**Video Recording:**[**https://youtu.be/WntdH9o\_Sgs**](https://youtu.be/WntdH9o_Sgs)

**GitHub:**<https://github.com/Malcolmff207/ConnectedGaming.git>

**AnalyticsDashboard.cs**

**Start()** - Initializes the analytics dashboard by finding the AnalyticsManager instance, setting up UI button listeners, and hiding the dashboard panel initially.

**OpenDashboard()** - Displays the dashboard panel and refreshes the analytics data.

**CloseDashboard()** - Hides the dashboard panel.

**RefreshData()** - Fetches and displays current analytics data from the AnalyticsManager, including session stats, match results, popular pieces, opening moves, and DLC purchases.

**ClearStats()** - Removes all child statistics UI elements from the stats container.

**AddSeparator()** - Creates a visual spacer between different sections of statistics.

**AddHeader(string text)** - Creates a section header with formatted text for the statistics dashboard.

**AddStatItem(string label, string value)** - Creates a new statistics item with label and value, properly formatted in the dashboard.

**ShowStatus(string message)** - Updates the status text display and logs the message to the console.

## **AnalyticsManager.cs**

**Awake()** - Sets up the singleton instance, initializes Firebase Analytics, and resets session statistics tracking.

**Start()** - Subscribes to game events including new games, game endings, move executions, and network events.

**OnDestroy()** - Unsubscribes from all game events to prevent memory leaks and nullifies the singleton instance.

**ResetSessionStats()** - Clears and reinitializes the session statistics tracking dictionary.

**InitializeFirebase()** - Configures and initializes Firebase Analytics and Firebase Database connections.

**OnGameStarted()** - Tracks when a new game begins, incrementing counters and logging the event to Firebase.

**OnGameEnded()** - Records game completion, including the result and number of moves, storing data in Firebase.

**OnMoveExecuted()** - Tracks each chess move, updates statistics, and records opening moves and piece usage patterns.

**OnClientConnected(ulong clientId)** - Logs when a player connects to the networked game.

**OnClientDisconnected(ulong clientId)** - Records when a player disconnects from the networked game.

**StoreMatchStartData(string matchId)** - Saves initial match information to the Firebase database.

**StoreMatchResultData(string result, int moveCount)** - Records the outcome of a match in the Firebase database.

**StoreOpeningMoveData(string moveName, int moveNumber)** - Stores information about opening moves to track popular strategies.

**StoreMoveData(HalfMove move)** - Records detailed information about each move made during gameplay.

**LogDLCPurchase(string profileId, string profileName, int price)** - Logs when a player purchases a DLC item, including item details and price.

**LogGameStateSaved(string saveId, string saveName)** - Records when a player saves their game state.

**LogGameStateLoaded(string saveId, string saveName)** - Records when a player loads a previously saved game state.

**GetMostPopularOpeningMoves(int count, Action<Dictionary<string, int>> callback)** - Retrieves data about the most common opening moves from Firebase.

**GetTopPurchasedDLCItems(int count, Action<Dictionary<string, int>> callback)** - Fetches information about the most purchased DLC items.

**GetWinLossStatistics(Action<Dictionary<string, int>> callback)** - Retrieves statistics about wins, losses, and draws.

**GetMostUsedPieces(Action<Dictionary<string, int>> callback)** - Fetches data about which chess pieces are moved most frequently.

**GetCurrentSessionStats()** - Returns a copy of the current session's statistics.

**GetCurrentUserId()** - Retrieves a unique identifier for the current player, using network ID or device ID.

## DLCAnalyticsIntegration.cs

**Awake()** - Initializes references to the DLCManager and AnalyticsManager components.

**Start()** - Sets up initial tracking values and starts monitoring for purchase events.

**MonitorPurchaseEvents()** - Coroutine that periodically checks for changes in player credits and purchased profiles to detect new purchases.

**OnProfilePurchased(string profileId, int price)** - Logs purchase analytics when a new profile is detected.

**OnDLCPurchased(ProfilePicture profile)** - Direct method that can be called when a DLC item is purchased, logging the purchase to analytics.

