . .)

5 Architecture de l'application

Notre application est découpée en 3 principaux packetages :

1. logiquemonopoly : il s'agit du modèle de notre application, il contient toutes les classes qui participent à la logique interne du Monopoly. C'est lui qui s'occupe de la gestion des tours, de la gestion des événements qui arrivent aux joueurs...

2. interfaceGraphique : il s'agit de la vue de notre application, il contient toutes les classes qui constituent des éléments d'interfaces : boutons cliquables, scores des joueurs, pop-up, le plateau isométrique, les éléments du plateau

3. moteurGraphique : il s'agit d'une bibliothèque graphique entièrement réalisée par Téo Pisenti à l'aide de la bibliothèque LWJGL (bibliothèque haute performance qui fournit l'accès à l'API bas niveau de la carte graphique : Opengl). Il fournit toutes les primitives essentielles à la réalisation d'une interface graphique (rectangles, boites de texte, images, grille en vue isométrique dans le sous-package moteurGraphique.Drawable), et les types nécessaires à la création d'une interface intéractive et adaptative (Button et WindowListener dans le sous-package moteurGraphique.window). C'est également lui qui s'occupe de la boucle de rendu graphique de la fenêtre (moteurGraphique.OpenglThread). Des performances "temps-réel" sont assurées par l'utilisation de l'accélération de la carte graphique via l'api "Opengl Core 3.3"

5.1 Architecture de "logiquemonopoly"

Vous trouverez ci-dessous le diagramme de classe du modèle de notre Monopoly. Plusieurs méthodes et attributs ont été occultés pour ne garder que les méthodes essentielles à la compréhension. Veillez vous référer au code source pour en savoir davantage.

