Dans le même esprit, on aurait pu préciser que l'arbre d'héritage de *Pen* n'est pas complet (il y a certainement d'autres sortes de *Pen* que les stylos et les feutres) en lui adjoignant la contrainte prédéfinie {incomplete}.

EXERCICE 4-4.
La modélisation du domaine en pratique (suite)

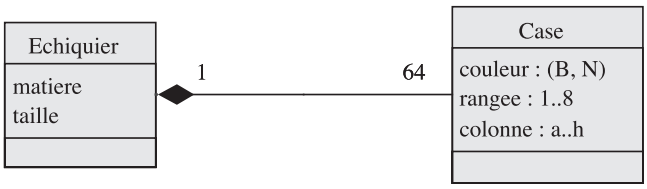
Proposez un cadre de modélisation des règles du jeu d'échecs.

Attachez-vous d'abord au matériel utilisé par les joueurs, puis à la notion de partie.

Solution

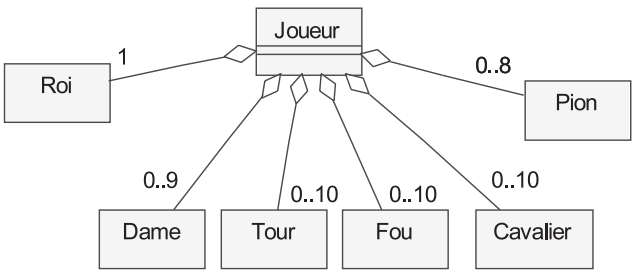
Commençons par interviewer un « expert métier » : le jeu d'échecs se joue à deux joueurs sur un échiquier carré composé de 64 cases, alternativement noires et blanches.

Figure 4-23.
Modélisation d'un échiquier et de ses cases



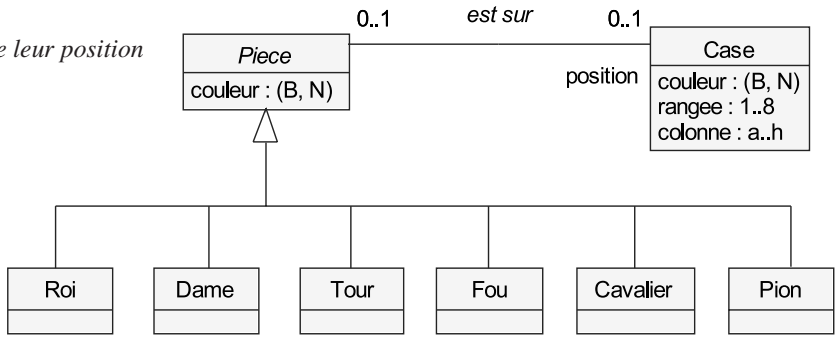
Chaque joueur possède initialement huit pions, ainsi qu'un roi, une dame, deux tours, deux fous et deux cavaliers. Par le biais de la promotion des pions en huitième rangée (transformation obligatoire en pièce à choisir sauf un roi), un joueur peut ainsi posséder jusqu'à neuf dames, dix tours, cavaliers ou fous et les multiplicités instantanées ne sont pas si évidentes !

Figure 4-24.
Modélisation des pièces de chaque joueur



Il ne peut y avoir qu'une pièce au maximum sur une case donnée. Les positions initiales des pièces sont représentées sur la figure précédente.

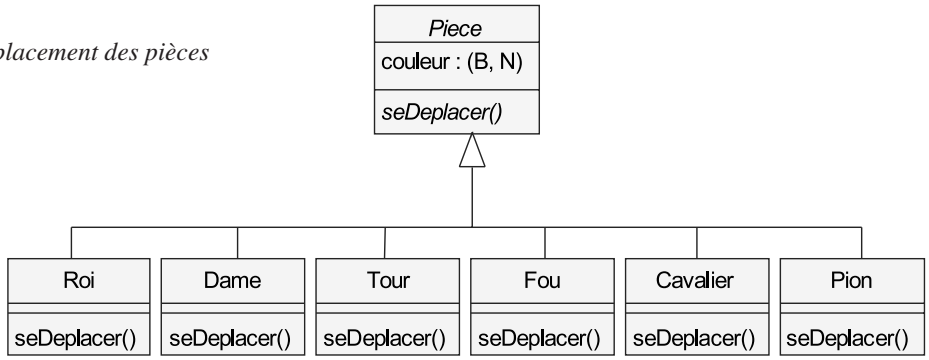
Figure 4-25.
Modélisation des pièces et de leur position