## GameStateAnalyticsIntegration.cs

**Awake()** - Initializes references to the GameStateManager and AnalyticsManager components.

**Start()** - Begins monitoring for save/load events from the GameStateManager.

**MonitorGameStateEvents()** - Coroutine that continuously checks for active save/load panels and detects when game states are saved or loaded.

**OnGameSaved(string saveId, string saveName)** - Logs game save events to the analytics system.

**OnGameLoaded(string saveId, string saveName)** - Logs game load events to the analytics system.

## StatItemUI.cs

**SetData(string label, string value)** - Updates the UI elements of a statistics item with the provided label and value.  
  
ChessNetworkSync.cs

**Awake()** - Initializes the component, finding necessary manager references.

**OnNetworkSpawn()** - Initializes network synchronization, subscribes to network events, and syncs the initial game state if this is the server.

**OnNetworkDespawn()** - Cleans up event subscriptions when the network object is despawned.

**OnNewGameStarted()** - Synchronizes the game state when a new game begins.

**OnMoveExecuted()** - Synchronizes the game state after a move is executed.

**SyncCurrentGameState()** - Retrieves the current game state as a FEN string and updates the network variable.

**OnFENChanged(NetworkFEN previousValue, NetworkFEN newValue)** - Handles updates to the network game state by applying the new FEN string to the local game.

**LoadGameAndUpdateVisuals(string fenString)** - Coroutine that loads a game from FEN notation and updates the visual representation.

**UpdateVisualPieces()** - Recreates all visual chess pieces based on the current game state.

**ClearVisualPieces()** - Removes all existing visual chess pieces from the board.

**OnVisualPieceMoved(Square movedPieceInitialSquare, Transform movedPieceTransform, Transform closestBoardSquareTransform, Piece promotionPiece)** - Handles when a client moves a piece, validating the move and sending it to the server.

**RequestMoveServerRpc(string startSquare, string endSquare, string promotionPieceName)** - ServerRPC that receives move requests from clients and validates/executes them on the server.

**ForceRefreshFromServer()** - Forces a client to request the latest game state from the server.

**RequestCurrentGameStateServerRpc()** - ServerRPC that handles client requests for the current game state.

**GetPing()** - Measures and returns the current network latency (ping).

## **EnsureManagerReferences()** - Helper method that validates and initializes references to required game manager components. ChessNetworkUI.cs

**Start()** - Sets up network UI elements, button listeners, and subscribes to network events.

**Update()** - Continuously updates the connection status, ping display, and turn indicator.

**UpdateConnectionStatus()** - Updates the text displaying the current network connection status.

**UpdatePingDisplay()** - Calculates and displays the current network latency (ping).

**OnHostButtonClicked()** - Generates a session code and starts a network session as the host.

**OnClientButtonClicked()** - Attempts to join an existing session using the input session code.

**OnRejoinButtonClicked()** - Attempts to reconnect to the last session the player was part of.

**OnDisconnectButtonClicked()** - Disconnects from the current network session.

**OnClientConnected(ulong clientId)** - Handles when a client successfully connects to the session.

**OnClientDisconnect(ulong clientId)** - Handles when a client disconnects from the session.

**ShowConnectionPanel()** - Displays the connection panel and hides the game panel.

**ShowGamePanel()** - Displays the game panel and hides the connection panel.

**ShowErrorMessage(string message)** - Displays an error message to the user.

**GenerateSessionCode()** - Creates a random 6-character session code for network games.

**OnDestroy()** - Unsubscribes from network events to prevent memory leaks.

## GameEndDetector.cs

**Awake()** - Initializes references to required components.

**OnNetworkSpawn()** - Initializes the game end detector when spawned on the network, subscribing to game events.

**OnNetworkDespawn()** - Unsubscribes from events when despawned from the network.

**Update()** - Continuously checks for and displays game end messages based on network state.

**OnGameEndedEvent()** - Handles the end of a game, determining checkmate or stalemate conditions.

**OnMoveExecutedEvent()** - Checks for game end conditions after each move.

**CheckGameEndConditionsWithDelay(float delay)** - Coroutine that waits briefly then checks for checkmate or stalemate.

**OnGameEndStateChanged(GameEndState previousValue, GameEndState newValue)** - Handles updates to the network game end state.