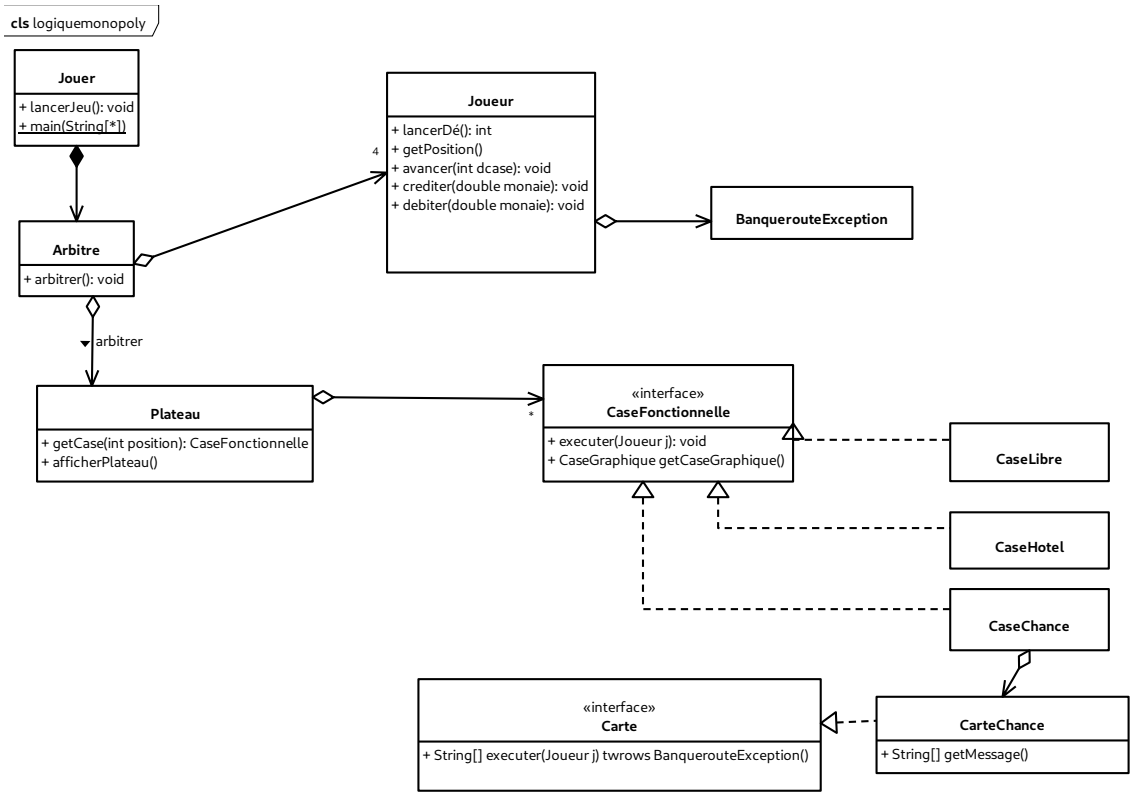

Figure 1: Diagramme de classe de la logique du Monopoly

Éléments principaux :

- Jouer : lecture de la ligne de commande et lancement de la partie
- Arbitre : gestion du système de tours du jeu. Boucle principale de gameplay
- Joueur : un joueur de la partie, avec son nom, son solde
- BanquerouteException : exception vérifiée déclenchée quand on débite à un joueur plus que le solde qu'il possède
- Plateau : Liste de cases ordonnées que le joueur peut parcourir dans l'ordre quand il avance pendant son tour
- CaseFonctionnelle : implémentation d'un patron "Stratégie" pour définir différents types de cases du plateau qui déclenchent des actions quand le joueur tombe dessus
- CaseLibre : propriété achetable qui n'appartient à personne

- CaseHotel : propriété achetée par un joueur, les autres joueurs lui donneront de l'argent en tombant dessus
- CaseChance : tire une carte chance qui peu avoir des effets sur la partie
- CarteChance : lance un événement aléatoire si tirée
- Carte : généralisation d'un carte chance pour un ajout futur de nouveaux types de cartes (exemple : carte immunité à la prison)

5.2 Architecture "interfaceGraphique"

Ce package étant relativement complexe, on ne donnera ici que des diagrammes d'exemples qui serviront de modèles pour l'ensemble des classes de ce package (elles sont en effet toutes très similaires).

Nous rappelerons tout de même brièvement les différents éléments d'interface listés dans ce package dans la figure suivante :

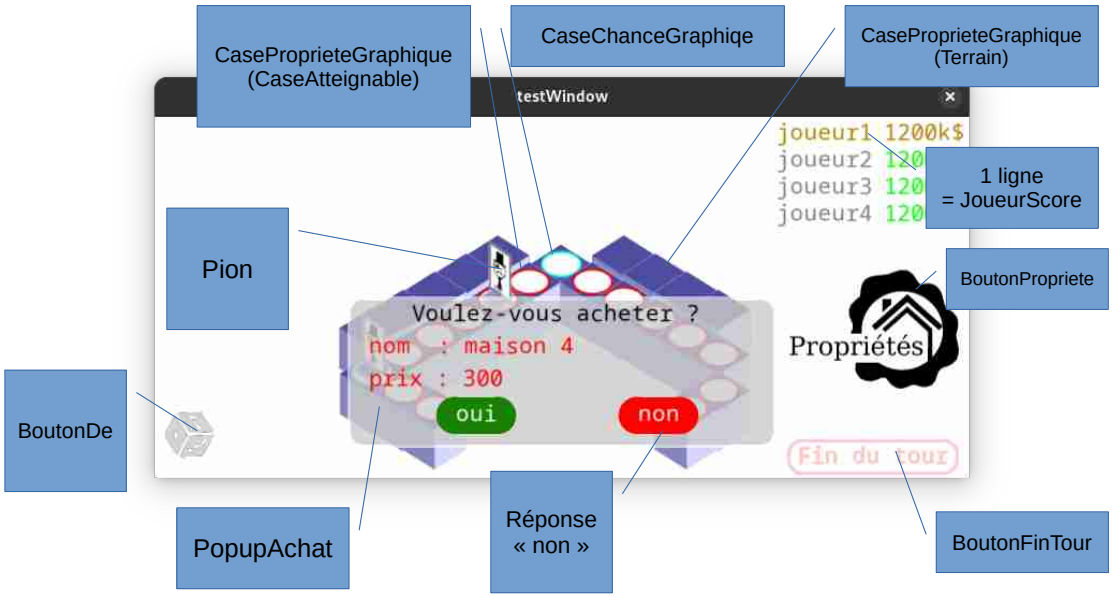

Figure 2: Noms des classes associés à chaque élément de l'interface

On présente ci-dessous le diagramme de classe de "PopupAchat.java" qui sert de modèle à tous les boutons de l'interface.

