# Handover Document: SGFPlayerClean (v3.125)

**Date:** December 18, 2025
**System Time:** Thursday, 12:25 AM (PST)
**Focus:** OGS Challenge Creation, UI Consistency, and Architecture Refactor

## 1. Executive Summary

The OGS integration is functionally complete (Creation, Listing, Joining, Canceling). Persistence and Logic constraints are working.
**Critical Pivot:** We have determined that the visual inconsistency (opacity) of the "New Challenge" window is caused by using a macOS `.sheet`. To achieve the desired "Frosted Glass" look, we must abandon the Sheet and refactor the view to use a **ZStack Overlay**, mimicking the architecture of the existing `SettingsPanel`.

## 2. Key Accomplishments

- **API Stability:** Fixed HTTP 500 errors by identifying that OGS requires the `system` parameter in `time_control_parameters` (e.g., `"system": "byoyomi"`).
- **Lifecycle Management:** Implemented `cancelChallenge` with a dynamic UI that swaps "Accept" (Green) for "Cancel" (Yellow) based on ownership.
- **User Experience:** Added persistence for challenge settings (using `UserDefaults`) and logic clamping for Rank ranges.

## 3. The Opacity Problem & Solution Strategy

We attempted to force a `.sheet` to look like a floating panel using `.presentationBackground(.clear)` and custom styles. This failed because:

1. **Window Isolation:** A Sheet creates a separate `NSWindow`.
2. **System Dimming:** macOS imposes a dimming layer behind modal windows. Our transparent view was looking at this dark dimming layer, making it appear gray/opaque.

**The Solution: "The Overlay Pattern"**

We must adopt the pattern used by `SettingsPanel`.

- **Old Way (Current):** `OGSBrowserView` presents a `.sheet`.
- **New Way (Required):** The `RightPanelView` (or `ContentView`) must contain a `ZStack`. The "New Challenge" view will render strictly as a View layer *on top* of the list, sharing the main window's rendering context. This eliminates the dimming layer and allows true vibrancy.

## 4. Technical Reference: The "Golden" JSON

To prevent regression of the HTTP 500 Error,
ensure `OGSModels.swift` always produces a payload matching this structure (specifically the `system` key):

code
JSON

```
{
  "game": {
    "name": "Friendly Match",
    "rules": "japanese",
    "ranked": true,
    "width": 19, "height": 19,
    "time_control": "byoyomi",
    "time_control_parameters": {
      "time_control": "byoyomi",
      "system": "byoyomi", // <--- CRITICAL: OGS server
crashes without this
      "main_time": 600,
      "period_time": 30,
      "periods": 5
    }
  },
  "challenger_color": "automatic",
  "min_ranking": 0,
  "max_ranking": 38
}
```

## 5. Files to Upload for Next Session

The success of the refactor depends on understanding
where `SettingsPanel` lives in the view hierarchy.

1. **`SettingsPanel.swift`:** (Reference) Contains the visual style we want to clone.
2. **`SupportingViews.swift`:** (Reference) Contains the `.frostedGlassStyle()` definition.
3. **`RightPanelView.swift`:** (Target) This is likely where the new Overlay should live.
4. **`ContentView.swift`:** (Target) We need to check if `SettingsPanel` is triggered here. If so, we might need to hoist the Challenge view state up to this level.
5. **`OGSBrowserView.swift`:** (Current Parent) We will need to rip out the `.sheet` logic from here.
6. **`OGSCreateChallengeView.swift`:** (The View) Needs to be adapted to work as an inline overlay (e.g., removing `presentationBackground` modifiers which are sheet-specific).

## 6. Kickoff Prompt for Next Session

"I am continuing the SGFPlayerClean project.
**Context:** The OGS Challenge Creation logic works, but the UI is implemented as a `.sheet`, which causes unfixable opacity issues. We decided to refactor this into a **ZStack Overlay** to match the visual style of the `SettingsPanel`.
**Goal:**

1. Analyze `SettingsPanel.swift` and `ContentView.swift` to understand how the app handles overlays.
2. Refactor `OGSCreateChallengeView` to be an overlay instead of a sheet.
3. Integrate this overlay into `RightPanelView` (or whichever parent view holds the stack), allowing it to float over the Lobby list with the correct transparent background.
   **Attached**
   **Files:** `SettingsPanel.swift`, `SupportingViews.swift`, `RightPanelView.swift`, `OGSBrowserView.swift`, `OGSCreateChallengeView.swift`."

## 7. Known Technical Debt

- **Automatch:** `OGSClient.swift` contains a stub

function `startAutomatch()` to satisfy a dependency in `OGSGameViewModel`. This feature is currently non-functional.

- **Error Handling:** Challenge creation errors are displayed in the UI, but silent network failures (timeouts) rely on generic console logging.

Use Arrow Up and Arrow Down to select a turn, Enter to jump to it, and Escape to return to the chat.

User

I am continuing the SGFPlayerClean project.
Current Status:
The OGS Challenge Creation logic is complete and working (API, Persistence, Rank Logic are all good). However, the UI implementation using a .sheet is causing unfixable opacity issues because of the system's mandatory window background.
The Goal:
We need to refactor the 'New Challenge' view to move from a Sheet architecture to a ZStack Overlay architecture. This must match the exact visual style and behavior of the existing SettingsPanel.
Please review the attached files:
SettingsPanel.swift (Reference: This has the correct transparent overlay behavior we want to mimic).
SupportingViews.swift (Reference: Contains the .frostedGlassStyle definition).
RightPanelView.swift and OGSBrowserView.swift (Context: This is where the new overlay needs to live).
OGSCreateChallengeView.swift (Target: This needs to be stripped of sheet-specific modifiers and adapted to work as an overlay).
Task:
Refactor the code to remove the .sheet modifier from OGSBrowserView and instead implement OGSCreateChallengeView as a ZStack overlay inside RightPanelView (or the appropriate parent), ensuring it looks visually identical to the SettingsPanel transparency.

Here is some additional information that may be useful.

