Partie 2 : Création d'une bibliothèque générique pour les jeux à deux joueurs

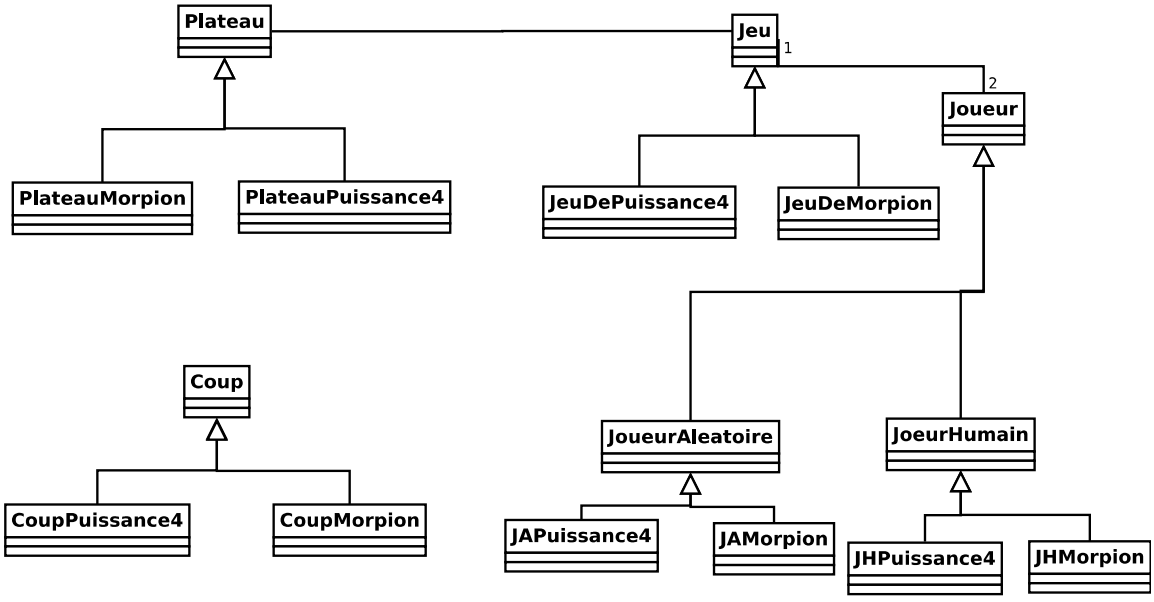
Dans le cadre de ce TP, faites l'analyse demandée sur papier et discutez de votre proposition avec votre encadrant avant de passer à l'implémentation

Exercice 1 : Généralisation à tous les jeux à deux joueurs

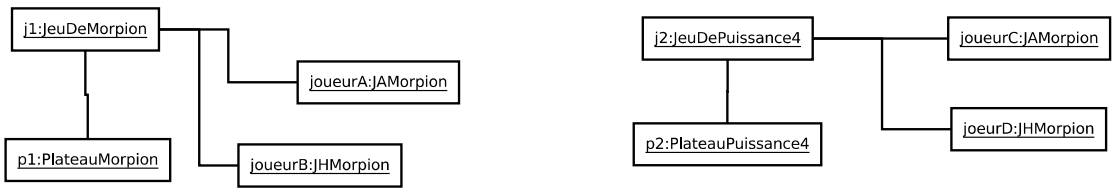
En vous inspirant des diagrammes ci-dessous, proposez votre solution qui reprend l'ensemble des fonctionnalités précédentes :

- quelles sont les classes/fonctions abstraites/concrètes ? (identifier ce qui est général à tous les jeux à deux joueurs)
- Implémentez l'ensemble pour le Jeu de Morpion et le Jeu de Puissance4
- Vérifier le bon fonctionnement
- Validez votre proposition avec votre encadrant