L'introduction de la classe abstraite *Piece* est tout à fait naturelle. Tout d'abord, il s'agit bien d'un mot faisant partie du vocabulaire du domaine. Ensuite, cela permet d'exprimer le concept de position par un rôle unique de l'association entre les classes *Piece* et Case. Sur une case donnée, il ne peut pas y avoir plus d'une pièce à la fois. Et une pièce est soit sur une case, soit hors du jeu (prise).

Les pièces ont chacune leur mode de déplacement propre.

Figure 4-26.
Modélisation du déplacement des pièces



Le déplacement des pièces est typiquement polymorphe. Chaque instance se déplace en fonction de l'algorithme déclaré au niveau des sous-classes concrètes.

En complétant l'interview de notre expert métier, nous ajoutons quelques attributs et opérations privées pour peaufiner la première partie du modèle (voir figure 4-27).

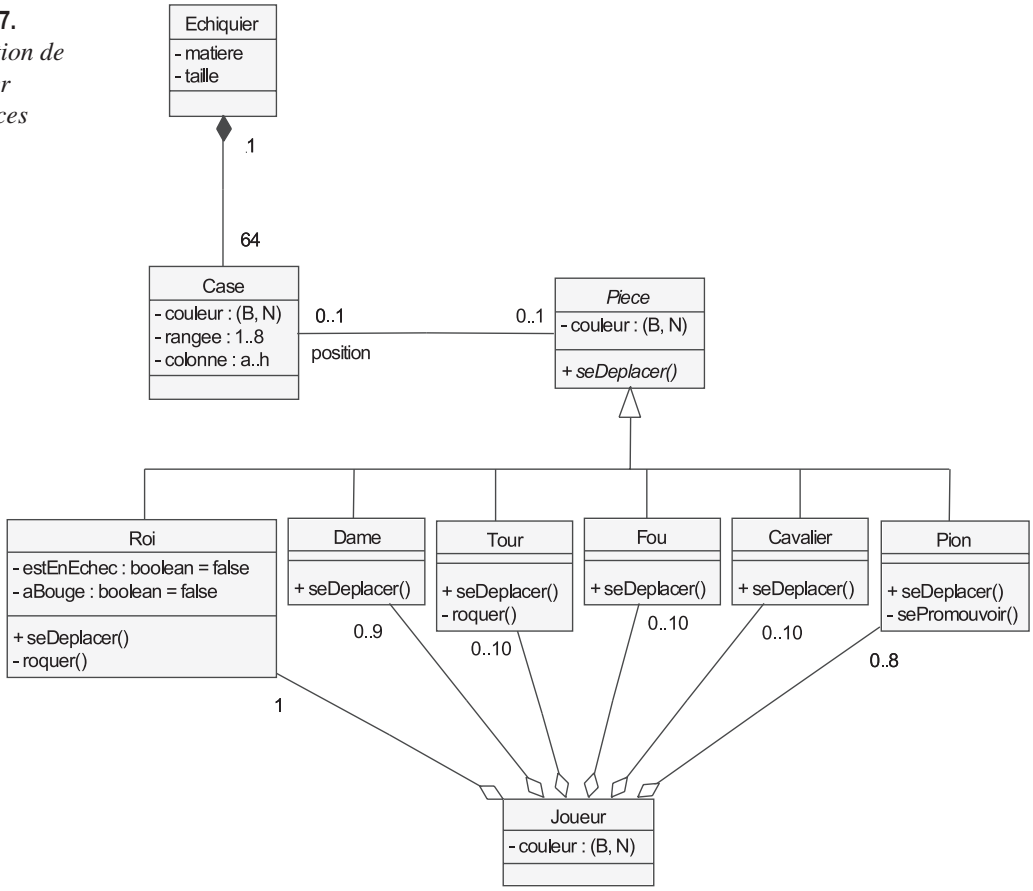


Figure 4-27.
Modélisation de l'échiquier et des pièces

À chaque tour, un joueur bouge une pièce de son camp (blanc ou noir). On ne peut ni passer son tour, ni jouer deux fois de suite. C'est toujours les blancs qui commencent la partie. Une partie est donc une suite ordonnée de coups.

