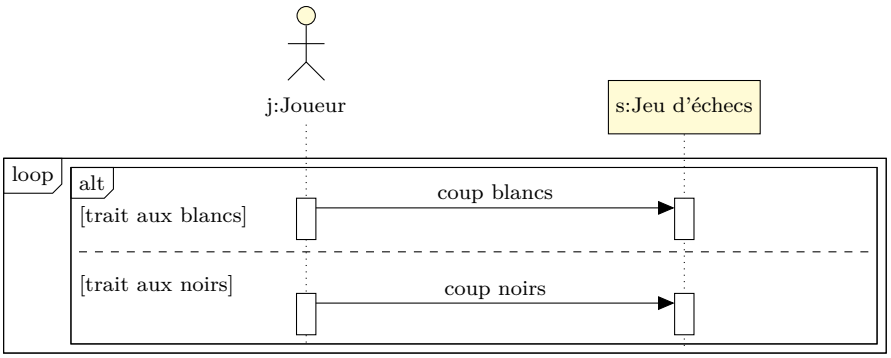
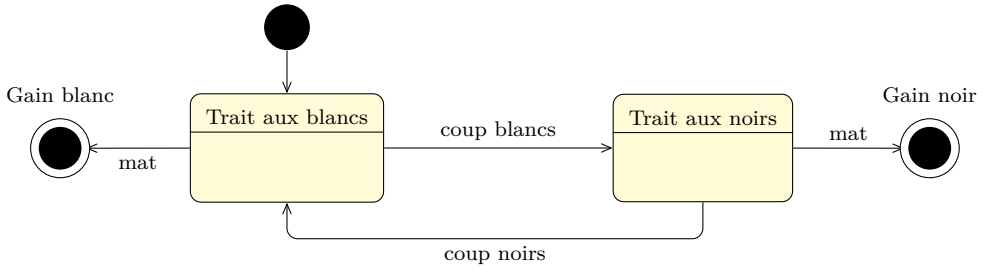
3.3 Mini-projet : jeu d'échecs

L'objectif de cette étude est de développer un jeu d'échecs. Le but du jeu est d'opposer deux joueurs sur un échiquier composé de 64 cases blanches et noires. Les joueurs jouent à tour de rôle en déplaçant l'une de leurs 16 pièces. Le but du jeu est d'infliger à son adversaire un échec et mat, une situation dans laquelle le roi d'un joueur est en prise sans possibilité de recours.

Le scénario d'utilisation typique du jeu d'échecs peut être décrit par un diagramme de séquence « au niveau système » qui contient une grande boucle, dans laquelle il arrive soit un coup des blancs soit un coup des noirs.



Les coups sont joués à tour de rôle par les blancs et les noirs. Lorsque le roi d'un joueur est en situation de mat, la partie se termine sur la victoire du joueur qui mate. Cela peut être représenté par un diagramme d'état.



INSTRUCTIONS Pour réaliser le jeu d'échecs, nous vous fournissons une solution Visual Studio organisée en deux dossiers. Le dossier « IHM » contient une simple interface homme-machine, alors que le dossier « Echecs » contient une implémentation partielle qui permet d'avoir une version exécutable du jeu (lancez-le pour tester). L'objectif du TP est de modéliser les règles du jeu d'échecs, afin d'identifier des classes qui serons ensuite utilisée pour implémenter le jeu (dans le dossier « Echecs »).

3.3.1 Modélisation du domaine

Modélisez les règles du jeu d'échecs en suivant les instructions ci-dessous. Cela vous permettra d'identifier un ensemble initial de classes que vous penserez à implémenter dans le dossier « Echecs » du projet fourni.

Échiquier

Le jeu d'échecs se joue à deux joueurs sur un **échiquier** carré composé de 64 **cases**, alternativement noires et blanches. Chaque case est caractérisée par la *couleur*, et son emplacement sur l'échiquier (*rangée* et *colonne*). Modélisez cette situation avec les relations appropriées, en sachant que les substantifs en gras et en italique sont respectivement des classes et des attributs.

Pièces

Chaque **joueur** a une *couleur* et 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. Modélisez cette situation avec les relations appropriées, en sachant que les substantifs en gras sont des classes.

