**Projektplan**

**Phase 1: Stabilisieren & Definieren (Grundlage schaffen)**

1. **Gegenlesen/Prüfen/Inline-Doku/Logs/TODOs:**
   * **Warum zuerst?** Bevor du neue Funktionalität hinzufügst oder Tests schreibst, sollte der *aktuelle* Code so sauber und verständlich wie möglich sein. Das Beheben kleinerer TODOs, das Übersetzen von Logs und das Hinzufügen von Docstrings (RestructuredText ist super!) schafft eine solide Basis und vermeidet, dass du Fehler oder Unklarheiten in neue Bereiche "mitziehst".
   * **Fokus:** Gehe jede Klasse und Methode durch. Sind die Namen klar? Funktionieren die Abläufe logisch? Sind die deutschen Log-Einträge präzise? Sind alle deine Kommentare noch aktuell? Füge Docstrings hinzu, die erklären, *was* eine Funktion/Klasse tut, welche Parameter sie erwartet und was sie zurückgibt.
2. **Nachrichten / Kommunikation zw. Server und Client definieren:**
   * **Warum jetzt?** Du kannst die Spiellogik in der GameEngine und den Test-Client (oder später die WebApp) nicht sinnvoll implementieren, wenn du nicht genau weißt, *welche* Daten in *welchem Format* (JSON-Struktur) ausgetauscht werden.
   * **Fokus:** Lege fest, wie z.B. eine "Spielkarte spielen"-Aktion vom Client aussieht ({'action': 'play\_cards', 'payload': {'cards': ['H10', 'H1']}}?), wie ein "Spielzustand-Update" vom Server aufgebaut ist ({'type': 'public\_state\_update', 'payload': {...PublicState...}}), wie Fehler gemeldet werden etc. Dokumentiere diese Strukturen.

**Phase 2: Kernlogik & Basis-Tests**

1. **Abgleich mit deprecated Code / Implementierung PublicState/PrivateState/GameEngine:**
   * **Warum jetzt?** Mit den definierten Nachrichtenstrukturen kannst du nun die Platzhalter in der GameEngine mit Leben füllen. Implementiere die Kernmechaniken: Karten austeilen, handle\_player\_message für die definierten Aktionen (Karten spielen, passen, Tichu ansagen?), Runden-/Stichlogik, Punkte zählen (zumindest grundlegend). Fülle PublicState und PrivateState mit den notwendigen Attributen.
   * **Fokus:** Ziel ist ein minimal funktionsfähiger Spielablauf innerhalb der Engine, der auf die definierten Nachrichten reagiert.
2. **Test-Client schreiben:**
   * **Warum jetzt?** Sobald die Engine auf grundlegende Nachrichten reagieren kann, ist ein Test-Client *Gold wert*. Du kannst damit viel schneller und gezielter testen als mit einer echten WebApp oder nur mit Unit-Tests. Du kannst damit das Verbinden, Trennen, Senden von Spielzügen etc. automatisieren und die Serverantworten prüfen.
   * **Fokus:** Ein einfaches Python-Skript (mit aiohttp als Client) das eine WebSocket-Verbindung aufbaut, sich authentifiziert (Parameter sendet), die definierten Nachrichten senden und empfangene Nachrichten ausgeben oder rudimentär prüfen kann.

**Phase 3: Umfassende Tests & Dokumentation**

1. **Unit-Tests schreiben:**
   * **Warum jetzt?** Mit dem Test-Client kannst du die Gesamt-Flows testen. Jetzt ist ein guter Zeitpunkt, detailliertere Unit-Tests (z.B. mit pytest) für spezifische, kritische oder komplexe Teile zu schreiben:
     + Einzelne Funktionen der GameEngine (z.B. Validierung von Kartenkombinationen).
     + Die Timer-Logik und Randfälle in der GameFactory.
     + Die Zustandsübergänge in PublicState/PrivateState.
     + **Deine genannten Fälle sind sehr gut!** Ergänzend:
       - Fall 7b: Connect, Tisch beitreten (wo schon 1 Mensch, 2 KIs sitzen).
       - Fall 8: Ungültige Aktion senden (z.B. Karten spielen, die man nicht hat; Aktion senden, wenn man nicht dran ist).
       - Fall 9: Schnelles Verbinden/Trennen/Wiederverbinden (Stresstest für Timer/Reconnect).
       - Fall 10: Fehler bei der JSON-Serialisierung/Deserialisierung.
   * **Fokus:** Teste einzelne Komponenten isoliert und decke Randfälle ab.
2. **Dokumentation für Entwickler schreiben:**
   * **Warum jetzt?** Der Code ist stabilisiert, die Kernlogik implementiert und getestet. Jetzt kannst du die übergeordnete Dokumentation schreiben, die die Architektur, Klassenaufgaben, den Datenfluss und wichtige Designentscheidungen erklärt. Das Glossar ist hier auch sehr sinnvoll.
   * **Fokus:** Eine README.md oder separate Dokumentationsdateien, die neuen Entwicklern (oder deinem zukünftigen Ich) den Einstieg erleichtern.

**Phase 4: Frontend**

1. **WebApp für den Spieler schreiben:**
   * **Warum zuletzt?** Die WebApp benötigt ein funktionierendes und stabiles Backend mit einer klar definierten API (den Nachrichten). Jetzt kannst du dich auf die Client-Seite konzentrieren. Die Portierung von Godot nach HTML/JS/CSS ist ein eigenes Projekt.
   * **Fokus:** UI-Design, Client-seitige Logik, Kommunikation mit dem WebSocket-Server basierend auf den definierten Nachrichten.

**Zusammenfassend die empfohlene Reihenfolge:**

1. Code prüfen/dokumentieren/übersetzen/TODOs bearbeiten
2. Nachrichten/Kommunikation definieren
3. Kernlogik GameEngine/States implementieren
4. Test-Client schreiben (und parallel zu 3 nutzen)
5. Unit-Tests schreiben (detaillierte Fälle)
6. Entwickler-Dokumentation schreiben
7. WebApp schreiben