**PieceController**

1. PieceController Class:

Manages the behavior and movement of individual chess pieces.

Contains references to the GameController, WhitePieces, BlackPieces, and QueenSprite.

Defines variables for movement speed, highest/lowest ranks, and flags for double steps and movement direction.

Implements methods for piece movement, validation, capturing, and special moves like castling, promotion, and en passant.

Handles piece selection, deselection, and checks if a piece is in the process of moving.

1. OnMouseDown Method:

Triggered when a chess piece is clicked.

Handles piece selection, deselection, and triggering movements.

1. MovePiece Method:

Initiates the movement of a chess piece to a new position.

Handles special moves like double steps, promotions, and castling.

Validates movements and checks for potential collisions or captures.

1. ValidateMovement Method:

Checks if a particular movement from the current position to a new position is valid for the specific chess piece.

Considers various rules for different chess pieces, such as straight lines, diagonals, L-shapes for knights, etc.

1. GetPieceOnPosition Method:

Retrieves the chess piece present on a specific board position.

Allows filtering by color (White or Black) or retrieving any piece regardless of color.

1. CountPiecesBetweenPoints Method:

Counts the number of pieces between two specified points on the board along a specific direction (horizontal, vertical, or diagonal).

1. IsInCheck Method:

Determines if a chess piece is currently in check.

Temporarily moves the piece to a potential position and checks for threats from opponent pieces.

1. MoveSideBySide and MoveDiagonally Methods:

Handle the smooth interpolation of piece movement along the Y-axis and X-axis, respectively.

1. Promote Method:

Converts a pawn to a queen when it reaches the opposite end of the board.

1. IsMoving Method:

Checks if a piece is currently in the process of moving.

1. **Start Method:***void Start()* 
   * Initializes the **MoveSpeed** for the piece, and for knight pieces, it doubles their move speed.
   * Ensures that the **GameController** reference is assigned.
2. **Update Method:***void Update()* 
   * Checks whether the piece is currently moving (**MovingY** or **MovingX**) and invokes corresponding movement methods.
3. **OnMouseDown Method:***void OnMouseDown()* 
   * Handles mouse clicks on the piece.
   * If another piece is already selected and is in the process of moving, it prevents further clicks.
   * If the selected piece is the current piece, it deselects it. Otherwise, it either selects the piece or attempts to move the piece.
4. **MovePiece Method:***public bool MovePiece(Vector3 newPosition, bool castling = false)* 
   * Moves the piece to the specified position if the move is valid.
   * Handles special moves like double-step for pawns, promotion for pawns reaching the opposite end, and castling for kings.
5. **ValidateMovement Method:***public bool ValidateMovement(Vector3 oldPosition, Vector3 newPosition, out GameObject encounteredEnemy, bool excludeCheck = false)* 
   * Validates whether the movement from **oldPosition** to **newPosition** is valid.
   * Checks if the path is clear and if the move is allowed based on the piece's type (king, rook, bishop, etc.).
6. **GetPieceOnPosition Method:***GameObject GetPieceOnPosition(float positionX, float positionY, string color = null)* 
   * Returns the piece on a given position on the board or null if the square is empty.
   * Optionally, you can specify the color of the piece to retrieve (white or black).
7. **CountPiecesBetweenPoints Method:***int CountPiecesBetweenPoints(Vector3 pointA, Vector3 pointB, Direction direction)* 
   * Counts the number of pieces between two points along a specified direction (horizontal, vertical, or diagonal).
8. **IsInCheck Method:***public bool IsInCheck(Vector3 potentialPosition)* 
   * Checks if the king is in check after a potential move to the specified position.
   * Temporarily moves the piece to the potential position to check for threats.
9. **MoveSideBySide Method:***void MoveSideBySide()* 
   * Handles the side-by-side movement of the piece (either horizontal or vertical).
10. **MoveDiagonally Method:***void MoveDiagonally()* 
    * Handles the diagonal movement of the piece.
11. **Promote Method:***void Promote()* 
    * Converts a pawn to a queen when it reaches the opposite end of the board.
12. **IsMoving Method:***public bool IsMoving()* 
    * Returns **true** if the piece is currently moving, either horizontally or vertically.
13. **Direction Enum:***enum Direction* 
    * An enum representing the possible directions for counting pieces (**Horizontal**, **Vertical**, **Diagonal**).

