**1. Kickoff Script for the Next Session**

"I am continuing the SGFPlayerClean project. We have successfully reached **v8.100**, which builds and renders the 'Tabletop' assembly (Board, Lids, Tatami) with synchronized scaling. The slider is high-performance due to a Render Cache. I have Handover Document v8.500. Please acknowledge the Rules of Engagement. My first priority is to eliminate the **Lid Update Delay** using a 'DidSet' or 'Subject' notification model, then move to OGS REST integration for joining live games."

**2. Rules of Engagement (Strict Enforcement)**

- **Trust the User:** Accept sidebar states and folder paths as absolute truth.
- **No Guessing:** Never assume the content of a file. If a file hasn't been pasted in the current session, ask the user.
- **Full Code Drop-ins Only:** Provide entire file contents so the user can "Select All > Paste."
- **Single Source of Truth:** All Enums, Structs, SGF Parsing, and Coordinate handling must live in `OGSModels.swift`.
- **Zero Redundancy:** If a type is moved to `OGSModels.swift`, any individual files containing that type must be deleted immediately.
- **Spatial Consistency:** Maintain the 1 : 1.0773 1:1.0773 aspect ratio and 6.5 % 6.5%

wood margin.

- **Linear Spatial Rendering:** Always render stones using a fixed-order loop (Row -> Col) or a sorted array to prevent Z-order flickering.

**3. Current Status: The "Lid Delay" Analysis**
While captures function, the bowls lag behind the board updates.

- **Hypothesis 1: The Combine "WillChange" Race:** `BoardViewModel` observes the Engine via `objectWillChange`. This fires *before* properties update. The ViewModel likely reads Move $N-1$ values while the Engine is still busy with Move $N$.
- **Hypothesis 2: Dependency Depth:** SwiftUI occasionally fails to propagate updates from deeply nested @Published objects (Engine inside ViewModel) to subviews (LidView) in a single render pass.
- **Hypothesis 3: Main Thread Congestion:** The `seek` operation is heavy. If the UI tries to render the lids while the Engine is still looping through captures, the lid render is deferred to the next frame.

**4. Architectural Map**

1. **The Root Coordinator (`AppModel.swift`)**
    - Owns the lifecycle. Manages the transition between Local and

Online modes.

2. **The Engine (`SGFPlayerEngine.swift`)**
    - **Logic:** Pure Go rules. Handles "Pass" moves as `nil` coordinates.
    - **Variables:** `whiteStonesCaptured` and `blackStonesCaptured` (Physical counts).
3. **The Bridge (`BoardViewModel.swift`)**
    - **Orchestration:** Calculates the **Render Cache** (`stonesToRender`).
    - **Safety:** Blocks stone placement in Local mode.
4. **The Truth (`OGSModels.swift`)**
    - Central vault for shared types and the SGF Parser.
5. **The Network (`OGSClient.swift`)**
    - WebSocket management and "Golden DNA" move payloads.

**5. Verified OGS Protocols**

- **Endpoint:** `wss://wsp.online-go.com/` (No `/socket.io/` path).
- **Heartbeat:** `["net/ping", {"client": <ms_timestamp>}]` every 5 seconds.
- **Write-Access Payload:** Requires `game_id`, `move`, `auth` (from REST), and `player_id`.
- **Move Encoding:** Standard SGF (e.g., "pd").

**6. Critical Variable Mapping (The "Truth Chain")**

To prevent "Sign Errors" (flipped colors), we use the **Loot Model**:

| Entity | Variable Name | Physical Meaning |
|---|---|---|
| **Engine** | `whiteStonesCaptured` | White stones physically pulled off the board. |
| **ViewModel** | `blackCapturedCount` | Black player's loot (Count of White stones he holds). |

| | | |
|---|---|---|
| **View (Bowl)** | `stoneColor: .white` | The visual stones appearing in Black's (bottom) bowl. |
| **Metadata** | `prisoners` | Displayed next to Black's name. |

**7. Build Stability & Compiler Safety**

- **The `buildExpression` Error:** SwiftUI's compiler times out if a View body contains too much imperative math.
    - **Fix:** Perform all layout math in private helper functions (e.g., `getLayout()`) or local constants (`let`) at the top of the `body`.
    - **Fix:** Avoid placing logic calls (e.g. `boardVM.step()`) inside Button labels; use `Image` or `Text` only.
- **Contextual Base Errors:** Modifiers like `.padding(.bottom)` can fail if the compiler loses track of the View type. Use `EdgeInsets` for complex layouts.

**8. Performance Persistence (The Render Cache)**

The **Render Cache** (`stonesToRender`) is mandatory.

- **Why:** A standard Go board has 361 intersections. A nested loop checking every intersection for jitter/color during a slider drag locks the Main Thread.
- **Solution:** The ViewModel must flatten the 19x19 grid into a simple `Identifiable` array of stones that actually exist. The View then uses a high-speed `ForEach` on that small array.

**9. Known Risks & Sensitivities**

- **Tatami Scaling:** The texture must scale using `ImagePaint` or a scaled frame based on a **750pt reference height**. Otherwise, the straws will appear to grow/shrink at a different rate than the stones.
- **Z-Stack Layering:** The `TatamiBackground` must remain the absolute first element in the root `ZStack` to allow the sidebar to correctly render its "Frosted Glass" transparency.
- **Coordinate Drift:** Lid positions must be calculated as a function

of `boardWidth` + `cellWidth` to ensure they stay "pinned" to the board edge during resizing.

## 10. Near-Term Agenda

1. **Sync Fix:** Move the prisoner update signal to an `ObservableObject` publisher that fires *after* the board is set.
2. **Lid variety:** Ensure `StoneView2D` continues to pick from the 5 clam assets deterministically.
3. **OGS Bridge:** Implement the REST call to `/api/v1/games/{id}` to fetch the `auth` token required for move authorization.