Handover Document: SGFPlayerClean (v3.125)
Date: December 18, 2025

System Time: Thursday, 12:25 AM (PST)
Focus: OGS Challenge Creation, UI Consistency, and Architecture Refactor

## 1 Executive Summary

The OGS integration is functionally complete (Creation, Listing, Joining, Canceling). Persistence and Logic constraints are working.
Critical Pivot: We have determined that the visual inconsistency (opacity) of the "New Challenge" window is caused by using a macOS .sheet. To achieve the desired "Frosted Glass" look, we must abandon the Sheet and refactor the view to use a ZStack Overlay, mimicking the architecture of the existing SettingsPanel.

## 2 Key Accomplishments

- API Stability: Fixed HTTP 500 errors by identifying that OGS requires the system parameter
in time_control_parameters (e.g., "system": "byoyomi").
- Lifecycle Management: Implemented cancelChallenge with a dynamic UI that swaps "Accept" (Green) for "Cancel" (Yellow) based on ownership.
- User Experience: Added persistence for challenge settings (using UserDefaults) and logic clamping for Rank ranges.

## 3 The Opacity Problem & Solution Strategy

We attempted to force a .sheet to look like a floating panel using .presentationBackground(.clear) and custom styles. This failed because:

1. Window Isolation: A Sheet creates a separate NSWindow.
2. System Dimming: macOS imposes a dimming layer behind modal windows. Our transparent view was looking at this dark dimming layer, making it appear gray/opaque.

The Solution: "The Overlay Pattern"
We must adopt the pattern used by SettingsPanel.

- Old Way (Current): OGSBrowserView presents a .sheet.
- New Way (Required): The RightPanelView (or ContentView) must contain a ZStack. The "New Challenge" view will render strictly as a View layer on top of the list, sharing the main window's rendering context. This eliminates the dimming layer and allows true vibrancy.

## 4 Technical Reference: The "Golden" JSON

---

To prevent regression of the HTTP 500 Error, ensure OGSModels.swift always produces a payload matching this structure (specifically the system key):

code

JSON

```
{
"game": {
"name": "Friendly Match",
"rules": "japanese",
"ranked": true,
"width": 19, "height": 19,
"time_control": "byoyomi",
"time_control_parameters": {
"time_control": "byoyomi",
"system": "byoyomi", // <-- CRITICAL: OGS server crashes without this
"main_time": 600,
"period_time": 30,
"periods": 5
}
},
"challenger_color": "automatic",
"min_ranking": 0,
"max_ranking": 38
}
```

## 5. Files to Upload for Next Session

The success of the refactor depends on

understanding where SettingsPanel lives in the view hierarchy.
1 SettingsPanel.swift: (Reference) Contains the visual style we want to clone.
2 SupportingViews.swift: (Reference) Contains the .frostedGlassStyle() definition.
3 RightPanelView.swift: (Target) This is likely where the new Overlay should live.
4 ContentView.swift: (Target) We need to check if SettingsPanel is triggered here. If so, we might need to hoist the Challenge view state up to this level.
5 OGSBrowserView.swift: (Current Parent) We will need to rip out the .sheet logic from here.
6 OGSCreateChallengeView.swift: (The View) Needs to be adapted to work as an inline overlay (e.g., removing presentationBackground modifiers which are sheet-specific).
6. Kickoff Prompt for Next Session
"I am continuing the SGFPlayerClean project. Context: The OGS Challenge Creation logic works, but the UI is implemented as a .sheet, which causes unfixable opacity issues. We decided to refactor this into a ZStack

Overlay to match the visual style of the SettingsPanel.
Goal:
1 Analyze SettingsPanel.swift and ContentView.swift to understand how the app handles overlays.
2 Refactor OGSCreateChallengeView to be an overlay instead of a sheet.
3 Integrate this overlay into RightPanelView (or whichever parent view holds the stack), allowing it to float over the Lobby list with the correct transparent background.
Attached Files: SettingsPanel.swift, SupportingViews.swift, RightPanelView.swift, OGSBrowserView.swift, OGSCreateChallengeView.swift."
7. Known Technical Debt
- Automatch: OGSClient.swift contains a stub function startAutomatch() to satisfy a dependency in OGSGameViewModel. This feature is currently non-functional.
- Error Handling: Challenge creation errors are displayed in the UI, but silent network

# failures (timeouts) rely on generic console logging.

//
//  SettingsPanel.swift
//  SGFPlayerClean
//
//  Created: 2025-12-01
//  Purpose: Slide-out settings drawer with visual preferences and library management
//

import SwiftUI

struct SettingsPanel: View {
@ObservedObject var app: AppModel
@ObservedObject var settings = AppSettings.shared
@Binding var isPresented: Bool