**BOX controller**

The "BoxController" class in the provided code appears to be responsible for managing individual squares (boxes) on a chessboard. Here is a summary of its functionality:

1. Initialization:

It has a reference to the "GameController."

In the Start method, it assigns the "GameController" if it's not provided in the inspector.

It generates an algebraic name for the square based on its position and assigns it to the parent's name.

1. Input Handling:

The OnMouseDown method is triggered when a user clicks on the square.

It checks if there is a selected piece in the game, and the selected piece is not in the process of moving.

If conditions are met, it calls the MovePiece method on the selected piece, passing the position of the clicked square.

**GameController**

1. **Initialization:**
   * The class has references to the game objects representing the chessboard, white pieces, and black pieces.
   * It tracks the currently selected piece and whether it is currently the turn for the white player.
2. **Piece Selection:**
   * The **SelectPiece** method is used to select a chess piece.
   * If the selected piece belongs to the current player (either white or black), it is highlighted in yellow, and its position is adjusted to be above other pieces.
3. **Piece Deselection:**
   * The **DeselectPiece** method is used to deselect the currently selected piece.
   * It removes the highlight from the selected piece and resets its position to the same level as other pieces.
4. **Turn End:**
   * The **EndTurn** method is responsible for ending the current player's turn and switching to the other player.
   * It checks for valid moves and whether the opponent's king is in check.
   * If no valid moves are available, it checks for stalemate or checkmate conditions.
5. **Checking Valid Moves:**
   * The **HasValidMoves** method checks if a given chess piece has any valid moves on the current board.
   * It iterates through each square on the board and checks if the piece can move to that square based on its movement rules.
6. **End Game Conditions:**
   * If no valid moves are available for the current player:
     + If the king is not in check, it triggers a stalemate.
     + If the king is in check, it triggers a checkmate.
7. **Stalemate and Checkmate Handling:**
   * The **Stalemate** and **Checkmate** methods log messages indicating the respective game states.

***OVERVIEW***

1. **PieceController.cs:**
   * Represents the behavior and attributes of individual chess pieces (e.g., pawns, rooks, kings).
   * Handles the movement, validation of moves, and special actions like castling and promotion.
   * Maintains information about the piece's current state (e.g., whether it has moved).
   * Interacts with the game board, tracks positions of other pieces, and checks for the king's safety.
2. **BoxController.cs:**
   * Represents the behavior of the chessboard squares.
   * Initializes each square with an algebraic name based on its position on the board.
   * Detects mouse clicks on the squares and communicates with the GameController to handle piece movement.
3. **GameController.cs:**
   * Manages the overall game state and flow.
   * Keeps track of the chessboard, white and black pieces, and the currently selected piece.
   * Handles player turns and switching between white and black players.
   * Invokes methods in PieceController to move pieces and checks for game end conditions like checkmate and stalemate.

**Interactions:**

* **PieceController and GameController:**
  + PieceController communicates with the GameController to request moves, check the validity of moves, and report the piece's current state.
  + GameController selects and deselects pieces based on player input and manages the game state.
* **BoxController and GameController:**
  + BoxController communicates with the GameController to handle mouse clicks on the chessboard squares.
  + GameController uses BoxController to initialize the chessboard and detect user input for moving pieces.

**Overall Flow:**

1. **Initialization:**
   * GameController initializes the chessboard, white and black pieces, and other game-related elements.
   * BoxController sets up the chessboard squares with algebraic names.
2. **Gameplay:**
   * Players interact with the chessboard by clicking on pieces (handled by BoxController) and making moves.
   * GameController manages the overall flow, validates moves using PieceController, and updates the game state.
3. **Piece Movement:**
   * PieceController handles the movement logic for individual pieces, including special moves and validations.
   * GameController communicates with PieceController to request moves and checks for valid moves.
4. **Game State Checks:**
   * GameController checks for conditions like checkmate and stalemate using methods in PieceController.
   * PieceController communicates the state of individual pieces to GameController.