DAPHNIS LABS

605, D Mall, Netaji Subhash Place, Pitampura DELHI-110034. Email: Hello@DaphnisLabs.com

Daphnis Labs — Full-Stack Developer Intern

Take-Home Assignment: Plinko Lab (Provably-Fair)

Timebox: Aim to complete core requirements within **~8 hours** of focused work. We'll allow **24–36 hours** wall-clock for submission.
AI usage: You may (and are encouraged to) use AI. **Document exactly where/how you used it** in the README.

What you'll build

An interactive **Plinko** game with:

1. A **provably-fair** commit-reveal RNG protocol,
2. A **deterministic, seed-replayable** outcome engine,
3. A polished **UI/UX** (animations, sound, responsive), and
4. A small **API + DB** to log rounds and expose a **verifier** page.

This is an engineering exercise. No real money. We're evaluating correctness, problemsolving, and craftsmanship.

Functional Requirements

A) Game UX

- **Board:** 12 rows, triangular peg layout, 13 bins at the bottom.
- **Controls:** choose **drop column** (0–12), set a **bet amount** (can be just a number; no wallet needed), **Drop** button.
- **Animations:** smooth ball movement with peg collisions; bin pulse + confetti on landing.
- **Sound:** subtle peg tick; celebratory SFX on landing. Provide a **mute** toggle.
- **Accessibility:** keyboard controls for left/right drop selection, space to drop; **Reduced motion** mode honoring `prefers-reduced-motion`.
- **Responsive:** mobile and desktop.

B) Provably-Fair Protocol

Use a standard commit-reveal with a client contribution:

- Server chooses a random **serverSeed** and a **nonce** per round.

- Server publishes only the **commit** before the round starts:
  `commitHex = SHA256(serverSeed + ":" + nonce)`

- Client provides a free-form **clientSeed** when starting the round.

- After the round ends, server **reveals** `serverSeed`.

- The **combinedSeed** used to drive all randomness is:
  `combinedSeed = SHA256(serverSeed + ":" + clientSeed + ":" + nonce)`

- All randomness in the round must come from a **deterministic PRNG** seeded by `combinedSeed` (see "Deterministic Engine").

- Provide a **public Verifier** page that recomputes the outcome from inputs `serverSeed, clientSeed, nonce, dropColumn` and matches the logged round.

C) Deterministic Engine (MVP spec)

To keep this feasible in ~8 hours, use a **discrete Plinko model** that is 100% deterministic and replayable:

- **Rows (R)** = 12. At each row the ball makes a **Left** or **Right** decision.

- Maintain a counter **pos** (number of Right moves so far), `pos ∈ [0..R]`.

- Final **binIndex = pos** (0..12).

- Generate a **peg map**: for each row `r` (0-based), create `r+1` pegs, each with a **leftBias ∈ [0.4, 0.6]`.
  `Example formula per peg using PRNG` rand()`in [0,1):` leftBias = 0.5 + (rand() - 0.5) * 0.2` → round to ~6 decimals for stable hashing.

- Compute `pegMapHash = SHA256(JSON.stringify(pegMap))` and store/log it.

- **Drop column influence:** player picks `dropColumn ∈ [0..12]`. Convert to a small bias adjustment:
  `adj = (dropColumn - floor(R/2)) * 0.01` and `bias' = clamp(leftBias + adj, 0, 1)`.

- **Decision per row:** at row `r`, use the peg at index `min(pos, r)` (peg under current path). Draw `rnd = rand()`. If `rnd < bias'` choose **Left**, else **Right** (then `pos += 1`).

- Use the **same PRNG stream** order every time: first for **peg map generation**, then for **row decisions**. This guarantees a verifier can reproduce results exactly.

- The front-end animation should visually follow the deterministic path. You may "fake" continuous physics as long as the landing bin always matches the deterministic path.

Stretch (optional): Implement true fixed-timestep physics (e.g., Matter.js) but keep the **discrete decisions authoritative** for fairness/replay.

D) Payouts (simple)

- Provide a symmetric example paytable for bins 0..12 (display it in the UI). Implementation can be fixed constants (e.g., edges higher multiplier).

- Record the **payoutMultiplier** used. (Payout math is **not** part of fairness proof.)

E) Verifier Page

- A public page `/verify` with a form for `serverSeed, clientSeed, nonce, dropColumn`.

- On submit, recompute `commitHex, combinedSeed, pegMapHash, final binIndex`.
- Show a ✅/❌ against the stored round by ID, and render a simple **replay** of the path.

Non-Functional Requirements

- **Performance:** 60fps on a reasonable laptop; avoid layout thrash.
- **Quality:** at least a few unit tests for RNG/combiner/engine.
- **Security:** no secrets in client; validate inputs server-side.
- **DX:** clear scripts: `dev`, `build`, `start`, `test`.

Tech Stack (preferred)

- **Frontend:** Next.js 14+ (App Router), React, TypeScript. Canvas/WebGL (your choice).
- **Backend:** Node.js with Express **or** Next.js API routes.
- **DB:** Postgres + Prisma (SQLite acceptable if you keep it consistent).
- **Hash/PRNG:** SHA-256 from a standard lib; PRNG can be **xorshift32**, **mulberry32**, or similar. Document exactly what you used.

If you swap technologies, ensure all requirements and the verifier still work.