code
Code

```
@State private var isMarkersExpanded = true

// Logarithmic binding for Time Delay
private var delayBinding: Binding<Double> {
    Binding<Double>(
        get: {
            let y = max(0.1, settings.moveInterval)
            return log10(y / 0.1) / log10(100.0)
        },
        set: { sliderValue in
            let y = 0.1 * pow(100.0, sliderValue)
            settings.moveInterval = y
        }
    )
}

var body: some View {
```

```
    ZStack(alignment: .leading) {
        // 1. Dimmed Background
        Color.black.opacity(0.2)
            .ignoresSafeArea()
            .onTapGesture { close() }

        // 2. The Sidebar Panel
        VStack(alignment: .leading, spacing: 0) {
            headerView

            ScrollView {
                VStack(alignment: .leading, spacing: 24) {
                    playbackSection

Divider().background(Color.white.opacity(0.3))
                    markersSection

Divider().background(Color.white.opacity(0.3))
                    librarySection
                }
                .padding()
            }

            Spacer()
            footerView
        }
        .frame(width: 350)
        .frostedGlassStyle()
        .padding(.leading, 10)
        .padding(.vertical, 10)
        .transition(.move(edge: .leading))
    }
    .zIndex(100)
}

// MARK: - Components

private var headerView: some View {
    HStack {
        Text("Settings")
            .font(.title2.bold())
```

```
                .foregroundColor(.white)
            Spacer()
            Button(action: close) {
                Image(systemName: "xmark.circle.fill")
                    .font(.title2)
                    .foregroundColor(.white.opacity(0.6))
            }
            .buttonStyle(.plain)
        }
        .padding()
        .background(Color.black.opacity(0.1))
    }

    private var playbackSection: some View {
        VStack(alignment: .leading, spacing: 12) {
            Label("Playback", systemImage: "play.circle.fill")
                .font(.headline).foregroundColor(.white)

            // Delay Slider
            VStack(alignment: .leading, spacing: 4) {
                HStack {
                    Text("Move Delay")
                    Spacer()
                    Text(String(format: "%.1fs",
settings.moveInterval))
                        .monospacedDigit().foregroundColor(.whi
te)
                }
                .font(.caption).foregroundColor(.white.opacity(
0.8))
                Slider(value: delayBinding, in:
0.0...1.0).tint(.cyan)
            }

            // Jitter Slider
            VStack(alignment: .leading, spacing: 4) {
                HStack {
                    Text("Stone Jitter")
                    Spacer()
                    Text(String(format: "%.2f",
settings.jitterMultiplier))
```

```
                    .monospacedDigit().foregroundColor(.whi
te)
                }
                .font(.caption).foregroundColor(.white.opacity(
0.8))
                Slider(value: $settings.jitterMultiplier, in:
0.0...2.0, step: 0.05).tint(.cyan)
            }

            Group {
                Toggle("Shuffle Games", isOn:
$settings.shuffleGameOrder)
                Toggle("Start on Launch", isOn:
$settings.startGameOnLaunch)
            }
            .toggleStyle(SwitchToggleStyle(tint: .cyan))
            .font(.caption).foregroundColor(.white)
        }
    }

    private var markersSection: some View {
        DisclosureGroup("Last move markers", isExpanded:
$isMarkersExpanded) {
            VStack(alignment: .leading, spacing: 12) {

                // --- SLIDERS HIDDEN (Hardcoded preferences)
---
                /*
                VStack(alignment: .leading, spacing: 4) {
                    HStack {
                        Text("Panel Opacity")
                        Spacer()
                        Text(String(format: "%.0f%%",
settings.panelOpacity * 100))
                            .monospacedDigit().foregroundColor(
.white)
                    }
                    .font(.caption).foregroundColor(.white.opac
ity(0.8))
                    Slider(value: $settings.panelOpacity, in:
0.0...1.0).tint(.orange)
```

```
            }

            VStack(alignment: .leading, spacing: 4) {
                HStack {
                    Text("Glass Blur")
                    Spacer()
                    Text(String(format: "%.1f",
settings.panelDiffusiveness))
                        .monospacedDigit().foregroundColor(
.white)
                }
                .font(.caption).foregroundColor(.white.opac
ity(0.8))
                Slider(value: $settings.panelDiffusiveness,
in: 0.0...1.0).tint(.purple)
            }
            .padding(.bottom, 8)
            */
            //
----------------------------------------

            Group {
                Toggle("Move Numbers", isOn:
$settings.showMoveNumbers)
                Toggle("Dot", isOn:
$settings.showLastMoveDot)
                Toggle("Circle", isOn:
$settings.showLastMoveCircle)
                Toggle("Board Glow", isOn:
$settings.showBoardGlow)
                Toggle("Enhanced Effects", isOn:
$settings.showEnhancedGlow)
                Toggle("Drop Stone Animation", isOn:
$settings.showDropInAnimation)
                    .disabled(app.viewMode == .view2D)
                    .foregroundColor(app.viewMode
== .view2D ? .white.opacity(0.4) : .white)
            }
        }
        .toggleStyle(SwitchToggleStyle(tint: .cyan))
        .font(.caption)
```

```
            .foregroundColor(.white)
            .padding(.leading, 10)
            .padding(.top, 5)
        }
        .font(.headline).foregroundColor(.white).accentColor(.w
hite)
    }

    private var librarySection: some View {
        VStack(alignment: .leading, spacing: 10) {
            Label("Library", systemImage:
"books.vertical.fill")
                .font(.headline).foregroundColor(.white)

            Button(action: { app.promptForFolder() }) {
                HStack {
                    Image(systemName: "folder.badge.plus")
                    Text("Choose Folder")
                }
                .frame(maxWidth: .infinity).padding(8)
                .background(Color.white.opacity(0.15)).cornerRa
dius(6)
            }
            .buttonStyle(.plain)

            if !app.games.isEmpty {
                ScrollView {
                    LazyVStack(alignment: .leading, spacing: 0)
{
                        ForEach(app.games) { wrapper in
                            Button(action:
{ app.selectGame(wrapper) }) {
                                GameListRow(wrapper: wrapper,
isSelected: app.selection?.id == wrapper.id)
                            }
                            .buttonStyle(.plain)

                            Divider().background(Color.white.opacity(0.1))
                        }
                    }
                }
```

```
            .frame(height: 250)
            .background(Color.black.opacity(0.2)).cornerRad
ius(6)
        } else {
            Text("No games
loaded").font(.caption).foregroundColor(.gray)
        }
    }
}

private var footerView: some View {
    HStack {
        Text("SGFPlayer Clean v1.0.47")
            .font(.caption2).foregroundColor(.white.opacity
(0.5))
        Spacer()
    }
    .padding()
    .background(Color.black.opacity(0.2))
}

private func close() {
    withAnimation(.easeInOut(duration: 0.45)) { isPresented
= false }
}
}
```

struct GameListRow: View {
let wrapper: SGFGameWrapper
let isSelected: Bool

code
Code

```
var body: some View {
    VStack(alignment: .leading, spacing: 2) {
        HStack {
            if isSelected {
                Image(systemName: "play.fill")
                    .font(.caption2)
```

```
                        .foregroundColor(.cyan)
                }

                Text(wrapper.game.info.playerBlack ?? "?")
                    .fontWeight(.bold) +
                Text(" vs ") +
                Text(wrapper.game.info.playerWhite ?? "?")
                    .fontWeight(.bold)

                Spacer()
            }
            .font(.caption)
            .foregroundColor(isSelected ? .cyan : .white)

            HStack {
                Text(wrapper.game.info.date ?? "Unknown
Date").font(.caption2).foregroundColor(.gray)
                Spacer()
                Text(wrapper.game.info.result ??
"").font(.caption).bold().foregroundColor(.yellow)
            }
        }
        .padding(.vertical, 6)
        .padding(.horizontal, 8)
        .background(isSelected ? Color.white.opacity(0.15) :
Color.clear)
    }
}


//
//  SupportingViews.swift
//  SGFPlayerClean
//
//  Created: 2025-11-28
//  Purpose: Shared UI components used across 2D and 3D views
//

import SwiftUI
```