**SetGameEndState(Side winnerSide, EndGameReason endReason)** - Updates the network variable with the game's end state information.

**DisplayGameEndMessageClientRpc(string winnerSide, int endReasonInt)** - ClientRPC that ensures all clients display the game end message.

**DisplayGameEndMessage(GameEndState state)** - Shows a formatted game end message (checkmate, stalemate, etc.) on the UI.

**OfferDraw()** - Public method for offering a draw.

**RequestDrawServerRpc()** - ServerRPC for requesting a draw from the server.

**ResignGame()** - Public method for resigning from the game.

**ResignGameServerRpc(Side resigningSide)** - ServerRPC for handling game resignation.

## GameManagerNullCheckFix.cs

**Awake()** - Initializes the component, adds protection against null references when moving pieces.

**OnDestroy()** - Cleans up event subscriptions.

**SafeOnPieceMoved(Square square, Transform pieceTransform, Transform squareTransform, Piece promotionPiece)** - A safety wrapper for the OnPieceMoved method that checks for null transforms before invoking the original method.

## NetworkChessPieceController.cs

**Awake()** - Initializes the component by finding a reference to the NetworkedChessController.

**Start()** - Subscribes to game events and initializes piece interactability.

**OnDestroy()** - Unsubscribes from game events.

**Update()** - Periodically checks and enforces correct piece interactability as turns change.

**UpdatePieceInteractability()** - Determines which side controls each piece and enables/disables pieces based on the current turn.

## NetworkedChessController.cs

**Awake()** - Sets up the singleton instance and validates references to the NetworkManager.

**Start()** - Initializes network components, subscribes to events, and configures the network manager.

**OnDestroy()** - Cleans up singleton reference and unsubscribes from events.

**OnClientConnected(ulong clientId)** - Handles client connection events, starting a new game if needed.

**OnClientDisconnected(ulong clientId)** - Handles client disconnection events.

**OnDisconnectWithReason(string reason)** - Handles disconnections that include a reason message.

**LogNetworkPerformance()** - Records network performance metrics such as ping.

**GetLocalPlayerSide()** - Determines which side (White or Black) this client controls.

## NetworkGameManager.cs

**Awake()** - Sets up the singleton instance and initializes references.

**OnNetworkSpawn()** - Initializes the network game manager when spawned on the network, subscribing to events.

**OnNetworkDespawn()** - Cleans up event subscriptions when despawned from the network.

**OnVisualPieceMoved(Square movedPieceInitialSquare, Transform movedPieceTransform, Transform closestBoardSquareTransform, Piece promotionPiece)** - Handles chess piece movements, validates turns, and sends moves to the server.

**SendMoveToServerServerRpc(string startSquare, string endSquare, string promotionPieceType)** - ServerRPC that receives move requests from clients.

**BroadcastMoveClientRpc(string startSquare, string endSquare, string promotionPieceType)** - ClientRPC that broadcasts moves to all clients.

**OnMoveExecuted()** - Toggles the current turn after a move is executed.

**OnClientConnected(ulong clientId)** - Handles new client connections, syncing game state if a game is in progress.

**SyncCurrentGameStateClientRpc()** - ClientRPC that triggers clients to sync with the current game state.

**SyncCurrentGameStateToClientClientRpc(ulong clientId)** - Targeted ClientRPC that syncs a specific client with the game state.

**GetCurrentPing()** - Measures and returns the current network latency.

## NetworkSessionManager.cs

**Awake()** - Sets up the singleton instance and configures the component for network session management.

**Start()** - Subscribes to network events for tracking connections.

**OnDestroy()** - Cleans up network event subscriptions.

**CreateSession(string sessionCode)** - Creates a new network session as host with the given session code.

**JoinSession(string sessionCode)** - Joins an existing session using the provided session code.

**RejoinSession()** - Attempts to reconnect to the last session the player was part of.

**ReconnectCoroutine()** - Coroutine that attempts multiple reconnection attempts with increasing delays.

**LeaveSession()** - Disconnects from the current session.

**OnClientConnected(ulong clientId)** - Handles when a client successfully connects to a session.

**OnClientDisconnect(ulong clientId)** - Handles when a client disconnects from a session.

**GetClientSide(ulong clientId)** - Determines which side (White or Black) a specific client is playing.

## DLCManager.cs

**Awake()** - Sets up the singleton instance and validates UI references.

**Start()** - Initializes Firebase, sets up event listeners, loads player data, and configures the UI.

**ValidateReferences()** - Checks that all required references are properly assigned.

**OnNetworkSpawn()** - Handles network object spawning, setting up network variables and synchronizing initial profile data.

