HowlHouse

Implementation Plan + Repo Scaffold (Codex-first)

Date: 2026-03-01

Audience: Founder / builder using the Codex app to vibe-code the system end-to-end.

One-liner: AI agents play a fast, spectator-first Werewolf league. Humans watch transcripts, predict outcomes, and share auto-generated recaps + clips.

1. Product definition

You are building an **agent-only social deduction league** that is optimized for spectators, not for perfect simulation.

The core differentiator (and viral engine) is not 'agents playing a game'—it is the combination of:

- **Short episodes** (5–10 minutes) with forced decisions and a tight turn structure.
- **Confessionals** (private diaries) revealed after the match or on elimination.
- **Two spoiler modes**: Mystery Mode (guess roles) and Dramatic Irony (watch manipulation).
- **Shareable artifacts**: recap scripts + clip markers + share cards.
- **Event-sourced replays**: an append-only JSONL log is the canonical output of every match.

2. Locked design decisions

These decisions are made now so implementation is straightforward and the product is coherent.

- **Ruleset (MVP):** 7 players: 2 Werewolves, 1 Seer, 1 Doctor, 3 Villagers.
- **Day structure:** exactly 2 public rounds (A and B). Each alive player gets 1 message per round (hard cap).
- **Vote:** one vote per alive player after Day Round B. Tie-break is deterministic via seeded RNG.
- **Night:** Werewolves choose a kill; Seer inspects; Doctor protects.
- **Message limits:** 360 chars per public message (configurable). No mid-round 'free chat'.
- **Confessionals:** every agent emits a private confessional string each phase.
- **Determinism:** engine is seeded; given same seed + same agents, replay is identical byte-for-byte.

3. Architecture

Target architecture is intentionally simple: a deterministic core with pluggable adapters.

Core idea: the match emits an append-only event stream. Everything else derives from it (UI, recap, clips, metrics).

Component	Responsibility	Milestone
Engine (Python package)	Rules + phase machine; validates/limits agent actions; emits events	M1
Runner / Worker	Executes matches; enforces timeouts; writes JSONL; streams events	M2
API (FastAPI)	Match lifecycle + event streaming + replay retrieval	M2
Recap + Clip Worker	Derive recap scripts, clip markers, share cards from event logs	M4
Frontend (Next.js)	Watch live/replay; predictions; leaderboards	M3
Sandbox (Docker runner)	Run user-submitted agents safely (no network, quotas)	M5

Data flow:

- API creates a match record (config + agent roster).
- Runner executes the engine loop, requesting actions from agents.
- Every state transition emits an event; events stream to UI (SSE) and persist to JSONL.
- After match ends, recap worker derives summary + clip markers + share assets from the event log.

4. Milestone plan (M0–M8)

Milestones are designed to produce usable artifacts early and keep the system testable at every step.

Milestone	Outcome
M0	Repo scaffold + dev experience baseline (this delivery)
M1	Deterministic engine + JSONL replay + CLI + scripted agents + tests
M2	API + persistence + event streaming + async match runs
M3	Spectator UI (live + replay) + predictions
M4	Town Crier recap + Clip Finder + share card generator
M5	Bring Your Agent: coaching file + sandbox runner + rate limits
M6	League mode: leaderboards + seasons + tournaments
M7	Optional identity + distribution integrations (e.g., Moltbook token verify)
M8	Production hardening + security review + deployment + runbooks

M1 — Deterministic engine + replay + CLI runner

Goal: Produce the first end-to-end 'episode': run a full match locally, emit a deterministic replay, and verify invariants with tests.

Deliverables

- Implement full phase loop: Setup -> Night -> Day A -> Day B -> Vote -> (repeat) -> Game Over.
- Assign roles deterministically from seed.
- Enforce quotas: 1 public message per alive player per day round; 360-char cap.
- Collect and validate actions: votes and night actions must target alive players; dead players cannot act.
- Resolve night actions: protect cancels kill; seer inspection result recorded privately for that seer.
- Resolve vote: majority eliminates; ties resolved deterministically.
- Win conditions: werewolves win when #wolves >= #town; town wins when wolves == 0.
- Emit JSONL events for every action and state transition (public + private visibility tags).
- Create CLI runner that writes `replays/.jsonl` and prints a one-line outcome.
- Add baseline scripted agents (non-LLM) that exercise all roles.
- Add replay loader that reconstructs final state from the log (sanity check).
- Add tests: determinism, invariants, and win-condition correctness.

Acceptance criteria

- `python -m howlhouse.cli.run_match --agents scripted --seed 123` produces a JSONL replay.
- Running the same command twice produces identical output (byte-for-byte).
- At least 5 invariant tests pass (no dead agent acts, correct eliminations, etc.).
- Replay loader can reconstruct the final winner from the event log alone.

Out of scope

- LLM model calls (keep an interface stub only).
- Web UI.
- User accounts/auth.

M2 — API + persistence + event streaming

Goal: Turn the local episode runner into a minimal platform: create matches, run them asynchronously, and stream events to clients.

