ARCHITECTURE

Auteur : Kim SAVAROCHE
Date : 31/03/2015
Sujet : Documentation technique du TP01 dans la matière « Architecture logicielle »

Ce document a pour objectif d'expliciter le raisonnement suivi pour répondre au sujet du TP. Ce dernier avait pour objectif d'implémenter un jeu de Morpion, appelé aussi Tic Tac Toe, et un jeu de Dames.

Les deux jeux comportent des similarités. En effet, on doit gérer la position des pions sur une grille. Ces pions sont posés sur une grille par un joueur. L'idéal est alors de réussir à créer des classes assez génériques pour être ensuite réutilisées dans n'importe quel jeu comportant une grille. La bataille navale ou les petits chevaux pourraient utiliser l'architecture proposée par exemple.

Il est imposé de déléguer la responsabilité entre les classes. On évite ainsi qu'une classe principale ait accès à tous les constructeurs, les destructeurs et les méthodes de toutes les classes. Ainsi la structure est plus découpée et l'architecture est plus équilibrée.

Vous trouverez donc ci-dessous une justification des choix ainsi qu'une explication de l'architecture proposée. Une attention particulière a été portée pour structurer le programme. Vous rencontrerez donc quelques termes liés au Design Pattern.

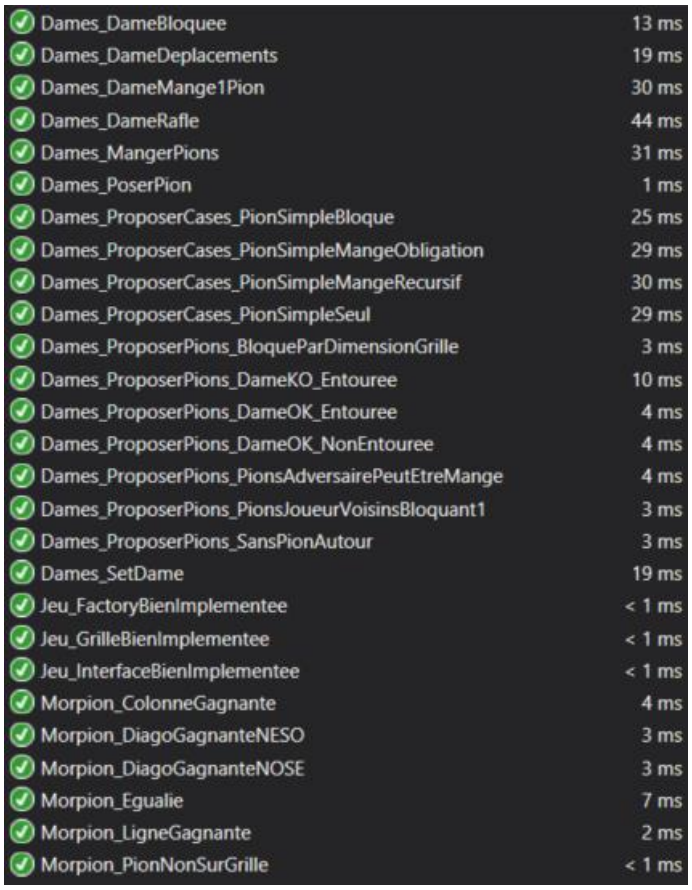
I. DEVELOPPEMENT

Aucun langage n'était imposé. Le C++ a été choisi afin de découvrir ce langage. En effet, n'ayant aucune expérience encore avec ce dernier, j'étais curieuse d'être confrontée à ses particularités. De plus, il est fortement recommandé pour programmer des jeux. L'occasion de s'initier au C++ s'est donc présentée.

L'objectif principal étant de créer des classes les plus génériques possibles, l'interface graphique est uniquement un affichage en console. Ceci a été convenu dès le départ en classe.

