Horropoly Multiplayer Sync Fix – Full Implementation Spec

Problem

Clients currently write directly into `/gameRooms/`, which causes: - Race conditions (two clients overwrite the same state). - Corruption of arrays (players, properties). - Slow & inconsistent multiplayer sync.

New Firestore Structure

```
/rooms/<roomId>              // Pre-game lobby (unchanged)
/gameRooms/<roomId>/state    // Authoritative game state (backend only)
/gameRooms/<roomId>/inbox    // Client intents (players push moves here)
/gameRooms/<roomId>/log      // Server-applied moves (append-only history)
```

Flow

1. Client pushes an intent to `/gameRooms//inbox` 2. Backend (Render) listens to `/inbox`, validates intent, and runs a pure reducer against current `/state`. 3. Backend assigns actionId, version, and integrity hash. 4. Backend writes new snapshot to `/state` and appends move to `/log`. 5. Clients only subscribe to `/state` for authoritative updates.

Firestore Security Rules

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /gameRooms/{roomId} {
      match /state {
        allow read: if request.auth != null;
        allow write: if false;
      }
      match /inbox/{intentId} {
        allow create: if request.auth != null;
        allow update, delete: if false;
        allow read: if request.auth != null;
      }
      match /log/{actionId} {
        allow read: if request.auth != null;
        allow write: if false;
      }
    }
    match /rooms/{roomId} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Backend Reducer (Express + Firestore Skeleton)

```
import express from "express";
import { initializeApp, cert } from "firebase-admin/app";
import { getFirestore } from "firebase-admin/firestore";
import crypto from "crypto";

const app = express();
app.use(express.json());
initializeApp({ credential: cert(JSON.parse(process.env.FIREBASE_SERVICE_ACCOUNT)) });
```

```
const db = getFirestore();

function hashState(state) {
  return crypto.createHash("sha256").update(JSON.stringify(state)).digest("hex");
}

function reduce(prevState, intent, actionId) {
  const next = structuredClone(prevState);
  switch (intent.type) {
    case "ROLL":
      const steps = intent.payload.steps || 0;
      const player = next.players.find(p => p.userId === intent.playerId);
      if (player) {
        player.position = (player.position + steps) % 40;
        next.lastDiceRoll = steps;
      }
      break;
    case "END_TURN":
      next.currentTurn = (next.currentTurn + 1) % next.players.length;
      break;
  }
  next.version = (prevState.version || 0) + 1;
  next.lastAppliedId = actionId;
  next.hash = hashState(next);
  next.lastUpdated = new Date().toISOString();
  return next;
}

app.post("/games/:roomId/intent", async (req, res) => {
  const { roomId } = req.params;
  const intent = req.body;

  const roomRef = db.collection("gameRooms").doc(roomId);
  const stateRef = roomRef.collection("state").doc("snapshot");
  const logRef = roomRef.collection("log");

  await db.runTransaction(async tx => {
    const snap = await tx.get(stateRef);
    const prevState = snap.exists ? snap.data() : { players: [], version: 0 };
    const actionId = (prevState.lastAppliedId || 0) + 1;
    const nextState = reduce(prevState, intent, actionId);

    tx.set(stateRef, nextState);
    tx.set(logRef.doc(String(actionId)), {
      actionId,
      intent,
      prevHash: prevState.hash || null,
      nextHash: nextState.hash,
      timestamp: new Date().toISOString()
    });
  });

  res.json({ ok: true });
});

app.listen(8080, () => console.log("Server running on port 8080"));
```

Client-Side Example

```
import { doc, onSnapshot } from "firebase/firestore";

// Send intent to backend
export async function sendIntent(roomId, intent) {
  await fetch(`/games/${roomId}/intent`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(intent)
  });
}

// Subscribe to authoritative state
export function subscribeToState(db, roomId, callback) {
  const stateRef = doc(db, "gameRooms", roomId, "state", "snapshot");
  return onSnapshot(stateRef, (docSnap) => {
    if (docSnap.exists()) {
      callback(docSnap.data());
    }
  });
}

// Example usage
sendIntent("A1B2C3", { playerId: "user_abc123def", type: "ROLL", payload: { steps: 6 } });
subscribeToState(db, "A1B2C3", (state) => {
  console.log("Authoritative state:", state);
});
```

What Codex Must Do

1. Implement backend reducer loop in Render backend (like above). 2. Enforce Firestore security rules so only backend writes /state and /log. 3. Modify client code: stop direct writes to /gameRooms/, instead POST intents to Render backend. 4. Clients subscribe to /state for real-time updates, reconcile after optimistic updates. 5. Add optimistic UI but always reconcile with /state.