Modélisation

❖ Classes Identifiées :

1. **Game**
2. **Board**
3. **Square**
4. **Piece**
5. **Move**
6. **Player**
7. **King**
8. **Queen**
9. **Rook**
10. **Bishop**
11. **Knight**
12. **Pawn**
13. **PromotedPawn**

❖ Attributs de Chaque Classe :

- **Game**
  - *id* : int
  - *playerList* : Array
  - *moveList* : List
  - *state* : GameState
  - *selectedIndex* : int
- **Board**
  - *squares* : Square[8][8]
  - *pieceSets* : *pieceSets*

- **Square**
  - *x* : int
  - *y* : int
  - *piece* : Piece
- **Piece**
  - *color* : PieceColor
  - *hasMoved* : bool
- **Move**
  - *toSquare* : Square
  - *fromSquare* : Square
  - *pieceMoved* : Piece
  - *pieceCaptured* : Piece
- **Player**
  - *name* : string
  - *isActive* : bool
- **King (hérite de Piece)**
  - Pas d'attributs supplémentaires
- **Queen (hérite de Piece)**
  - Pas d'attributs supplémentaires
- **Rook (hérite de Piece)**
  - *hasCastled* : bool
- **Bishop (hérite de Piece)**
  - Pas d'attributs supplémentaires
- **Knight (hérite de Piece)**
  - Pas d'attributs supplémentaires
- **Pawn (hérite de Piece)**
  - *doubleStepMove* : bool

- **PromotedPawn (hérite de Piece)**
  - *promotionMoveNo* : int

❖ Opérations de Chaque Classe :

- **Game**
  - *initialize()*
  - *updateState()*
  - *render()*
- **Board**
  - *reset()*
  - *movePiece(move:Move)*
  - *validateMove(move:Move) : bool*
- **Square**
  - *getPiece() : Piece*
- **Piece**
  - *getAvailableMoves(board:Board) : List*
  - *move(destination:Square)*
  - *capture()*
- **Move**
  - *execute()*
  - *undo()*
- **Player**
  - *makeMove()*
- **King**
  - *check() : bool*
  - *checkmate() : bool*
- **Queen, Rook, Bishop, Knight, Pawn, PromotedPawn**
  - *getAvailableMoves(board:Board) : List*

Traduction des termes spécifiques visible dans le diagramme, en attributs ou logique programmable pour chaque classe et objet

1. Classe Game

- **Attributs :**
  - playedDate: Date - Date à laquelle la partie a été jouée.
  - status: GameStatus - Statut de la partie (par exemple, Active, Complete, Pending).
  - result: String - Résultat de la partie (par exemple, White wins, Black wins, Draw).
- **Associations :**
  - Gérer la relation avec Player pour les joueurs de blanc et de noir.
  - Un lien vers Board pour représenter l'état actuel du jeu.

2. Classe Player

- **Attributs :**
  - username: String - Nom d'utilisateur du joueur.
  - rating: Int - Classement du joueur.
- **Opérations :**
  - updateRating(newRating: Int) - Mettre à jour le classement du joueur.

3. Classe Board

- **Attributs :**
  - squares: Square[][] - Un tableau à deux dimensions qui représente les cases de l'échiquier (8x8).

4. Classe Piece

- **Attributs :**
  - color: Color - Couleur de la pièce (Blanc ou Noir).
  - position: Square - Position actuelle de la pièce sur l'échiquier.
- **Opérations :**
  - move(destination: Square) - Déplace la pièce à la destination spécifiée.

Classes et leurs Attributs

- **ChessPiece (Pièce)**
- **Attributs**
  - color: Enum(White, Black) - détermine la couleur d'une pièce.
  - type: Enum(King, Queen, Rook, Bishop, Knight, Pawn) - spécifie le type de la pièce.
  - isMoved: Boolean - indique si la pièce a déjà été déplacée.
  - isCaptured: Boolean - indique si la pièce est capturée.

❖ Square (Case)

- **Attributs**
  - piece: ChessPiece (facultatif) - pièce présente sur la case.
  - coordinate: Coordinate - coordonnées de la case sur l'échiquier.

❖ Board (Échiquier)

- **Attributs**
  - squares: Array[8][8] de Square - représente les cases de l'échiquier.

❖ Player (Joueur)

- **Attributs**
  - pieces: List - les pièces que le joueur possède.
  - isActive: Boolean - indique si c'est le tour du joueur.
  - isInCheck: Boolean - si le roi du joueur est en échec.
  - isCheckmated: Boolean - si le roi du joueur est en échec et mat.

❖ Game (Partie)

- **Attributs**
  - players: Array[2] de Player - les joueurs de la partie.
  - board: Board - l'échiquier utilisé pour la partie.
  - status: Enum(Active, Check, Checkmate, Stalemate) - état actuel de la partie.

- **Méthodes**
  - play(): Débuter ou continuer une partie.
  - join(): Permet à un joueur de rejoindre la partie.
  - operate(): Opérations de jeu générales, telles que le déplacement des pièces.

Détails Supplémentaires des Classes

❖ Coordinate (Coordonnées)

- **Attributs**
  - x: int - coordonnée X sur l'échiquier.
  - y: int - coordonnée Y sur l'échiquier.

❖ Move

- **Attributs**
  - fromSquare: Square - case de départ du mouvement.
  - toSquare: Square - case d'arrivée du mouvement.

- **Méthodes**
  - isValid(): valide si le mouvement est légal.

Sous-classes spécifiques aux pièces

Les différentes classes de type de pièces (King, Queen, etc.) héritent de la classe ChessPiece et peuvent avoir des méthodes spécifiques pour valider les mouvements selon les règles de déplacement de chaque pièce.

Logique Additionnelle

- **PlayerType** : Diversifie les types de joueurs (ComputerPlayer, HumanPlayer), ce qui permet d'intégrer une intelligence artificielle ou un contrôle utilisateur.
- **Engine**: Gère la logique de jeu, l'IA, et les interactions joueur-partie.