Position des pièces

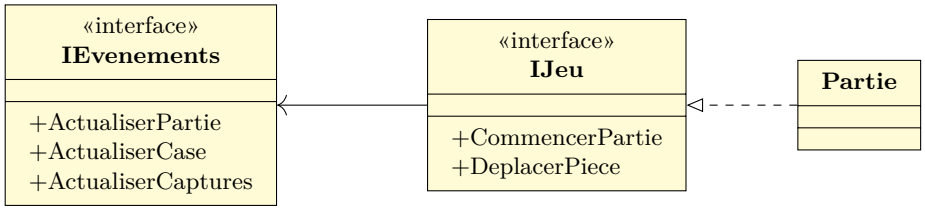
Une case ne peut accueillir qu'une seule **pièce** à la fois, laquelle est soit sur une case soit hors du jeu. Le mot « pièce » exprime que le pion, roi, dame, tour, fou et cavalier ont des choses en communs : ils ont une *couleur* et sont placés sur une seule case au maximum. On se demande si *pièce* est un concept plus général que pion, roi, etc. Respectent-ils la règle « est-un » ? Les sous-classes ont-elles toutes le même attribut (couleur) et la même association (case) qui peut être extraite et factorisée dans la super-classe ? Modifier le modèle précédent en fonction des réponses.

Partie

À chaque tour, un joueur déplace une pièce de son camp (blanc ou noir). On ne peut ni passer son tour, ni jouer deux fois de suite, et c'est toujours les blancs qui jouent en premiers. Une **partie** est donc une suite ordonnée de **coups**. Chaque coup comporte une pièce qui bouge de la case de départ à celle d'arrivée. Modifier le modèle précédent en reliant les concepts de joueur et partie (utilisez deux associations pour distinguer les blancs et les noires). Ensuite, reliez les concepts de partie, coup et pièce, puis les concepts de coup et case (utilisez deux associations pour les cases de départ et arrivée).

3.3.2 Conception de la couche présentation

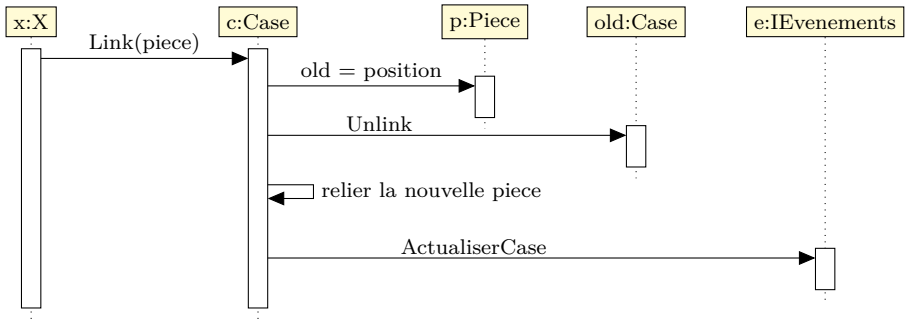
L'interaction entre les joueurs et le jeu est véhiculée par l'interface utilisateur, qui est responsable de visualiser l'état courant du jeu et d'interpréter les actions exécutées par les joueurs. Plus précisément, les couches présentation et domaine communiquent par le biais des interfaces `IJeu` et `IEvenements`. La première déclare les opérations que le joueur peut invoquer durant une partie, telles que `CommencerPartie()` et `DeplacerPiece()`. Elle est implémentée par la classe `Partie`. La deuxième spécifie les opérations pour mettre à jour l'interface utilisateur au fur et à mesure que le jeu avance.



Visualiser le contenu d'une case

La classe `Case` possède une association à la classe `Piece` pour indiquer qu'une case peut accueillir (au plus) une pièce. Pour relier correctement une case à une pièce, la classe `Case` fournit les méthodes `Unlink` et `Link`. La méthode `Unlink` annule la référence sur l'objet `Piece` et soulève un événement `ActualiserCase` pour effacer la case correspondante dans l'IHM.

En revanche, la méthode `Link` commence par invoquer `Unlink` sur la case où la pièce d'entrée se trouve, relie celle-ci avec la case courante (notamment l'objet « this »), et enfin soulève un événement `ActualiserCase` pour mettre à jour le contenu de la case correspondante dans l'interface utilisateur.



instructions Implémentez les méthodes `Unlink` et `Link` dans la classe `Case`, puis complétez la méthode `CommencerPartie` en suivant les indications en commentaire. Testez en supprimant le code entre TEST et FIN TEST.

