Projet de Programmation Orienté Objet : Mille Bornes

Master Ingénierie des Systèmes Intelligents - MU4RBI01 Python
Sorbonne Université - 2020/2021

Étudiants : DUSSARD Bastien & SIMON Louis

Ce document traite de l'implémentation en Python du jeu *Mille Bornes* dans le cadre d'un projet de fin de semestre. Il contient le diagramme UML ainsi que de brèves explications sur les liens entre les classes et le fonctionnement global du programme et de l'interface graphique.

Contexte :
Le *mille bornes* est un jeu de carte français créé par Edmond Dujardin en 1954. Le principe du jeu est simple ; accumuler 1000 *bornes* (kilomètres) tout en évitant les attaques sournoises des autres joueurs. Le but ultime des mille bornes fait ici référence à la longueur de l'ancienne Route Nationale 7 ou "route des vacances" qui reliait Paris à Menton.
Ci-joint une chronique de l'émission *Karambolage* sur le sujet : `https://www.youtube.com/watch?v=yGBzBWDIKgE`

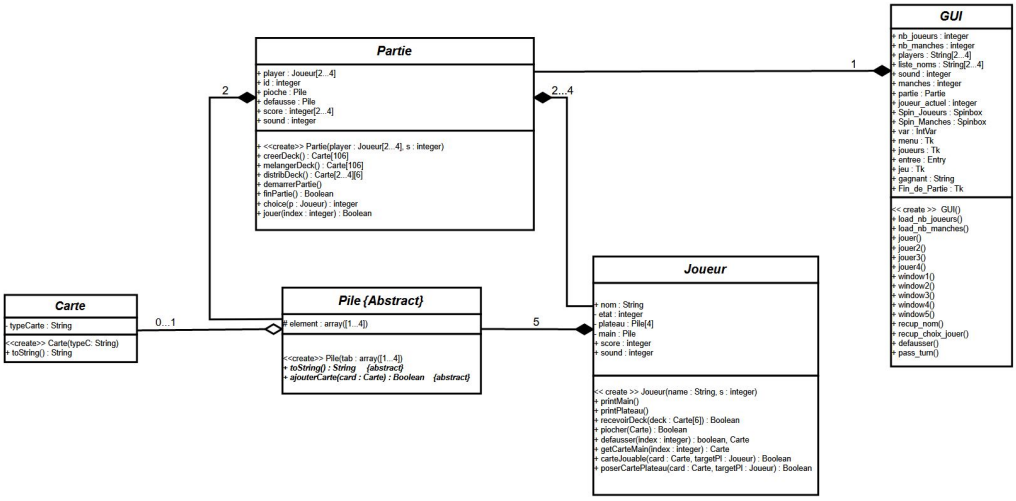
Liens entre les classes :
Le programme Python peut être divisé en quatre classes principales :

- Une classe `Partie` qui modélise la boucle de jeu.
- Une classe `Carte` qui constitue l'objet de base utilisé dans toute les autres classes.
- Une classe `Pile` qui gère différents types d'amas de cartes.
- Une classe `Joueur` qui fait le lien entre l'utilisation des piles et la partie.

Afin de gérer la multitude de cartes différentes, la classe `Carte` est subdivisée en quatre classes filles reliées à la classe mère par une relation d'**héritage**. L'existence d'une classe mère est justifiée par les nombreuses opérations communes à tout les types de cartes (pioche, défausse, mélange ...) qui ne dépendent généralement pas du type. De plus, l'attribut `typeCarte` de la classe mère `Carte` permet une meilleur lisibilité, contrairement à la fonction `isInstance()`, lors des nombreuses disjonctions de cas contenues dans le programme.
La classe `Pile` a elle aussi une relation d'**héritage** avec ses sept classes filles. Chaque pile ayant une fonction et un fonctionnement très spécifique (placement des cartes, contenu, impact sur le joueur) ; il est donc plus agréable de travailler avec plusieurs classes filles héritant d'une classe mère abstraite. La classe mère est abstraite, ce qui permet de définir une seule et même structure pour la fonction abstraite `ajouterCarte()` qui est à la base de toute action dans la partie. La fonction d'affichage `__str__()` est elle aussi abstraite et définie dans chaque classes filles.

La classe `Partie` fait usage des trois autres classes principales notamment pour ses attributs. Une `Partie` est en effet constituée d'une pioche et une défausse, objets héritant de la classe `Pile`, ainsi que d'un ensemble de joueurs. Ces deux relations sont par essence des **compositions** puisque l'on ne créé qu'une partie à la fois et que l'existence des piles et des joueurs en dehors d'une partie est illogique, du moins dans le contexte d'un jeu de société. Il existe enfin un lien d'**agrégation** entre les classes filles de la classe `Carte` et les classes filles de la classe `Pile`. Une fois créées, les cartes peuvent être utilisées dans des piles contenues dans la classe `Partie`, la pioche et la défausse, ainsi que dans la main d'un joueur, type de pile héritant de la classe `Pile` et attribut de la classe `Joueur` (main, bataille, vitesse, borne et botte). Il est néanmoins important de souligner qu'une carte ne peux appartenir qu'à une seule pile à la fois. Le lien d'**agrégation** est alors justifié par la non-appartenance d'une carte à une pile unique au cours de la partie.
Enfin, il existe un lien de **composition** entre la classe `Partie` et la classe `GUI` de jeu contenue dans la classe `Partie`. A noter qu'une partie n'existe que dans une interface graphique ou dans un programme main quelconques qui affiche le jeu dans l'invite de commande, ce qui justifie la relation de **composition**.

UML simplifié



UML détaillé

Les relations entre classes filles de `Carte` et `Pile` et entre les classes filles de `Pile` et `Joueur` étant complexes, il est difficile de les réunir dans un diagramme UML concis. Ces relations sont donc listées ci dessous sous forme de texte :

- `pBataille`, `pMain`, `pBotte`, `pVitesse` et `pBorne` → lien de **composition** avec `Joueur`
- `pPioche`, `pDefausse` → lien de **composition** avec `Partie`
- `Parade`, `Attaque` → lien d'**agrégation** avec `pMain`, `pPioche`, `pDefausse` et `pBataille`
- `Borne`, `Botte` → lien d'**agrégation** avec `pMain`, `pPioche` et `pDefausse`