API & Data

Minimal Data Model (Prisma-style pseudocode)

```
model Round {
  id            String   @id @default(cuid())
  createdAt     DateTime @default(now())
  status        String   // CREATED | STARTED | REVEALED

  // Fairness
  nonce         String
  commitHex     String   // SHA256(serverSeed:nonce)
  serverSeed    String?  // revealed post-round
  clientSeed    String
  combinedSeed  String   // SHA256(serverSeed:clientSeed:nonce)
  pegMapHash    String

  // Game
  rows          Int      // 12
  dropColumn    Int      // 0..12
  binIndex      Int      // 0..12
  payoutMultiplier Float
  betCents      Int
  pathJson      Json     // decisions per row for replay
  revealedAt    DateTime?
}
```

Suggested Endpoints

- `POST /api/rounds/commit` → `{ roundId, commitHex, nonce }`
  Creates a round internally with a random **serverSeed** and **nonce**; stores `commitHex`.

- `POST /api/rounds/:id/start` body: `{ clientSeed, betCents, dropColumn }` → `{ roundId, pegMapHash, rows }`
  Computes `combinedSeed`, generates peg map + path + `binIndex`, calculates payout; **does not** reveal `serverSeed`.

- `POST /api/rounds/:id/reveal` → `{ serverSeed }`
  Moves to REVEALED, persists `serverSeed`.

- `GET /api/rounds/:id` → full details for UI + verifier.

- `GET /api/verify?serverSeed&clientSeed&nonce&dropColumn` → deterministic recompute; returns `{ commitHex, combinedSeed, pegMapHash, binIndex }`.

- *(Optional)* `GET /api/rounds?limit=20` → recent rounds for a small session log.

Test Vectors (use in your unit tests & README)

Use these to prove your combiner/PRNG match our reference.

Inputs

rows = 12
serverSeed = "b2a5f3f32a4d9c6ee7a8c1d33456677890abcdeffedcba0987654321ffeeddcc"
nonce = "42"
clientSeed = "candidate-hello"

Derived

commitHex = bb9acdc67f3f18f3345236a01f0e5072596657a9005c7d8a22cff061451a6b34
combinedSeed= e1dddf77de27d395ea2be2ed49aa2a59bd6bf12ee8d350c16c008abd406c07e0
PRNG = xorshift32 seeded from first 4 bytes of combinedSeed (big-endian)
First 5 rand() in [0,1): 0.1106166649, 0.7625129214, 0.0439292176, 0.4578678815, 0.3438999297

Peg map (first rows, leftBias rounded to 6dp)

Row 0: `[0.422123]`
Row 1: `[0.552503, 0.408786]`
Row 2: `[0.491574, 0.468780, 0.436540]`
(You don't need to match rounding beyond 6dp, but your `pegMapHash` must be stable for your chosen rounding.)

Path outcome (center drop)

`dropColumn = 6` (center), `adj = 0` → **binIndex = 6**.
(We will check your verifier returns the same bin for the inputs above.)

Easter Eggs (implement any two)

- **TILT mode:** press `T` → board visually rotates ±5° with a vintage arcade filter (visual only).
- **Golden Ball:** if the last three landings were exactly the center bin, next ball uses a golden trail.
- **Secret theme:** typing `open sesame` once toggles a torchlight/dungeon theme for one round.
- **Debug grid:** press `G` to overlay peg positions and show RNG values for the next row.

Deliverables

1. **GitHub repo** (public or invite us). Include clear commit messages.
2. **Live deployment** (Vercel/Render/Fly/etc.).
3. **README** with:
    - **How to run** locally + environment variables.
    - **Architecture** overview (diagram optional).
    - **Fairness spec** you implemented (hash/PRNG details, rounding, peg map rules).
    - **Where/how you used AI** (paste key prompts or summarize; what you kept/changed and why).
    - **Time log** (rough is fine) and what you would do next with more time.
    - Links: live app, verifier page, example round permalink.
4. A couple of **unit tests** (e.g., combiner→PRNG sequence, test vector above, and replay determinism).

Scoring Rubric (100 pts)

- **Provably-fair & Verifier (25):** correct commit-reveal, reproducible outputs, clear verifier UX.
- **Deterministic Engine (20):** peg map + path reproducibility; stable hashes; correct handling of `dropColumn`.
- **Frontend polish (25):** animation smoothness, sound, responsiveness, accessibility, attention to detail.
- **Backend/API/DB (15):** clean schema, validations, logging, tidy code structure.
- **Testing (10):** meaningful tests for RNG/engine; one integration test if possible.
- **Docs & AI usage (5):** clarity, honesty about AI assistance, tradeoffs.

Bonus (+ up to 10): realtime session log, downloadable CSV of round hashes, proper fixed-timestep physics, or an elegant theming system.

Constraints & Notes

- Keep secrets server-side. Do not use external gambling SDKs.

- Use open-source assets or generate your own; include licenses/attributions if needed.
- If you customize the engine (e.g., different bias function), **document it clearly** so the verifier remains unambiguous.

Submission

Reply with:

- GitHub URL
- Live URL
- A 1–2 paragraph note on key decisions and where AI helped you most.

Good luck! We’re excited to see how you approach correctness, UX, and speed under constraints.

For **DAPHNIS LABS**

Candidate signature _____

(This is a computer-generated softcopy; you are required to collect hard copy at the time of joining)

Confidentiality: The matter of your compensation is the confidential information of the company. Any discussion or disclosure of your compensation with anybody other than designated executives will be considered a breach of the agreement by you. Your compensation package is unique to you and not for comparison with other employees of the company.