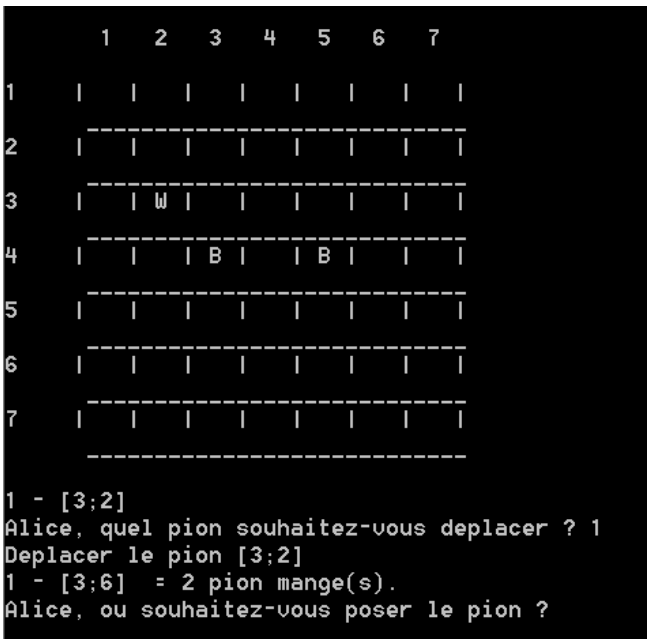
Les deux jeux incluent la même bibliothèque **Jeu_lib**. Afin de s'assurer qu'aucune régression n'apparait à la suite d'une modification d'une classe, le développement s'est effectué en TDD. Vous trouverez donc un total de 27 tests unitaires.

Liste des tests unitaires

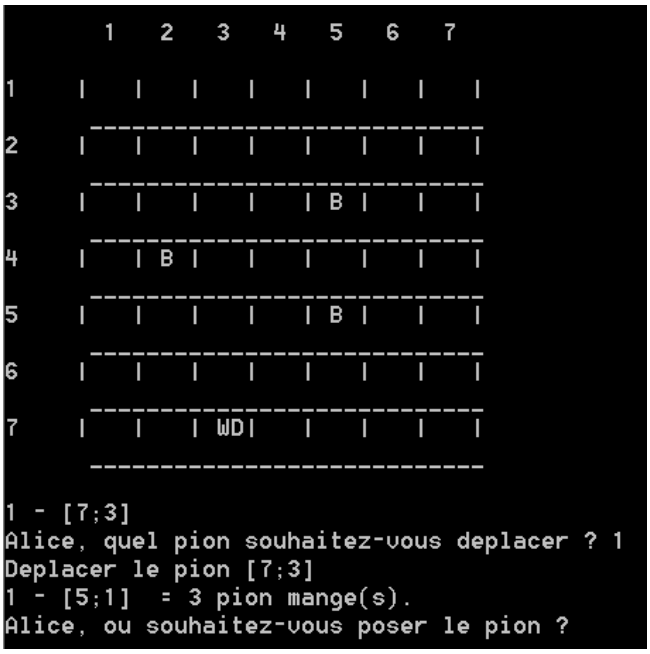


On peut remarquer que les deux tiers sont associés au jeu de Dames. Ceci peut être expliqué par le simple fait que les règles des Dames sont plus nombreuses que pour le Morpion. Ainsi l'obligation de manger le pion, la possibilité d'effectuer une rafle avec une dame et le choix de quel chemin emprunter si plusieurs sont possibles sont des situations testées unitairement.

Manger les pions récursivement



Gestion d'une rafle avec la dame



II. ARCHITECTURE GLOBALE

Les classes les plus génériques sont dans la bibliothèque **Jeu_lib.** La structure présentée ci-dessous est reprise dans **Morpion_lib** et **Dames_lib**.

La classe **Jeu** est la classe comportant les propriétés d'un jeu. Le Jeu indique le nombre de joueurs à créer, combien de lignes et de colonnes la Grille doit comporter et transfert les informations entre le Joueur et le DriverGrille. Elle permet de demander l'avis aux utilisateurs. Cette dernière fait la jonction entre la classe Joueur et DriverGrille. De cette façon, l'aspect plateau de jeu est séparé de l'aspect joueur humain, ce qui est assez proche de la réalité.

