**Overall Workflow for findPieces.tsx**

This function is a key part of the chess detection system. It processes video frames using the AI model to detect chess pieces, updates the game state, and renders the results on the canvas. Below is a detailed breakdown of how findPieces works:

**1. Initialization**

The findPieces function initializes variables and sets up the processing loop.

• **Where Initialization Begins**:

• The initialization code starts at the top of the function:

let centers: number[][] | null = null;

let boundary: number[][];

let centers3D: tf.Tensor3D;

let boundary3D: tf.Tensor3D;

let state: number[][];

let keypoints: number[][];

let possibleMoves: Set<string>;

let requestId: number;

let greedyMoveToTime: { [move: string]: number };

• **Purpose**:

• These variables store information like detected chessboard centers, boundaries, the current state of the chessboard, and potential moves.

**2. Video Processing Loop**

The loop function runs continuously using requestAnimationFrame, ensuring real-time video frame processing.

• **Where the Loop is Defined**:

const loop = async () => {

  if (playingRef.current === false || invalidVideo(videoRef)) {

    centers = null; *// Stop processing if the video is invalid or paused*

  } else {

*// Main video processing logic*

  }

  requestId = requestAnimationFrame(loop); *// Continuously call the loop*

};

• **Purpose**:

• The loop ensures that each video frame is processed in real-time.

**3. Chessboard Detection and Transformation**

If centers is null, the chessboard is detected and transformations are calculated.

• **Where Chessboard Detection Happens**:

if (centers === null) {

  keypoints = getKeypoints(cornersRef, canvasRef); *// Get chessboard corners*

  const invTransform = getInvTransform(keypoints); *// Calculate inverse transform*

  [centers, centers3D] = transformCenters(invTransform); *// Transform piece centers*

  [boundary, boundary3D] = transformBoundary(invTransform); *// Transform chessboard boundaries*

  state = zeros(64, 12); *// Initialize the chessboard state*

  possibleMoves = new Set<string>(); *// Initialize possible moves*

  greedyMoveToTime = {}; *// Initialize greedy move timing*

}

• **Purpose**:

• This block initializes key data structures for further processing, such as:

• centers and boundary for the chessboard.

• state to store the current board state.

• possibleMoves to track valid moves.

**4. Frame Inference**

Each video frame is processed by the AI model to detect pieces.

• **Where Inference Happens**:

const {boxes, scores} = await detect(modelRef, videoRef, keypoints);

• **Purpose**:

• The detect function uses the AI model (modelRef) to predict:

• boxes: Bounding boxes for detected pieces.

• scores: Confidence scores for the detected pieces.

**5. Board State Update**

The detected boxes and scores are used to update the state of the chessboard.

• **Where State Update Happens**:

const squares: number[] = getSquares(boxes, centers3D, boundary3D); *// Map detections to squares*

const update: number[][] = getUpdate(scores, squares); *// Prepare state update*

state = updateState(state, update); *// Apply the update to the current state*

• **Purpose**:

• This step:

1. Maps detected bounding boxes to chessboard squares (getSquares).

2. Updates the confidence scores for each square and piece type.

**6. Move Processing**

The AI evaluates possible moves based on the updated state and determines the best move.

• **Where Move Processing Happens**:

const {bestScore1, bestScore2, bestJointScore, bestMove, bestMoves} = processState(state, movesPairsRef.current, possibleMoves);

• **Purpose**:

• The processState function evaluates:

• The best single move (bestMove).

• The best combined move pair (bestMoves).

• The scores for these moves to decide the most likely valid move.

**7. Move Execution**

Based on the processed moves, the function updates the chessboard state.

• **Where Moves are Executed**:

if ((bestMoves !== null) && (mode !== "play")) {

*// Execute the best joint move*

  const move: string = bestMoves.sans[0];

  hasMove = (bestScore2 > 0) && (bestJointScore > 0) && (possibleMoves.has(move));

  if (hasMove) {

    boardRef.current.move(move); *// Apply the move to the chessboard*

    possibleMoves.clear();

    greedyMoveToTime = {};

  }

}

• **Purpose**:

• Executes the best detected move on the chessboard if it meets certain criteria (e.g., valid score thresholds).

**8. Rendering and Cleanup**

The detected state is rendered on the canvas, and resources are managed to prevent memory leaks.

• **Where Rendering Happens**:

renderState(canvasRef.current, centers, boundary, state); *// Render the updated state*

setText([`FPS: ${fps}`, moveTextRef.current]); *// Update displayed text (e.g., FPS)*

tf.dispose([boxes, scores]); *// Release memory for tensors*

• **Purpose**:

• Visualizes the current chessboard state on the canvas.

• Displays additional information (e.g., FPS and move text).

• Ensures TensorFlow resources are released to prevent memory leaks.

**9. Return and Cleanup**

The function sets up the processing loop and ensures resources are cleaned up on termination.

• **Where Cleanup is Defined**:

return () => {

  tf.disposeVariables(); *// Dispose of all TensorFlow variables*

  if (requestId) {

    window.cancelAnimationFrame(requestId); *// Stop the animation loop*

  }

};

• **Purpose**:

• Stops the video processing loop.

• Releases all TensorFlow variables to free up memory.

**Summary**

1. **Initialization**: Sets up variables and prepares for video processing.

2. **Chessboard Detection**: Identifies the chessboard’s position and alignment.

3. **Frame Inference**: Uses the AI model to detect pieces in the current video frame.

4. **State Update**: Updates the chessboard’s state based on detections.

5. **Move Processing**: Evaluates and executes the best move detected.

6. **Rendering**: Visualizes the chessboard state on the canvas.

7. **Cleanup**: Manages resources and stops the loop on termination.

This detailed breakdown aligns with the “Overall Workflow” style. Let me know if you’d like further clarification on any part!