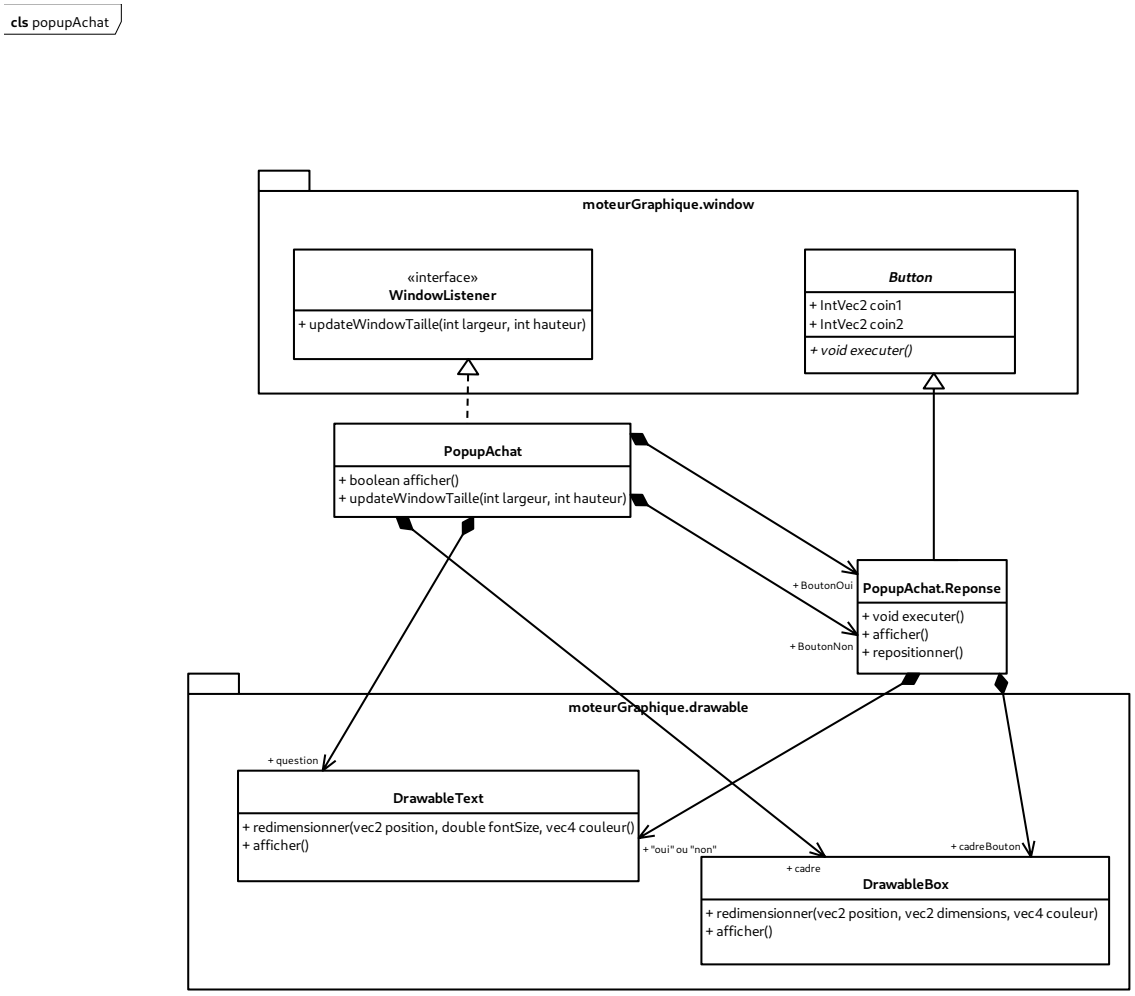

Figure 3: Diagramme de classe de la pop-up d'achat d'une propriété

Voici quelques éléments de compréhension du diagramme de classe :