// MARK: - Shared Visual Style
struct FrostedGlass: ViewModifier {
@ObservedObject var settings = AppSettings.shared

code
Code

```
// Tint Color: Darker, Bluer version of Tatami
let tintColor = Color(red: 0.05, green: 0.2, blue: 0.15)

func body(content: Content) -> some View {
    content
        // 1. The Blur Effect (Diffusiveness)
        // We assume .ultraThinMaterial has a fixed radius.
        // By fading it out (opacity < 1.0), we blend the
sharp background
        // with the blurred background, simulating a "lower
radius" blur.
        .background(
            Rectangle()
                .fill(.ultraThinMaterial)
                .opacity(settings.panelDiffusiveness) //
0.0 = Sharp, 1.0 = Frosted
        )
        // 2. The Color Tint (Opacity)
        .background(tintColor.opacity(settings.panelOpacity
))
        .cornerRadius(12)
        .overlay(
            RoundedRectangle(cornerRadius: 12)
                .stroke(Color.white.opacity(0.15),
lineWidth: 1)
        )
}
}

extension View {
func frostedGlassStyle() -> some View {
modifier(FrostedGlass())
}
```

```
}

// MARK: - GameInfoCard
struct GameInfoCard: View {
@ObservedObject var boardVM: BoardViewModel
let ogsVM: OGSGameViewModel?

code
Code

var body: some View {
    VStack(alignment: .leading, spacing: 10) {
        // Header
        HStack {
            Text("Game Info")
                .font(.headline)
                .foregroundColor(.white.opacity(0.9))
            Spacer()
            if let phase = ogsVM?.gamePhase, phase != 
"none" {
                Text(phase.capitalized)
                    .font(.caption2)
                    .padding(.horizontal, 6)
                    .padding(.vertical, 2)
                    .background(Color.blue.opacity(0.6))
                    .cornerRadius(4)
            }
        }

        Divider().background(Color.white.opacity(0.2))

        // Metadata
        if let game = boardVM.currentGame?.game {
            GameMetadataView(game: game)
            Divider().background(Color.white.opacity(0.2))
        }

        // Stats
        StatRow(label: "Move", icon: "arrow.right.circle", 
value: "\(boardVM.currentMoveIndex)", isMonospaced: true)
```

```
        StatRow(label: "Black Captures", icon:
"circle.fill", value: "\(boardVM.blackCapturedCount)")
        StatRow(label: "White Captures", icon: "circle",
value: "\(boardVM.whiteCapturedCount)")
    }
    .padding(12)
    .frostedGlassStyle() // Applied shared style
}
}

// Helper to display Player Names, Ranks, Result, and Date
struct GameMetadataView: View {
let game: SGFGame

code
Code

var body: some View {
    VStack(alignment: .leading, spacing: 6) {
        // Black Player
        HStack {
            Text("Black:").foregroundColor(.gray)
            Text(game.info.playerBlack ?? "Unknown").bold()
            if let rank = game.info.blackRank {
                Text("(\
(rank))").font(.caption).foregroundColor(.white.opacity(0.7
))
            }
        }
        .font(.caption).foregroundColor(.white)

        // White Player
        HStack {
            Text("White:").foregroundColor(.gray)
            Text(game.info.playerWhite ?? "Unknown").bold()
            if let rank = game.info.whiteRank {
                Text("(\
(rank))").font(.caption).foregroundColor(.white.opacity(0.7
))
            }
```

```
        }
        .font(.caption).foregroundColor(.white)

        // Result & Date
        HStack {
            Text(game.info.date ?? "").font(.caption2).foregroundColor(.gray)
            Spacer()
            Text(game.info.result ?? "").font(.caption2).bold().foregroundColor(.yellow)
        }
        .padding(.top, 2)
    }
}
}

struct StatRow: View {
let label: String
let icon: String
let value: String
var isMonospaced: Bool = false

code
Code

var body: some View {
    HStack {
        Label(label, systemImage: icon).font(.caption)
        Spacer()
        Text(value)
            .font(isMonospaced ? .system(.caption, design: .monospaced) : .caption)
    }
    .foregroundColor(.white.opacity(0.9))
}
}

// MARK: - PlaybackControls
struct PlaybackControls: View {
```