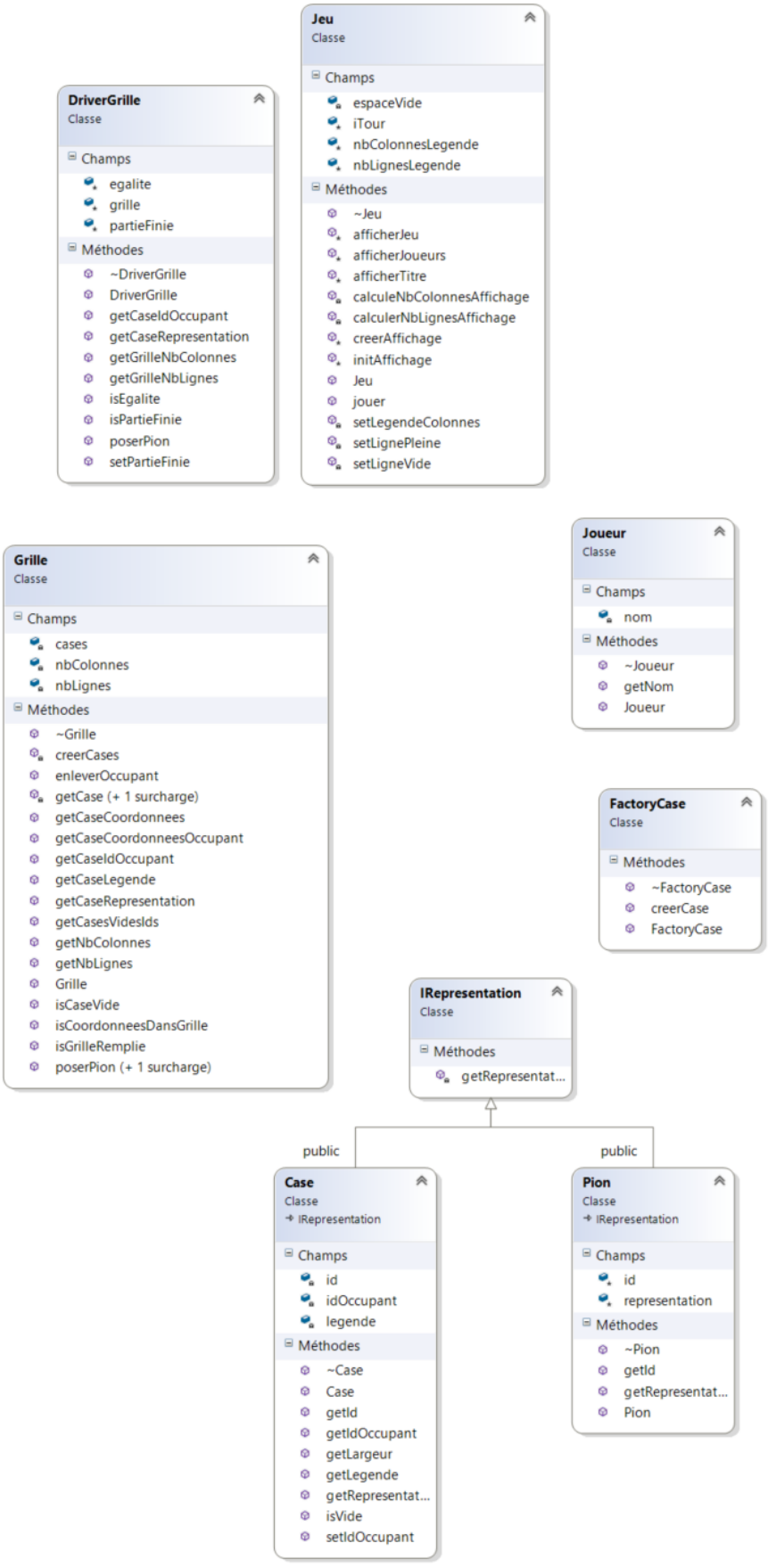
Chaque **Joueur** utilise une **FactoryPion** pour créer le nombre de pions indiqués par les règles du jeu. Un **Pion** est une entité avec une représentation. Ainsi seul le joueur est conscient de quel pion il possède et c'est le seul à avoir accès aux méthodes d'un pion directement.