- moteurGraphique.window est le sous-package du moteur graphique qui donne accès aux éléments d'interactivité de la fenêtre de l'application
- moteurGraphique.drawable est le sous-package du moteur graphique qui donne accès aux primitives graphiques élémentaires (rectangles, texte, image)
- WindowListener : interface qui permet d'être informé lorsque la taille de la fenêtre change (la fonction updateWindowTaille est automatiquement appelée quand l'utilisateur redimenssionne la fenêtre)

- Button : classe abstraite qui permet de rendre un élément cliquable. La zone de détection du clic de la souris est rectangulaire et définie par les deux attributs de type "protected" coin1 et coin2 (respectivement les coins suppérieur gauche et coin inférieur droit de la zone de clic du bouton). Ils doivent être mis à jour quand on redimenssionne le bouton. La fonction "executer" est automatiquement appelée au clic sur le bouton.

- DrawableText et DrawableBox sont respectivement une ligne de texte et un rectangle coloré avec bordures arrondies. Ils disposent d'une méthode afficher pour l'afficher à l'écran (le dernier affiché est au premier plan), et d'un méthode "redimensionner" pour changer leur position et taille à l'écran (utile quand la taille de la fenêtre change). Il existe également DrawableImage qui est très similaire et permet d'afficher une image (utilisé pour le BoutonDe)

- PopupAchat : code principal de la popup, la méthode "afficher" bloque l'exécution jusqu'à que l'utilisateur aie cliqué sur "oui/non". On réalise "WindowListener" pour recentrer la pop-up quand on redimensionne la fenêtre.

- PopupAchat.Reponse : bouton "oui" ou bouton "non" qui hérite de bouton pour qu'on puisse cliquer dessus

On présente également le diagramme de classe du plateau graphique avec l'exemple de "CaseProprieteGraphique" qui modélise graphiquement une propriété d'un joueur.