3.3.3 Conception des opérations métier

Continuez l'implémentation des classes du domaine à travers l'ajout des opérations qui sont spécifiées dans les sections suivantes.

Création d'une nouvelle partie

La méthode `CommencerPartie` permet de démarrer une nouvelle partie. En particulier, elle crée les deux joueurs et l'échiquier, elle demande aux joueurs de positionner leurs pièces sur l'échiquier, et enfin elle émet l'événement `ActualiserPartie` pour donner le trait aux blancs.

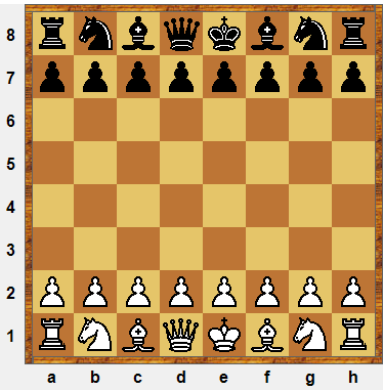


Figure 3.1 *Configuration initiale de l'échiquier.*

instruction Complétez la méthode `PlacerPieces` dans la classe `Joueur`, afin de positionner les pièces sur l'échiquier comme montré dans la figure ci-dessus. Ensuite, décommentez le contenu de la méthode `DeplacerPiece` et supprimez le code entre TEST et FIN TEST. Vérifier que vous pouvez bouger les pièces sur l'échiquier en alternance entre les blancs et les noirs.

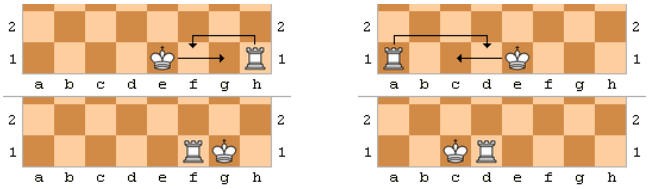
Déplacement des pièces

La méthode `DeplacerPiece` s'occupe de bouger une pièce de la case de départ à celle d'arrivée. Cependant, pas tous les déplacements sont valides, car les pièces ont chacune leur mode de déplacement propre. Votre tache est de modifier la méthode `DeplacerPiece`, ainsi que d'implémenter toute autre méthode que vous jugez nécessaires, afin de gérer le déplacement des pièces conformément aux règles des échecs. *Conseil :* L'opération de déplacement est polymorphe, c'est-à-dire chaque instance de la classe `Piece` doit se déplacer en fonction de l'algorithme implémenté au niveau des sous-classes concrètes.

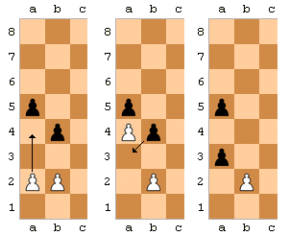
Autres opérations

Vous êtes encouragés à implémenter un maximum d'autres opérations. Elles peuvent concerner le domaine du jeu d'échecs, telles que :

1. le roque, consistant à déplacer en un seul coup le roi et l'une des tours ;



2. la prise en passant, qui s'applique aux pions déplacés de deux cases ;



3. la détection de l'échec et l'interdiction de déplacer les pièces de telle sorte que le roi est exposé à l'échec ;
4. la promotion des pions qui atteignent la dernière rangée de l'échiquier (la huitième pour les blancs et la première pour les noirs) ;
5. la déclaration d'une partie nulle, qui désigne une position dans laquelle le camp ayant le trait et n'étant pas sous le coup d'un échec, ne peut plus jouer de coup légal sans mettre son propre roi en échec.

Également, elles peuvent concerner la conception d'une couche applicative qui supporte des fonctionnalités avancées de l'interface utilisateur, telles que :

1. l'affichage des pièces perdues ;
2. la possibilité de recommencer une nouvelle partie ;
3. l'affichage du score ;
4. la gestion du « play & pause » ;
5. l'historique des mouvements précédents ;
6. l'affichage des prédictions lors d'un déplacement d'une pièce ;
7. la gestion de `Undo()` et `Redo()`.