Diagramme de Classes

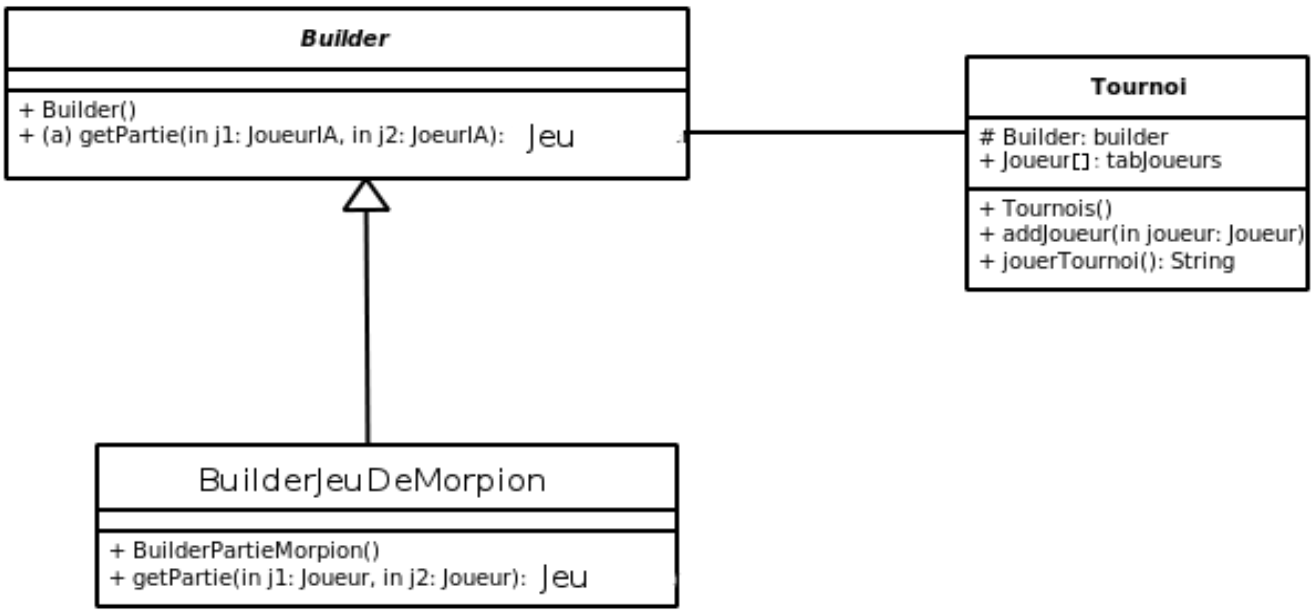


Diagrammes d'Objets



Exercice 2 : Gestion des tournois

La classe Tournoi gère les tournois. Elle utilise un objet utilitaire de construction (Builder), afin de récupérer les parties dont elle a besoin pour exécuter le tournoi pour les joueurs de **tabJoueurs**. Suivant le type de tournoi (morpion, dame, etc.) que l'on souhaite réaliser, on initialise le tournoi avec l'objet Builder approprié (BuilderPartieMorpion, BuilderParieDame, etc.) depuis son constructeur.



Remarque : Ce motif de modélisation (design pattern) se nomme «Factory». Il est utilisé à chaque fois qu'un objet a besoin pour son fonctionnement de créer un autre objet dont il ne connaît pas le type précis à priori (par exemple, ici la classe Tournoi pourra fonctionner avec un Builder de type BuilderJeuScrable créé ultérieurement à la classe Tournoi). Ce motif de modélisation met encore une fois en pratique le polymorphisme des langages objet.

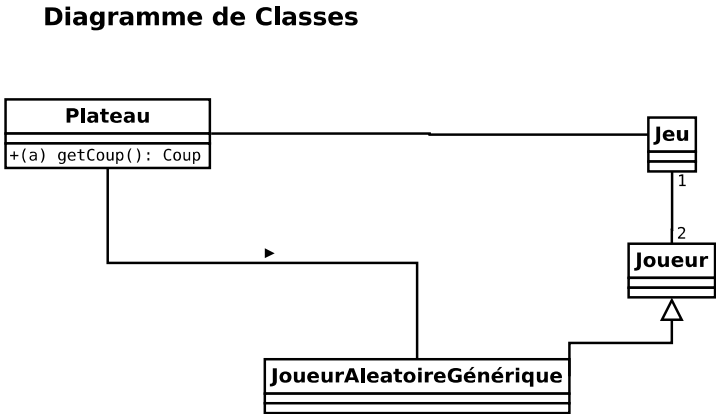
Pour cet exercice :

- Proposer une implémentation pour la classe tournois (selon les règles, assez simples, que vous définissez).
- Proposer un Builder de *JeuDeMorpion*, et un Builder de *JeuDePuissance4*.

Exercice 3 : Proposition d'une IA générique

PartieA : Proposez un joueur générique qui exploite une nouvelle fonctionnalité de la classe plateau : +(a) getCoup() : Coup

Vous remarquez que ce nouveau joueur est complètement générique.



Partie B : Développer la classe JoeurIA_MonteCarlo qui fonctionne de la façon suivante :

- Creer X copies de l'état de jeu courant (utiliser un constructeur par copie, on une fonction de clonage en vous documentant sur l'API de la fonction clone() de la classe Object). Ces copies utilisent chacune deux joueurs de JoueurAleatoireGénérique
- Jouer toutes les parties jusqu'à la fin (ne pas développer de nouveau, réutiliser l'existant)
- Le JoeurIA propose le coup N+1 ayant permis le plus de victoire (pour le joeur ayant joué le coup N+1). (en cas d'égalité, choix aléatoire parmi les meilleurs coups)

Partie C : Proposez la parallélisation suivante : un processus par partie à simuler.

Un processus a pour fonction de créer tous les autres, pour s'endormir ensuite. Le dernier processus à terminer l'exécution d'une partie réveille le processus initiateur afin qu'il poursuive son traitement.

Exercice 4 : Proposez la partie Vue et Contrôleur de MVC pour le Jeu du Morpion en utilisant la librairie graphique de votre choix.

- commencer avec uniquement des joueurs aléatoires
- ajouter ensuite le traitement des événements de l'utilisateur récupérés sur le damier