De la même façon, un **DriverGrille** est une classe intermédiaire implémentant les règles liées à la grille de jeu. Il aide le Jeu à vérifier si un pion peut être placé à tel endroit, si la partie est finie et si un joueur a gagné ou non. Cette classe a donc à disposition la **Grille** qui est composée de cases. Seule la grille a la notion des coordonnées d'une case.

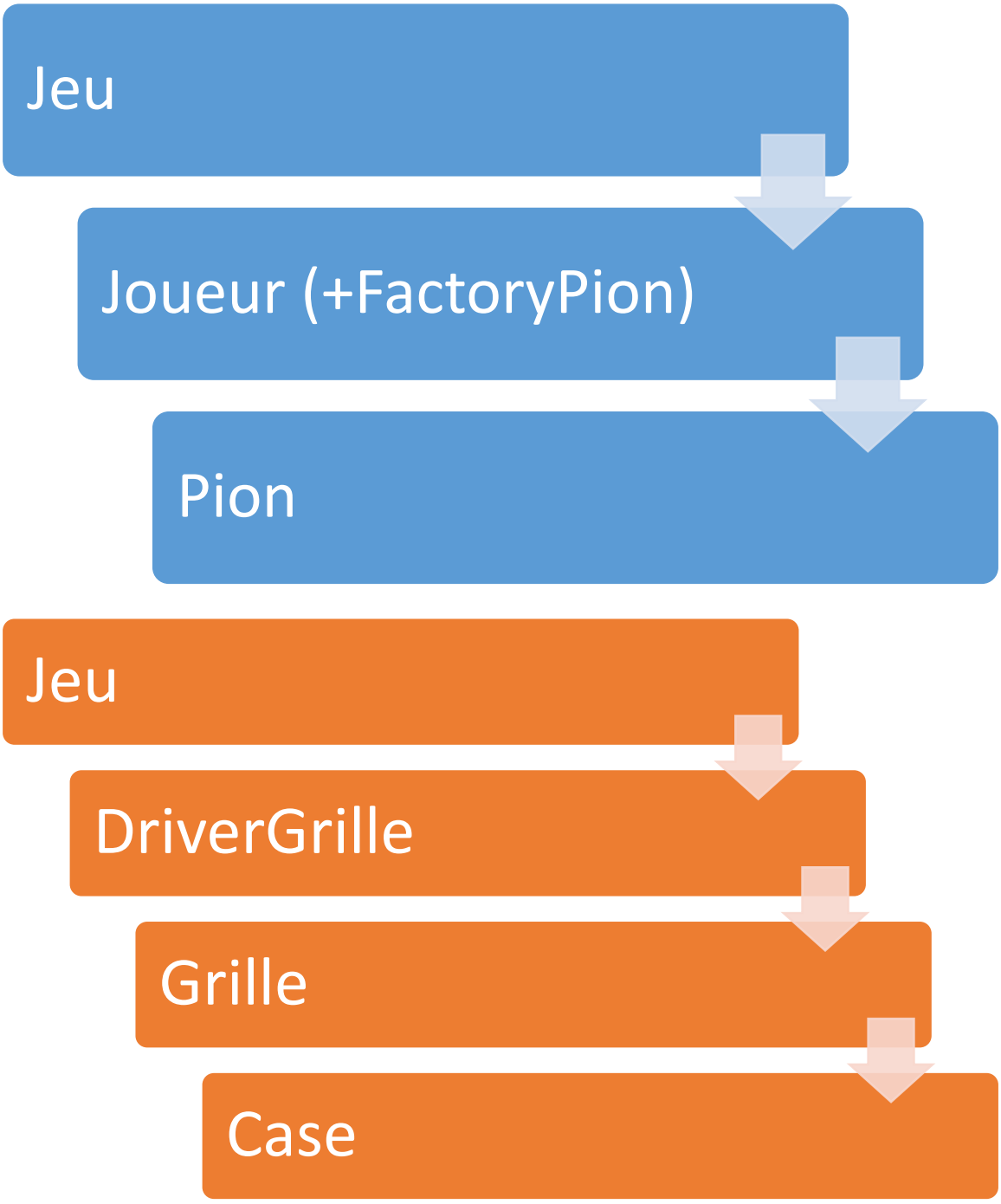
Une **Case** peut accueillir une entité quelconque. Elle possède également une représentation. Le nombre d'entité a été fixé à 1 de par les deux jeux imposés, cependant si on transforme le long en tableau de long alors la Case pourra comporter plusieurs occupants. La classe est donc facilement adaptable pour le Jeu de l'oie ou le Monopoli où plusieurs pions peuvent être positionnés sur la même case.