@ObservedObject var boardVM: BoardViewModel

code
Code

```
var body: some View {
    HStack(spacing: 12) {

        // 1. Buttons
        HStack(spacing: 4) {
            controlButton("backward.end.fill", help: "Start") { boardVM.goToStart() }
            controlButton("backward.fill", help: "-1 Move") { boardVM.previousMove() }

            Button(action: { boardVM.toggleAutoPlay() }) {
                Image(systemName: boardVM.isAutoPlaying ? "pause.fill" : "play.fill")
                    .font(.system(size: 14, weight: .bold))
                    .frame(width: 28, height: 28)
                    .background(Color.white.opacity(0.1))
                    .clipShape(Circle())
            }
            .buttonStyle(.plain)
            .help("Play/Pause")

            controlButton("forward.fill", help: "+1 Move") { boardVM.nextMove() }
            controlButton("forward.end.fill", help: "End") { boardVM.goToEnd() }
        }

        Divider().frame(height: 20).background(Color.white.opacity(0.3))

        // 2. Slider
        Slider(value: Binding(
            get: { Double(boardVM.currentMoveIndex) },
            set: { boardVM.seekToMove(Int($0)) }
        ), in: 0...Double(boardVM.totalMoves))
```

```
            .tint(.white)
            .frame(width: 200)
        }
        .padding(.horizontal, 12)
        .padding(.vertical, 8)
        .frostedGlassStyle() // Applied shared style
    }

    private func controlButton(_ icon: String, help: String,
    action: @escaping () -> Void) -> some View {
        Button(action: action) {
            Image(systemName: icon)
                .font(.system(size: 12))
                .foregroundColor(.white.opacity(0.9))
                .frame(width: 24, height: 24)
                .contentShape(Rectangle())
        }
        .buttonStyle(.plain)
        .help(help)
    }
}

// MARK: - Loading View
struct LoadingView: View {
var body: some View {
VStack {
ProgressView().scaleEffect(1.5).padding()
Text("Loading Games...").font(.headline).foregroundColor(.white)
}
.frame(maxWidth: .infinity, maxHeight: .infinity)
.background(Color.black.opacity(0.4))
}
}

// MARK: - Tatami Background
struct TatamiBackground: View {
var body: some View {
ZStack {
Color(red: 0.90, green: 0.85, blue: 0.72)
```

```
Image("tatami")
.resizable()
.aspectRatio(contentMode: .fill)
.frame(minWidth: 0, maxWidth: .infinity, minHeight: 0, maxHeight: .infinity)
.clipped()
}
.ignoresSafeArea()
}
}


//
//  RightPanelView.swift
//  SGFPlayerClean
//
//  v3.108: Wiring Challenge Creation.
//  - Pass $app.isCreatingChallenge to OGSBrowserView.
//

import SwiftUI

enum RightPanelTab: String, CaseIterable {
case local = "Local"
case online = "Online"
}

struct RightPanelView: View {
@ObservedObject var app: AppModel
@ObservedObject var boardVM: BoardViewModel

code
Code

@State private var selectedTab: RightPanelTab = .online

var body: some View {
    VStack(spacing: 0) {
```

```
        // 1. TOP TOGGLE
        Picker("Mode", selection: $selectedTab) {
            Text("Local").tag(RightPanelTab.local)
            Text("Online").tag(RightPanelTab.online)
        }
        .pickerStyle(.segmented)
        .padding(10)
        .background(Color.black.opacity(0.1))

        Divider().background(Color.white.opacity(0.1))

        // 2. CONTENT AREA
        ZStack {
            if selectedTab == .local {
                VStack(spacing: 0) {
                    // Fixed Metadata Panel at the top
                    if let game = app.selection {
                        LocalGameMetadataView(game: game, boardVM: boardVM)
                            .transition(.opacity)

Divider().background(Color.white.opacity(0.1))
                    }

                    // Scrollable Playlist
                    LocalPlaylistView(app: app)
                }
                .transition(.move(edge: .leading))

            } else {
                // ONLINE: Auto-switch between Lobby and Active Game
                if app.ogsClient.activeGameID != nil && app.ogsClient.isConnected {
                    ActiveGamePanel(appModel: app)
                        .transition(.opacity)
                } else {
                    // v3.108 FIX: Pass binding for creation sheet
                    OGSBrowserView(client: app.ogsClient,
```

```
isPresentingCreate: $app.isCreatingChallenge)
                            .transition(.opacity)
                    }
                }
            }
        }
        .frostedGlassStyle()
        .animation(.easeInOut(duration: 0.2), value: selectedTab)
        .animation(.easeInOut(duration: 0.2), value: app.ogsClient.activeGameID)
        .animation(.easeInOut(duration: 0.2), value: app.selection?.id)
    }
}

// ... [Keep LocalGameMetadataView, LocalPlaylistView, LocalGameRow unchanged] ...
// (Omitted for brevity as they are identical to v3.107)

// MARK: - Local Metadata View
struct LocalGameMetadataView: View {
let game: SGFGameWrapper
@ObservedObject var boardVM: BoardViewModel

code
Code

var body: some View {
    VStack(spacing: 12) {
        HStack(alignment: .top) {
            VStack(alignment: .leading, spacing: 4) {
                Label(game.game.info.playerBlack ?? "Black", systemImage: "circle.fill")
                    .font(.headline)
                    .foregroundColor(.white)
                    .imageScale(.small)
                Text("\(boardVM.whiteCapturedCount) prisoners")
```

```
                        .font(.caption)
                        .foregroundColor(.white.opacity(0.6))
                }
                Spacer()
                VStack(alignment: .trailing, spacing: 4) {
                    Label(game.game.info.playerWhite ??
"White", systemImage: "circle")
                        .font(.headline)
                        .foregroundColor(.white)
                        .imageScale(.small)
                    Text("\(boardVM.blackCapturedCount)
prisoners")
                        .font(.caption)
                        .foregroundColor(.white.opacity(0.6))
                }
            }
            Divider().background(Color.white.opacity(0.1))
            HStack {
                if let result = game.game.info.result, !
result.isEmpty {
                    
Text(result).font(.callout.bold()).foregroundColor(.yellow)
                } else {
                    Text("In
Progress").font(.caption).foregroundColor(.white.opacity(0.
5))
                }
                Spacer()
                if let date = game.game.info.date {
                    
Text(date).font(.caption).foregroundColor(.white.opacity(0.
5))
                }
            }
        }
        .padding()
        .background(Color.black.opacity(0.1))
    }
}

struct LocalPlaylistView: View {
```

```
@ObservedObject var app: AppModel
var body: some View {
VStack(spacing: 0) {
if app.games.isEmpty {
Spacer()
VStack(spacing: 12) {
Image(systemName:
"folder.badge.questionmark").font(.largeTitle).foregroundColor(.white.opacit
y(0.3))
Text("No SGF files loaded").foregroundColor(.secondary)
Button("Open Folder...") { app.promptForFolder() }
.buttonStyle(.borderedProminent).tint(.white.opacity(0.1))
}
Spacer()
} else {
ScrollView {
LazyVStack(alignment: .leading, spacing: 0) {
ForEach(app.games) { game in
Button(action: { app.selectGame(game) }) {
LocalGameRow(game: game, isSelected: app.selection?.id == game.id)
}
.buttonStyle(.plain)
Divider().background(Color.white.opacity(0.05))
}
}
}
}
}
}

struct LocalGameRow: View {
let game: SGFGameWrapper
let isSelected: Bool
var body: some View {
VStack(alignment: .leading, spacing: 2) {
HStack {
if isSelected { Image(systemName:
```

```
"play.fill").font(.caption2).foregroundColor(.cyan) }
Text(game.game.info.playerBlack ?? "?").fontWeight(.bold) + Text(" vs ") +
Text(game.game.info.playerWhite ?? "?").fontWeight(.bold)
Spacer()
}
.font(.caption).foregroundColor(isSelected ? .cyan : .white)
HStack {
Text(game.game.info.date ?? "Unknown
Date").font(.caption2).foregroundColor(.gray)
Spacer()
if let result = game.game.info.result
{ Text(result).font(.caption).bold().foregroundColor(.yellow) }
}
}
.padding(.vertical, 6).padding(.horizontal, 8).contentShape(Rectangle())
.background(isSelected ? Color.white.opacity(0.15) : Color.clear)
}
}


//
//  ContentView.swift
//  SGFPlayerClean
//
//  v3.75: Root Container (Fixes Argument Labels)
//  - Removed redundant 'Group' that was causing type inference issues.
//  - Wraps the 2D/3D views.
//  - Manages the DebugDashboard presentation.
//  - Global Keyboard Shortcuts (Cmd+D for Debug).
//

import SwiftUI
import Combine

struct ContentView: View {
@StateObject var appModel = AppModel()

code
```