**OnDestroy()** - Cleans up event subscriptions and the singleton reference.

**OnProfileDataChanged()** - Handles when a player's profile data changes across the network.

**SyncProfileWithNetwork()** - Synchronizes the local player's profile data with the network.

**RequestProfileRefresh()** - Requests a refresh of profile data for all connected players.

**InitializeFirebase()** - Configures and initializes the Firebase SDK for storage and database access.

**LoadProfilePicturesFromFirebase()** - Retrieves profile pictures from Firebase Storage.

**LoadFallbackItems()** - Creates fallback profile items if Firebase loading fails.

**GetPriceForProfile()** - Determines a price for a profile based on its piece type.

**FormatProfileName()** - Formats a raw profile name into a display-friendly format.

**PopulateStoreItems()** - Creates UI elements for each available profile picture in the store.

**LoadImagePreview()** - Coroutine that loads an image from Firebase Storage and displays it in a UI element.

**PurchaseProfilePicture()** - Handles the purchase of a profile picture, checking for ownership and credits.

**PurchaseProfilePictureCoroutine()** - Coroutine that processes a profile picture purchase.

**DownloadProfilePicture()** - Coroutine that downloads a full-size profile picture from Firebase.

**SaveTextureToFile()** - Saves a downloaded texture to the device's persistent storage.

**SelectProfilePicture()** - Handles when a player selects a profile picture, updating local state and the network.

**UpdateProfileServerRpc()** - ServerRPC that updates a player's profile on the server.

**NotifyProfileUpdateClientRpc()** - ClientRPC that notifies all clients about a profile update.

**UpdatePlayerProfileImage()** - Updates the local player's profile image based on the selected profile.

**LoadImageFromFile()** - Coroutine that loads an image from a local file and applies it to a UI element.

**RecordPurchaseInFirebase()** - Records a DLC purchase in the Firebase database.

**NotifyProfileUpdateServerRpc()** - ServerRPC for notifying about profile updates.

**NotifyProfileUpdateClientRpc()** - ClientRPC for notifying clients about profile updates.

**SavePlayerData()** - Saves player data to PlayerPrefs.

**LoadPlayerData()** - Loads player data from PlayerPrefs.

**UpdatePlayerCreditsUI()** - Updates the UI display of the player's current credits.

**ShowStatus()** - Displays a status message and logs it to the console.

**SetupUI()** - Configures the initial state of UI panels.

**OpenStore()** - Opens the DLC store UI and loads available profile pictures.

**CloseStore()** - Closes the DLC store UI.

**AddCredits()** - Debug method for adding credits to the player's account.

## GameStateManager.cs

**Awake()** - Sets up the singleton instance.

**Start()** - Initializes Firebase and starts the auto-save coroutine if enabled.

**OnDestroy()** - Cleans up event subscriptions and the singleton reference.

**InitializeFirebase()** - Configures and initializes Firebase for game state storage.

**AutoSaveCoroutine()** - Coroutine that periodically saves the game state if it has changed.

**OnGameEnded()** - Automatically saves the final game state when a game ends.

**SaveCurrentGame()** - Manually saves the current game state with an optional name.

**SaveGameToFirebase()** - Stores a game state in Firebase with the given name and FEN string.

**SaveGameCoroutine()** - Coroutine that handles the Firebase save operation.

**LoadSavedGame()** - Loads a previously saved game by its ID.

**LoadSavedGameCoroutine()** - Coroutine that retrieves a saved game from Firebase and loads it.

**LoadGame()** - Loads a game state from a FEN string, handling network synchronization if needed.

**GetSavedGames()** - Returns the list of saved games.

**LoadSavedGamesCoroutine()** - Coroutine that retrieves the list of saved games from Firebase.

**DeleteSavedGame()** - Deletes a saved game from Firebase.

**DeleteSavedGameCoroutine()** - Coroutine that handles the Firebase delete operation.

**RequestLoadGame()** - Requests the server to load a game state (placeholder for network implementation).

**LoadGameForAllClients()** - Broadcasts a game state to all clients (placeholder for network implementation).

**GetCurrentUserId()** - Retrieves a unique identifier for the current player.

## GameUIDLCIntegration.cs

**Start()** - Initializes references, verifies UI components, sets up button listeners, and subscribes to network events.