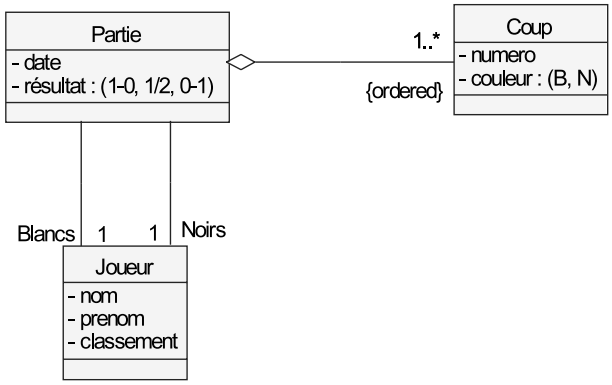


Figure 4-28.
Modélisation du déroulement de la partie

Si nous relions les concepts de coup, de pièce et de case, nous obtenons la figure 4-29.

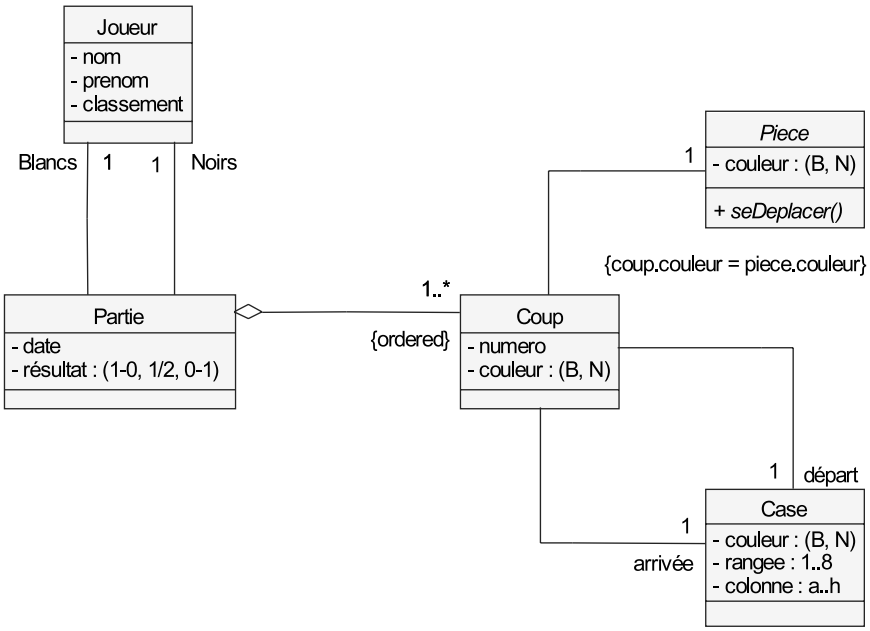


Figure 4-29.
Modélisation détaillée du déroulement de la partie

Nous arrêtons là cette exploration du jeu d'échecs par le biais de la modélisation statique, que nous pourrions encore détailler.

Par ailleurs, si nous souhaitons ajouter d'autres règles plus dynamiques concernant par exemple la fin de la partie (cas de mat, pat, abandon, partie nulle, etc.), un diagramme d'états est tout à fait approprié. Nous le présenterons au chapitre 6 (exercice 6-1).

LES CLASSES STRUCTURÉES UML 2

EXERCICE 4-5.
Limitations de la relation de composition

La figure 4-30[3] explique que les voitures et les bateaux possèdent un moteur, que les voitures ont des roues alors que les bateaux ont des hélices, et que les moteurs se décomposent tous de façon similaire. La composition exprime également le fait que la même instance de *Moteur* ne peut pas appartenir simultanément à une instance de *Voiture* et une instance de *Bateau*, même si la classe *Moteur* est partagée entre les classes *Voiture* et *Bateau*. C'est pour

3. Cet exercice est inspiré de l'excellent article de Conrad Bock intitulé « UML 2 Composition Model » et publié dans le « Journal of Object Technology », Vol. 3, N° 10, November-December 2004. Il est disponible sur le site web suivant : http://www.jot.fm/issues/issue_2004_11/column5.