Vous pouvez remarquer l'objet **IRepresentation** qui est une Interface. Cette dernière est rattachée à tout objet qui peut être affiché en console. De cette manière, si on souhaite implémentée une interface 2D ou 3D, seule la fonction abstraite getRepresentation() serait à modifier.

Diagramme de classe de l'architecture globale



Toutes les classes énoncées ci-dessus sont reprises sous forme d'héritage dans les architectures du Morpion et des Dames. Afin de mettre en avant les objectifs du TP, un schéama récapitulatif des responsabilités de chaque classe vous est proposé :

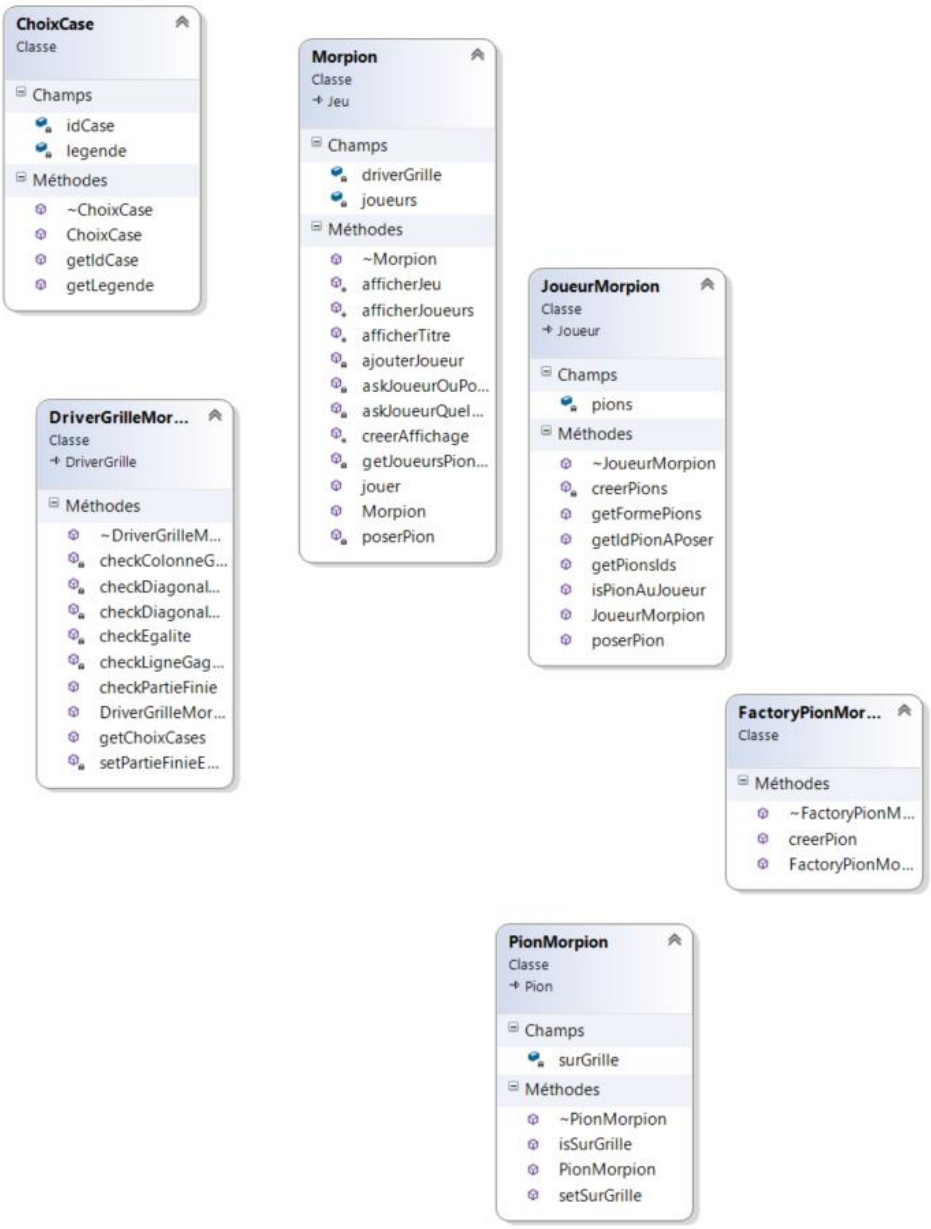


III. ARCHITECTURE HERITEE

Pour les deux jeux demandés, leur architecture reprend la globale décrite au-dessus. Seules les spécificités de chacun sont ajoutées aux classes filles.