**OnDestroy()** - Cleans up button listeners and network event subscriptions.

**VerifyUIComponents()** - Validates that all required UI components are assigned, attempting to find them if not.

**LoadInitialProfileImage()** - Loads the player's previously selected profile image on startup.

**OnClientConnected()** - Handles when a client connects to the network, updating UI and sharing profile data. **OnClientDisconnect()** - Handles when a client disconnects from the network, resetting UI elements.

**RequestAllProfileRefreshes()** - Requests all connected players to refresh their profile data.

**DelayedProfileRefresh()** - Coroutine that waits briefly before requesting profile refreshes.

**ShareProfileWithNewPlayer()** - Shares the local player's profile with a newly connected player.

**TriggerNetworkWideProfileRefreshServerRpc()** - ServerRPC that requests all clients to refresh their profiles.

**TriggerProfileRefreshClientRpc()** – ClientRPC that triggers profile refresh on all clients.

**UpdatePlayerName()** - Updates the player's name based on their network role.

**OpenDLCStore()** - Opens the DLC store UI.

**OnProfileSelected()** - Handles when a player selects a profile picture, updating UI and network state.

**UpdateOtherPlayerProfile()** - Updates the UI to show another player's profile. **LoadProfilePictureForImage()** - Coroutine that loads a profile picture for a specific UI image element.

**TryLoadFromDLCManager()** - Helper method that attempts to load a profile image via the DLCManager.

**LoadFallbackImage()** - Loads a fallback image based on profile ID when the actual image isn't available.

**LoadPlaceholderImage()** - Loads a placeholder image from resources as a last resort.

**GetDefaultPlayerName()** - Determines a default player name based on network role.

**LogStatus()** - Displays a status message and logs it to the console.

## NetworkPlayerSetup.cs

**Awake()** - Sets up the singleton instance for network player setup.

**OnNetworkSpawn()** - Initializes the network player setup when spawned on the network.

**OnNetworkDespawn()** - Cleans up event subscriptions when despawned from the network.

**OnClientConnectedCallback()** - Handles client connection events, ensuring each client has a properly set up player object.

**CreatePlayerObjectForClient()** - Creates a network player object for a client if one doesn't exist.

**EnsurePlayerObjectHasComponents()** - Adds required components to a player object to ensure proper functionality.

**TriggerProfileSyncAfterDelay()** - Coroutine that waits briefly before triggering profile synchronization.

**VerifyAllPlayerObjects()** - Manually checks and fixes player objects for all connected clients.

**TriggerProfileSharingClientRpc()** - ClientRPC that requests existing clients to share their profiles with a new client.

## ProfilePicture.cs

**ProfilePicture()** - Default constructor that creates an empty profile picture data object.

**ProfilePicture(string id, string name, int price, string imageUrl)** - Parameterized constructor that creates a profile picture with specified values.

## SavedGameItemUI.cs

**Setup()** - Configures the UI elements of a saved game item with the provided information.

**OnLoadButtonClicked()** - Handler for when the load button is clicked, invoking the OnLoadClicked event.

**OnDeleteButtonClicked()** - Handler for when the delete button is clicked, invoking the OnDeleteClicked event.

## StoreItemUI.cs

**OnEnable()** - Validates references when the store item becomes active.

**ValidateReferences()** - Checks that all required UI components are assigned and logs errors if not.

**Setup()** - Configures the store item UI with profile data and purchase/selection state.

**OnPurchaseButtonClicked()** - Handler for when the purchase button is clicked, invoking the OnPurchaseClicked event.

**OnSelectButtonClicked()** - Handler for when the select button is clicked, invoking the OnSelectClicked event.

**OnDestroy()** - Cleans up button listeners when the component is destroyed.