Code

```
var body: some View {
    ZStack {
        // MAIN CONTENT
        // We removed the 'Group' here to avoid compiler ambiguity with TabContentBuilder
        if appModel.viewMode == .view3D {
            ContentView3D(app: appModel)
        } else {
            // FIX: Use 'app:' label here as well
            ContentView2D(app: appModel)
        }

        // PRE-GAME OVERLAY (If applicable)
        if appModel.showPreGameOverlay {
            Color.black.opacity(0.4).ignoresSafeArea()
            // You can insert a specific OverlayView here if you have one
        }
    }
    // GLOBAL MODIFIERS
    .environmentObject(appModel)
    .preferredColorScheme(.dark)

    // DEBUG DASHBOARD SHEET
    .sheet(isPresented: $appModel.showDebugDashboard) {
        DebugDashboard(appModel: appModel)
            .frame(minWidth: 700, minHeight: 500)
    }

    // KEYBOARD SHORTCUTS
    .background(
        Button("Toggle Debug") {
            appModel.showDebugDashboard.toggle()
        }
        .keyboardShortcut("d", modifiers: .command)
        .opacity(0) // Hidden button to capture the shortcut
    )
}
```

```
}

--

//
//  OGSBrowserView.swift
//  SGFPlayerClean
//
//  v3.123: Revert to Sheet.
//  - Restored .sheet presentation (Stable).
//  - Preserves Green/Yellow button logic.
//  - Preserves Persistence.
//

import SwiftUI

enum BoardSizeCategory: String, CaseIterable, Identifiable {
case size19 = "19", size13 = "13", size9 = "9", other = "Other"
var id: String { rawValue }
}

enum GameSpeedFilter: String, CaseIterable, Identifiable {
case all = "All Speeds", live = "Live", blitz = "Blitz", correspondence =
"Correspondence"
var id: String { rawValue }
}

struct OGSBrowserView: View {
@ObservedObject var client: OGSClient
@Binding var isPresentingCreate: Bool

code
Code

// Persistence via AppStorage
@AppStorage("ogs_filter_speed") private var selectedSpeed:
GameSpeedFilter = .all
@AppStorage("ogs_filter_ranked") private var
```

```
showRankedOnly: Bool = false
@AppStorage("ogs_filter_sizes") private var sizeFiltersRaw:
String = "19,13,9,Other"

// Helper to read filters
private var sizeFilters: Set<BoardSizeCategory> {
    Set(sizeFiltersRaw.split(separator: ",").compactMap
{ BoardSizeCategory(rawValue: String($0)) })
}

var body: some View {
    VStack(spacing: 0) {
        // Header
        VStack(spacing: 12) {
            HStack {
Text("Lobby").font(.headline).foregroundColor(.white)
                Button(action: { isPresentingCreate =
true }) {
                    Image(systemName:
"plus").font(.system(size: 14, weight: .bold)).frame(width:
24, height: 24)
                }
                .buttonStyle(.bordered).tint(.blue).disable
d(!client.isConnected)
                Spacer()
                if client.isConnected { Label("Connected",
systemImage:
"circle.fill").font(.caption).foregroundColor(.green) }
                else { Button("Retry")
{ client.connect() }.font(.caption).buttonStyle(.borderedPr
ominent).tint(.orange) }
            }
            Divider().background(Color.white.opacity(0.1))
            // Filters
            HStack(spacing: 12) {
Text("Size:").font(.caption).foregroundColor(.white.opacity
```

```
(0.7))
                ForEach(BoardSizeCategory.allCases) { size
in
                    Toggle(size.rawValue, isOn: Binding(
                        get:
{ sizeFilters.contains(size) },
                        set: { isActive in
                            var current = sizeFilters
                            if isActive
{ current.insert(size) } else { current.remove(size) }
                            sizeFiltersRaw = current.map
{ $0.rawValue }.sorted().joined(separator: ",")
                        }
                    )).toggleStyle(.checkbox).font(.caption
)
                }
                Spacer()
            }
            HStack {
                Picker("", selection: $selectedSpeed) {
                    ForEach(GameSpeedFilter.allCases)
{ speed in Text(speed.rawValue).tag(speed) }
                }.labelsHidden().frame(width:
120).controlSize(.small)
                Toggle("Rated Only", isOn:
$showRankedOnly).toggleStyle(.checkbox).font(.caption)
                Spacer()
            }
        }
        .padding()
        .background(Color.black.opacity(0.1))

        Divider().background(Color.white.opacity(0.1))

        // List
        if client.availableGames.isEmpty {
            VStack { Spacer(); Text(client.isConnected ?
"No challenges found." :
"Connecting...").foregroundColor(.white.opacity(0.5));
Spacer() }
        } else {
```

```
            ScrollViewReader { proxy in
                List(filteredChallenges) { challenge in
                    let isMine = (client.playerID != nil && challenge.challenger.id == client.playerID!)
                    ChallengeRow(challenge: challenge, isMine: isMine, onAction: {
                        if isMine { client.cancelChallenge(challengeID: challenge.id) }
                        else { client.joinGame(gameID: challenge.id) }
                    })
                    .id(challenge.id)
                }
                .listStyle(.inset)
                .scrollContentBackground(.hidden)
                .contentMargins(.top, 8, for: .scrollContent)
            }
        }

        // Footer
        HStack {
            Text("\(filteredChallenges.count) / \(client.availableGames.count)").font(.caption).foregroundColor(.white.opacity(0.3))
            Spacer()
        }.padding(6).background(Color.black.opacity(0.1))
    }
    .onAppear { if !client.isSubscribedToSeekgraph { client.subscribeToSeekgraph() } }
    // Reverted to .sheet for stability