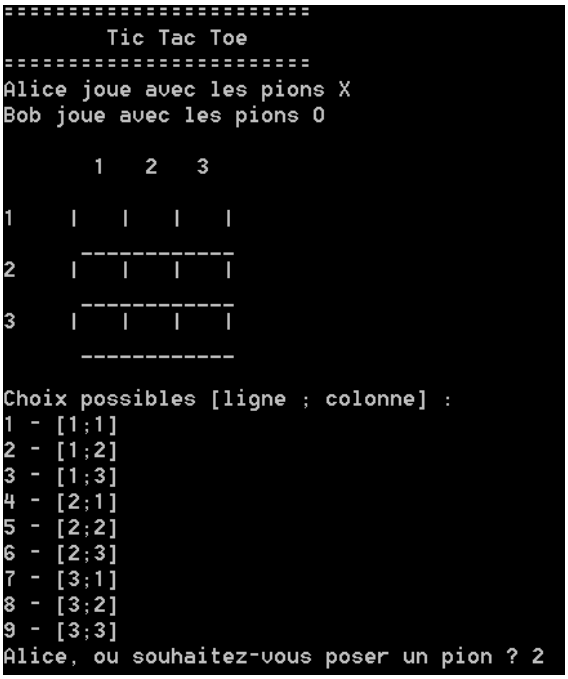
Dans un premier temps, le Morpion comporte une grille de 3 lignes et 3 colonnes. Chaque joueur dispose de 5 pions. A chaque tour, le Joueur donne l'id d'un pion non posé. Le Driver propose les cases disponibles. Le Jeu demande à l'utilisateur où il veut poser le pion. L'information est transmise au Driver qui pose le pion et vérifie si une combinaison gagnante a été réalisée. Le jeu indique au Joueur que son pion est bien posé et le Joueur transmet ce statut au Pion.

Diagramme de classe de l'architecture du Morpion



L'algorithme part de la case impactée par le choix pour ensuite observer les cases autour. La reconnaissance d'une fin de partie ne se fait donc pas en parcourant toute la grille à chaque tour.

