2. Cursor-Sized In-Depth Tasks (Phase 0 → Phase 1)

Each task is scoped to 5-10 lines of output—small enough for Cursor to stay sharp, big enough to avoid overhead.

#	Cursor prompt (copy-paste)	Purpose
0.1	*(unchanged – repo skeleton bash script)*	Clean repo
0.2	*(unchanged – `anchor init wager --typescript` into `/contracts`)*	Anchor template
0.2-fix	`Generate a bash script named scripts/apply_anchor_fix.sh that:<br>1) rustup toolchain install nightly-2025-04-16;<br>2) rustup override set nightly-2025-04-16 (run in /contracts);<br>3) Appends to /contracts/wager/Cargo.toml:<br>[patch.crates-io]<br> anchor-syn = { git = "https://github.com/solana-foundation/anchor.git", branch = "master", package = "anchor-syn" }<br>4) rm -f Cargo.lock`	Locks nightly + cargo-patch
0.2-cli	`Add a pnpm script "anchor:install" that runs: cargo uninstall anchor-cli	

0.3	*“Inside /web run:* `npx create-next-app@latest chessbets-web --ts --tailwind --eslint --import-alias "@/ *"`*.”*	Web app
0.4	*“Add @solana/wallet-adapter-react, wallet-adapter-wallets, wallet-adapter-react-ui. Wrap* `_app.tsx` *with ConnectionProvider, WalletProvider, WalletModalProvider. Use QuickNode Devnet URL from* `.env`*.”*	Wallet modal
0.5	*“Convert repo to pnpm workspaces. Root package.json should hoist devDeps and add scripts:* `dev:web`*,* `dev:contracts`*,* `test`*.”*	Dev flow
0.6	**Update** CI YAML: before tests run ```yaml - name: Install patched Anchor run:	\n rustup toolchain install nightly-2025-04-16\n rustup default nightly-2025-04-16\n cargo install --git https://github.com/solana-foundation/anchor --branch master anchor-cli --force ```
0.7	*(manual* `solana config` *+ airdrop)*	Devnet wallet

(Phase 0 mirror of Indepth PDF tasks)

Phase 1 — Core Wager MVP

#	Cursor Prompt	Outcome

1.1	*“In /contracts/wager, add PDA structs:* `Match { player_one, player_two, stake_lamports, winner, is_settled, start_slot }`, `PlayerStats { rating_i32, games_u32, wins_u32, is_provisional_bool }`*.”*	Data layer
1.2	*“Implement* `create_match` *instruction: requires two signer PDAs, identical* `stake_lamports`*, checks stake ≤ user cap,* ***does not transfer funds yet****—just emits Challenge event.”*	Match offer w/out escrow
1.3	*“Add* `confirm_match` *instruction (called within 10 s): both players sign; transfers stake into Match PDA escrow.”*	Confirm window logic
1.4	*“Implement* `submit_result` *with enum {Mate, Resign, Timeout, Disconnect}; require 15 s slot guard for timeouts.”*	Final result write
1.5	*“Implement* `settle_match`*: pays 93 % winner, 4 % platform, 1.5 % × 2 to* `royalty_router_pubkey` *list.”*	Payout flow
1.6	*“Generate Anchor Mocha tests for: (a) confirm window timeout cancels; (b) full flow pays correct lamport splits.”*	Regression safety
1.7	*“In /web, add* `useStakeSelector()` *hook with default 0.02 SOL presets, custom input, and 10 s confirm dialog UI.”*	Stake-→-Play-→-Confirm UX
1.8	*“Add* `useMatchmaker()` *hook: announces desired stake over WebSocket, pairs equal stakes, then auto-calls* `confirm_match`*.”*	Hidden lobby
1.9	*“Render Chess.tsx using* `@react-chess/chessboardjs` *with 3 + 2 timer, premove, resign, and WebSocket move relay.”*	Playable board

phase 2 — Opening NFTs & Royalty Pipeline

(Apr 22 – 25, milestone that turns matches into revenue for creators)

All tasks are broken into Cursor-sized chunks (5-10 lines each). Copy a prompt → paste in Cursor → review diff → commit → ping me for the next.
Assumes Phase 1 has shipped and `/contracts`, `/web`, `/scripts` directories already exist.

#	Cursor prompt (copy-paste)	Outcome
2.0 — Opening master list	**“Create `scripts/opening_data.ts` that exports an `Opening[]` array for every ECO code A00-E99. Each item: `{ eco: string; name: string; pgn: string }`. Use hard-coded JSON (no fetch).”**	Canonical source of 3 000 openings
2.1 — Bubblegum tree	**“Write `scripts/create_tree.ts` that (a) uses `@metaplex-foundation/js` to create a Bubblegum Merkle tree sized for 4 096 leaves, canopy = 3; (b) logs the tree public key; (c) saves it to `./scripts/.env.local` as `TREE_ADDRESS=`.”**	Devnet tree ready
2.2 — Metadata & mint	**“Write `scripts/mint_openings.ts`: imports `Opening[]`, iterates, builds compressed metadata `{ name: \`\${eco} – ${name}`, symbol:'OPEN', uri:'https://example.com/opening.png' }`, and mints each leaf to the TREE_ADDRESS with platform treasury as owner. Batch-mint in chunks of 100.”**	3 000 compressed NFTs on Devnet
2.3 — CSV mapping	**“Extend `mint_openings.ts` to append a line to `openings.csv` per mint: `eco,name,assetId`. At end, echo `TotalSOL:` and the lamports spent.”**	`openings.csv` + cost printout
2.4 — Web lookup table	**“Generate `web/lib/openings.json` from `openings.csv` keyed by `eco` → `assetId`.”**	Front-end can map ECO → asset
2.5 — Royalty router constants	**“In `/contracts/wager/src/lib.rs` update `settle_match`: add remaining accounts `white_owner: Signer<'info>`, `black_owner: Signer<'info>`. Compute lamports = (stake * 15) / 1000; use `system_program::transfer` to pay each owner.”**	1.5 % routed on-chain
2.6 — Lamport-split test	**“Add Anchor test `royalty_split.ts`: simulate 0.08 SOL stake, assert winner = 0.0744 SOL,**	Regression guard

	each owner = 0.0012 SOL, platform = 0.0032 SOL (±1 lamport).”	
2.7 — Owner lookup hook	**“In `/web/hooks` add `useOpeningOwner(eco: string)`: (1) read assetId from `openings.json`; (2) call Helius `GET /v0/assets/\${assetId}`; (3) return owner pubkey (cached for 60 s).”**	Client can fetch NFT owners
2.8 — Royalty injection	**“Modify `useMatchmaker` settle flow: after PGN classification, call `useOpeningOwner` for both sides, then pass `whiteOwner` & `blackOwner` pubkeys into the `settleMatch` RPC.”**	End-to-end royalties
2.9 — Cost evidence	*(Manual)* Run `pnpm ts-node scripts/mint_openings.ts`, capture terminal with SOL total; save screenshot to `docs/cost_breakdown.png`.	Proof minting < 0.2 SOL

🔑 What this accomplishes

- **Compressed NFTs live on Devnet** (cost under 0.2 SOL).
- **Smart contract now routes 1.5 % + 1.5 %** to actual NFT owners.
- **Front-end automatically discovers owners** via Helius and injects them into every `settleMatch` call.
- **Unit test locks the math** so splits can’t regress.

After task **2.9** passes CI, Phase 2 is complete and we’re ready to move on to Fair-Play v1 (Phase 3).