    .sheet(isPresented: $isPresentingCreate) {
        OGSCreateChallengeView(client: client, isPresented: $isPresentingCreate)
    }
}

var filteredChallenges: [OGSChallenge] {
    client.availableGames.filter { game in
        if showRankedOnly && !game.game.ranked { return false }
```

```
        let w = game.game.width, h = game.game.height
        let cat: BoardSizeCategory = (w==19 &&
h==19) ? .size19 : (w==13 && h==13) ? .size13 : (w==9 &&
h==9) ? .size9 : .other
        if !sizeFilters.contains(cat) { return false }
        let speed = game.speedCategory.lowercased()
        switch selectedSpeed {
        case .all: break
        case .live: if speed != "live" && speed != "rapid"
{ return false }
        case .blitz: if speed != "blitz" { return false }
        case .correspondence: if speed != "correspondence"
{ return false }
        }
        return true
    }.sorted { ($0.challenger.username ==
client.username) != ($1.challenger.username ==
client.username) ? ($0.challenger.username ==
client.username) : ($0.id > $1.id) }
}
}
```

```
struct ChallengeRow: View {
let challenge: OGSChallenge
let isMine: Bool
let onAction: () -> Void
```

```
code
Code
```

```
var body: some View {
    HStack(alignment: .center, spacing: 14) {
        Button(action: onAction) {
            Text(isMine ? "CANCEL" : "ACCEPT")
                .font(.system(size: 10, weight: .bold))
                .padding(.horizontal, 2)
                .foregroundColor(isMine ? .black : .white)
        }
        .buttonStyle(.borderedProminent)
        .tint(isMine ? .yellow : .green)
```

```
            .controlSize(.small)
        }
        VStack(alignment: .leading, spacing: 3) {
            HStack(spacing: 6) {
                Text(challenge.challenger.displayRank).font(.system(size:
13, weight: .bold)).foregroundColor(rankColor)
                Text(challenge.challenger.username).font(.system(size: 13,
weight: .medium)).lineLimit(1)
            }
            Text(challenge.timeControlDisplay).font(.system(size:
11)).foregroundColor(.white.opacity(0.6))
        }.frame(minWidth: 100, alignment: .leading)
        Spacer()
        VStack(alignment: .leading, spacing: 3) {
            Text(challenge.boardSize).font(.system(size:
13, weight: .bold))
            Text(challenge.game.rules.capitalized).font(.system(size:
11)).foregroundColor(.white.opacity(0.6))
        }.frame(width: 60, alignment: .leading)
        Group {
            if challenge.game.ranked {
                Text("RATED").font(.system(size: 10,
weight: .heavy)).foregroundColor(.white.opacity(0.8))
                    .padding(.horizontal,
6).padding(.vertical, 2)
                    .overlay(RoundedRectangle(cornerRadius:
4).stroke(Color.white.opacity(0.3), lineWidth: 1))
            } else {
                Text("UNRATED").font(.system(size: 9,
weight: .semibold)).foregroundColor(.white.opacity(0.3)).op
acity(0)
            }
        }.frame(width: 50, alignment: .trailing)
    }
    .padding(.vertical, 6)
    .listRowBackground(Color.clear)
    .listRowSeparatorTint(Color.white.opacity(0.1))
```

```
}

var rankColor: Color {
    guard let rank = challenge.challenger.ranking else
{ return .gray }
    if rank >= 30 { return .cyan }
    if rank >= 20 { return .green }
    return .orange
}
}


//
//  OGSCreateChallengeView.swift
//  SGFPlayerClean
//
//  v3.125: Final Opacity Fix (Matching SettingsPanel).
//  - Removed standard sheet background to prevent "double layering".
//  - Replaced NavigationStack with a manual V-Stack header.
//  - Applied .frostedGlassStyle() manually to match the exact aesthetic of
SettingsPanel.
//

import SwiftUI

struct OGSCreateChallengeView: View {
@ObservedObject var client: OGSClient
@Binding var isPresented: Bool

code
Code

// Load persisted settings or defaults
@State private var setup = ChallengeSetup.load()
@State private var isSending = false
@State private var errorMessage: String?

var body: some View {
    // 1. Main Container (Imitating SettingsPanel
```

```
structure)
    VStack(spacing: 0) {

        // Header
        HStack {
            Button(action: { isPresented = false }) {
                Text("Cancel")
                    .foregroundColor(.white.opacity(0.8))
            }
            .buttonStyle(.plain)

            Spacer()

            Text("New Challenge")
                .font(.headline)
                .foregroundColor(.white)

            Spacer()

            Button(action: submitChallenge) {
                if isSending {
                    ProgressView().controlSize(.small)
                } else {
                    Text("Create")
                        .fontWeight(.bold)
                        .foregroundColor(.white)
                }
            }
            .buttonStyle(.borderedProminent)
            .tint(.green)
            .disabled(isSending)
        }
        .padding()
        .background(Color.black.opacity(0.1)) // Header
tint

        Divider().background(Color.white.opacity(0.2))

        // Scrollable Form
        ScrollView {
            VStack(alignment: .leading, spacing: 0) {
```

```
                // SECTION 1: Game Info
                SectionHeader("Game Info")
                VStack(spacing: 12) {
                    HStack { Text("Name"); Spacer();
TextField("Name", text:
$setup.name).textFieldStyle(.plain).multilineTextAlignment(
.trailing).frame(width: 150) }
Divider().background(Color.white.opacity(0.1))
                    Toggle("Ranked", isOn: $setup.ranked)
Divider().background(Color.white.opacity(0.1))
                    HStack { Text("Color"); Spacer();
Picker("", selection: $setup.color)