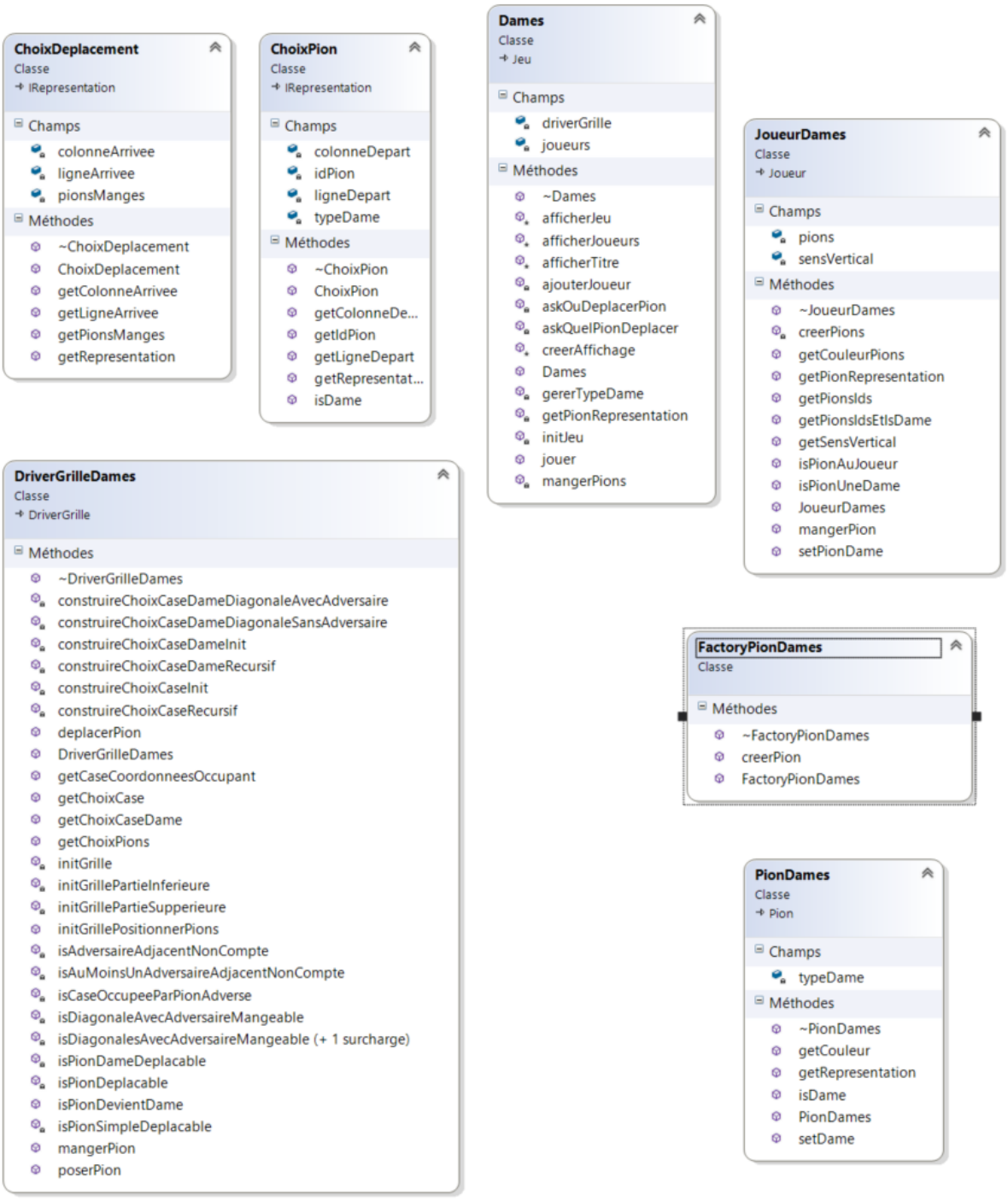
Le Morpion en console



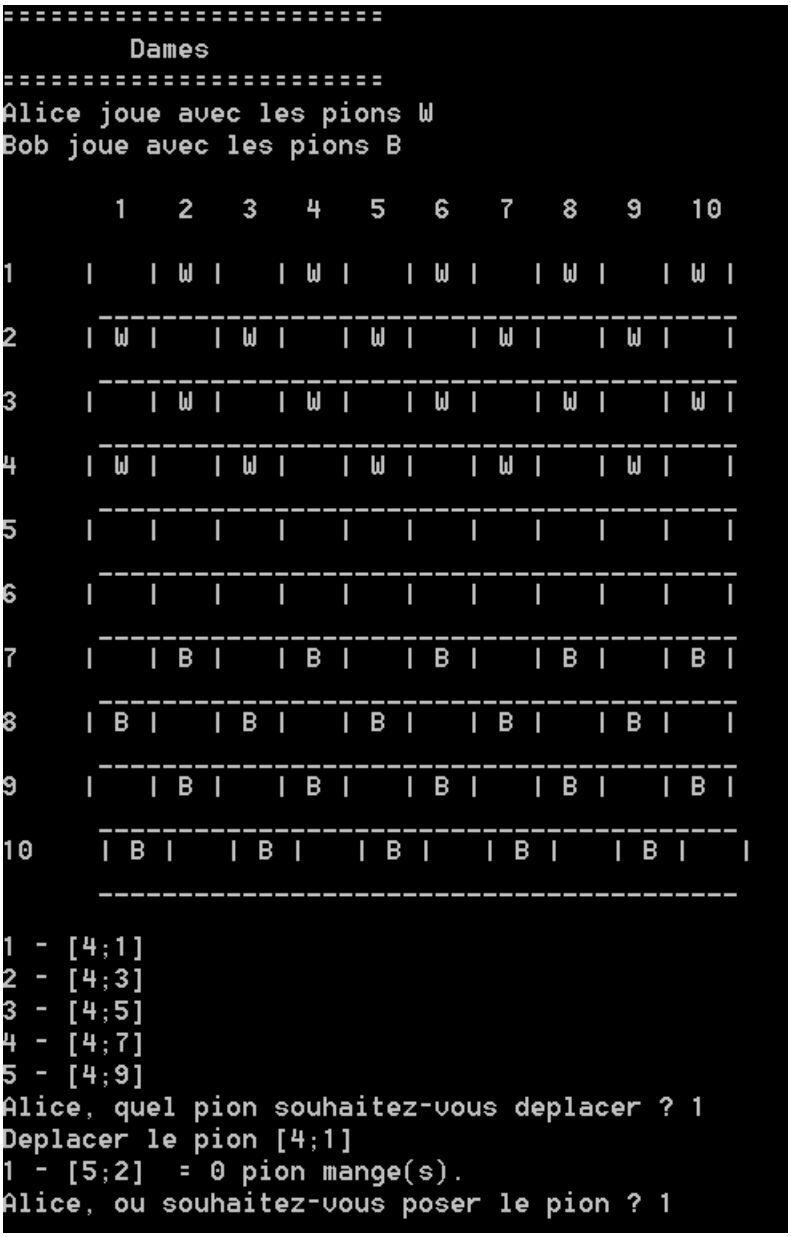
Quant aux dames, le jeu crée 2 joueurs et une grille de 10 lignes sur 10 colonnes. Pour chaque tous, on propose d'abord les pions déplaçables. Une fois le pion choisi, les cases possibles pour ce pion sont énumérées. Les choix tiennent compte de l'obligation de manger les pions adverses.

L'architecture reste la même comme convenu. On a besoin d'une classe choix supplémentaire pour réaliser une distinction plus nette entre le choix du pion du Joueur et les choix des cases par le DriverGrille.

Diagramme de classe de l'architecture des Dames



Les Dames en console



Chaque classe de Jeu_lib est assez générique pour être reprise dans n'importe quel type de jeu. Les factories permettent l'implémentation d'entités similaires plus facilement. L'interface permet d'implémenter aisément des changements liés à la modélisation visuelle des éléments. Enfin, la responsabilité de chaque classe est répartie afin de minimaliser les risques de bugs. Une classe entière n'est donc jamais retournée dans un get(), on passera toujours par l'intermédiaire du responsable.