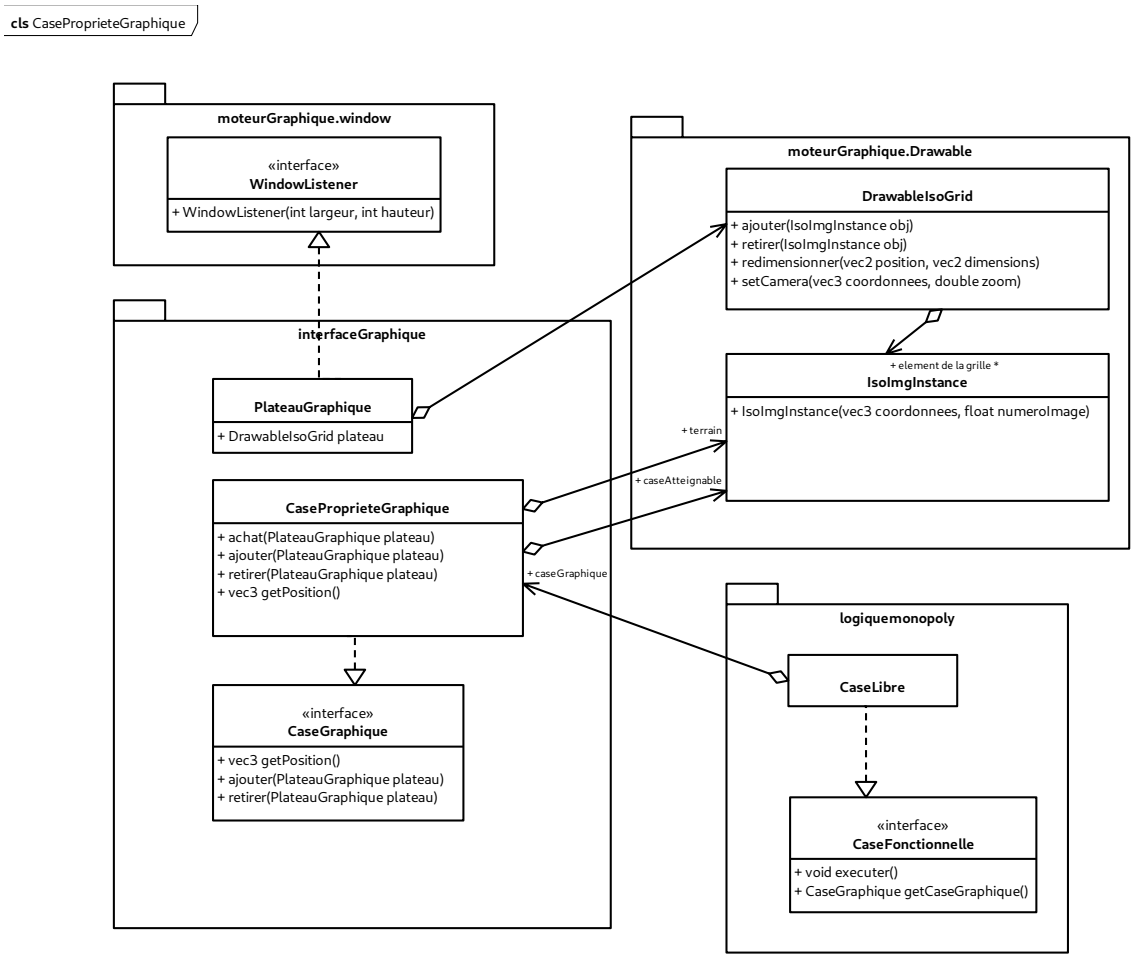

Figure 4: Diagramme de classe d'une propriété du joueur affichée à l'écran

Quelques éléments de compréhension du diagramme :

- DrawableIsoGrid : grille isométrique avec son propre repère à 3 dimensions. Fait la conversion entre les coordonnées à 3 dimensions dans la grille et les coordonnées à 2 dimensions à l'écran. Contient également l'image de référence (voir res/isoTex.png) qui contient toutes les images qui peuvent être placées dans la grille. Cet objet possède aussi une caméra et un zoom pour choisir l'origine du repère de la grille et ainsi centrer un élément (utilisé pour suivre les pions)
- IsoImgInstance : couple contenant le vecteur de coordonnées dans le repère en 3D de la grille isométrique et l'indice de la sous-image utilisée parmi l'image de référence de la grille (l'image de la grille sera automatiquement

découpée pour choisir uniquement l'image sélectionnée. Cela permet de n'avoir qu'un seul transfert d'images entre les mémoires cpu/gpu pour de meilleures performances)

- PlateauGraphique : crée un "DrawableIsoGrid" avec toutes les images qui peuvent être affichées sur le plateau (voir "res/isoTex.png") et réalise "WindowListener" pour le garder centré à l'écran et de bonne taille (pour ne pas se chevaucher avec les boutons)

- CaseProprietéGraphique : une propriété est composée de 2 images sur le plateau, celle du terrain (ou l'on peut rajouter une maison ou non) et celle de la "caseAtteignable" qui est la case sur laquelle le pion est placé pour intéragir avec la propriété

- CaseGraphique : permet de généraliser la CaseProprieteGraphique pour avoir plusieurs types de cases graphiques (ex : la case chance graphique, non représentée ici). La fonction getPosition renvoie uniquement les coordonnées de la case sur laquelle on posera le pion du joueur (caseAtteignable) dans le repère de la grille isométrique.

- CaseLibre de la logique du monopoly fait référence à la CasePropriétéGraphique pour qu'elle se mette à jour au moment de l'achat. À ce moment là, la CaseLibre sera remplacé par la CaseHotel dans "logiquemonopoly" qui prendra sa place dans le diagramme