{ Text("Auto").tag("automatic");
Text("Black").tag("black");
Text("White").tag("white") }.labelsHidden().fixedSize() }
Divider().background(Color.white.opacity(0.1))
                    HStack { Text("Handicap"); Spacer();
Picker("", selection: $setup.handicap)
{ Text("None").tag(0); ForEach(2...9, id: \.self) { i in
Text("\(i)").tag(i) } }.labelsHidden().fixedSize() }
                }
                .padding()

Divider().background(Color.white.opacity(0.15))

                // SECTION 2: Board
                SectionHeader("Board")
                VStack(spacing: 12) {
                    Picker("", selection: $setup.size)
{ Text("19x19").tag(19); Text("13x13").tag(13);
Text("9x9").tag(9) }.pickerStyle(.segmented)
Divider().background(Color.white.opacity(0.1))
                    HStack { Text("Rules"); Spacer();
Picker("", selection: $setup.rules)
{ Text("Japanese").tag("japanese");
```

```
Text("Chinese").tag("chinese");
Text("AGA").tag("aga") }.labelsHidden().fixedSize() }
                    }
                    .padding()

Divider().background(Color.white.opacity(0.15))

                    // SECTION 3: Time Control
                    SectionHeader("Time Control")
                    VStack(spacing: 12) {
                        Picker("", selection:
$setup.timeControl) { Text("Byoyomi").tag("byoyomi");
Text("Fischer").tag("fischer");
Text("Simple").tag("simple") }.pickerStyle(.segmented)

Divider().background(Color.white.opacity(0.1))
                        if setup.timeControl == "byoyomi" {
                            TimeFieldRow(label: "Main Time",
value: $setup.mainTime)
                            TimeFieldRow(label: "Period Time",
value: $setup.periodTime)
                            HStack { Text("Periods"); Spacer();
Stepper("\(setup.periods)", value: $setup.periods, in:
1...10) }
                        } else if setup.timeControl ==
"fischer" {
                            TimeFieldRow(label: "Initial",
value: $setup.initialTime)
                            TimeFieldRow(label: "Increment",
value: $setup.increment)
                            TimeFieldRow(label: "Max", value:
$setup.maxTime)
                        } else {
                            TimeFieldRow(label: "Per Move",
value: $setup.perMove)
                        }
                    }
                    .padding()
```

```
Divider().background(Color.white.opacity(0.15))
                
                // SECTION 4: Rank Range
                SectionHeader("Rank Range")
                VStack(spacing: 12) {
                    HStack { Text("Min: \
(formatRank(setup.minRank))"); Spacer(); Stepper("", value: 
$setup.minRank, in: 0...setup.maxRank).onChange(of: 
setup.minRank) { v in if v > setup.maxRank { setup.maxRank 
= v } } }
                    
Divider().background(Color.white.opacity(0.1))
                    HStack { Text("Max: \
(formatRank(setup.maxRank))"); Spacer(); Stepper("", value: 
$setup.maxRank, in: setup.minRank...38).onChange(of: 
setup.maxRank) { v in if v < setup.minRank { setup.minRank 
= v } } }
                }
                .padding()
                
                if let error = errorMessage {
                    
Text(error).foregroundColor(.red).font(.caption).padding()
                }
            }
        }
        .scrollContentBackground(.hidden) // Ensure 
ScrollView is clear
    }
    .frame(minWidth: 400, minHeight: 650)
    // CRITICAL FIX: Match SettingsPanel exactly
    .frostedGlassStyle()
    // CRITICAL FIX: Kill the default sheet background so 
our custom glass shows
    .presentationBackground(.clear)
}

private func submitChallenge() {
    setup.save()
    isSending = true; errorMessage = nil
    client.createChallenge(setup: setup) { success, error 
```

```
in
            DispatchQueue.main.async {
                isSending = false
                if success { isPresented = false; if !
client.isSubscribedToSeekgraph
{ client.subscribeToSeekgraph(force: true) } }
                else { errorMessage = error ?? "Unknown
error" }
            }
        }
    }

private func formatRank(_ val: Int) -> String { val < 30 ?
"\(30 - val)k" : "\(val - 29)d" }
}

// Helpers
struct SectionHeader: View {
let text: String
init(_ text: String) { self.text = text }
var body: some View {
Text(text)
.font(.caption).fontWeight(.bold)
.foregroundColor(.white.opacity(0.5))
.frame(maxWidth: .infinity, alignment: .leading)
.padding(.horizontal).padding(.top, 16)
}
}

struct TimeFieldRow: View {
let label: String; @Binding var value: Int
var body: some View { HStack { Text(label); Spacer(); TextField("", value:
$value,
format: .number).textFieldStyle(.plain).multilineTextAlignment(.trailing).fram
e(width:
50).padding(4).background(Color.white.opacity(0.1)).cornerRadius(4);
Text("s").foregroundColor(.gray) } }
}
```

Finally, I'd like to make three points. First, please, there's no need to guess the content of any file. Just let me know and I'd be happy to upload it for you to review. This is a much faster and more efficient process than having you guess what might be in it.

Second, I don't like to receive pieces of files which I have to copy and paste into the appropriate spot. It is much better and less error prone, though granted less efficient in terms of tokens, for you to give me the entire file and have me just drop it in.

As a final point, if there are any protocol issues, or any issues about communication with the OGS server, we can use the working web client and developer tools to see what the actual communications are in a working system and simply imitate what's needed. Again, no need to guess.