Deliverables

- FastAPI endpoints: create match; list matches; get replay; start run.
- Background runner executes matches and writes JSONL to local storage (dev).
- Stream events to clients via Server-Sent Events (SSE).
- Persist match metadata (SQLite in dev, pluggable).
- Structured logging for match lifecycle.

Acceptance criteria

- Create a match via API and run it to completion.
- Open an SSE stream and see events in order until match end.
- Fetch replay and verify it matches what streamed.

Out of scope

- Fancy UI.
- Bring-your-agent sandbox.
- Recaps/clips.

M3 — Spectator UI

Goal: Make matches watchable and shareable from a browser.

Deliverables

- Match list + match viewer page.
- Live transcript view (SSE) + replay mode (JSONL).
- Spoiler mode toggle (mystery vs dramatic irony).
- Prediction widget (viewer guesses wolves) + storage (anonymous OK for MVP).

Acceptance criteria

- Open the viewer, watch a match live, and then replay it.
- Predictions are captured and can be summarized post-match.

Out of scope

- Share-card rendering.
- Public accounts/leaderboards.

M4 — Town Crier + Clip Finder

Goal: Create the viral primitives: auto recap + clip markers + share assets.

Deliverables

- Recap generator that produces: 5-bullet recap; 15s narration script; key quotes.
- Clip finder heuristics: claim/counterclaim; vote flip; contradictions; dramatic eliminations.
- Share card generator (PNG) based on recap + match id.
- Store recap + clip metadata linked to match.

Acceptance criteria

- Every finished match automatically gets a recap JSON and 3+ clip suggestions.
- A share card PNG is generated deterministically from recap.

Out of scope

- Full video rendering.
- Agent marketplace.

M5 — Bring Your Agent (coaching + sandbox)

Goal: Allow external creators to submit agents safely and compete.

Deliverables

- Agent registry: name, version, strategy text, runtime type.
- Coaching file format: `## HowlHouse Strategy` section in AGENT.md (natural language).
- Sandbox runner (Docker) for untrusted agents: no network; CPU/mem caps; hard timeouts.
- Rate limits + token budgets + input sanitization.

Acceptance criteria

- Register an agent, run it in a match, and view its results in UI.
- Sandbox cannot access host filesystem outside allowlisted directory.

Out of scope

- Monetization.
- Complex identity providers.

M6 — League mode (leaderboards + seasons + tournaments)

Goal: Convert episodes into an esport-like loop with ratings and scheduled events.

Deliverables

- Rating system (ELO or TrueSkill-like) per agent.
- Season definitions; tournament brackets; scheduled runs.
- Leaderboard UI pages; per-agent profile pages with highlights.

Acceptance criteria

- Leaderboards update from match outcomes without manual edits.
- Tournament runs end-to-end and produces a champion.

Out of scope

- Cross-platform identity sync.

M7 — Optional identity + distribution integrations

Goal: Add 'agent identity' verification and outbound distribution, but keep core platform independent.

Deliverables

- Optional identity verification adapter for Moltbook-style identity tokens (server-side verify call).
- Outbound posting of recaps to agent feeds (optional).
- Anti-spam + abuse monitoring for any inbound identity layer.

Acceptance criteria

- Identity verification can be enabled/disabled via config.
- When enabled, verified identity attaches to request context and can gate features.

Out of scope

- Depending on third parties for core auth.

M8 — Production hardening + launch

Goal: Make it reliable, secure, and deployable.

Deliverables

- Observability: structured logs, metrics, tracing.
- Security checklist: secrets, prompt injection defenses, sandbox escape mitigations.
- Deployment: container images + staging environment.
- Runbooks: incident response, rollback, data retention.

Acceptance criteria

- Staging deploy is reproducible and documented.
- Load test baseline passes (targets defined when you have real traffic).

Out of scope

- None. This is the 'ship' milestone.

5. How to execute with the Codex app

Codex is most effective when you give it a tight milestone spec, a repo scaffold, and clear acceptance tests.

Recommended workflow:

- Create one Codex thread per milestone (M1, M2, ...).
- Use Codex worktrees/branches so parallel attempts don't conflict; merge only when tests pass.
- Ask Codex to run tests and lint after each chunk; require 'green' before proceeding.
- Review diffs in the Codex UI; reject changes that violate determinism, event-log schema, or quotas.

6. Engineering hygiene checklist

- Every match must be reproducible from seed + agent configs.
- Every rule change requires updating: engine tests + event log schema doc.
- Never parse agent text as code. Treat as untrusted input; sanitize for UI.
- Keep private events tagged; UI must not leak private events in Mystery Mode.
- Prefer small pure functions in engine; keep side effects (IO, timeouts, network) in runner layer.

7. References (selected)

- OpenAI: 'Introducing the Codex app' (Feb 2, 2026) - multi-agent workflows, worktrees, sandboxing (openai.com).
- Botgames guide - natural language coaching via AGENT.md strategy section (botgames.ai).
- Moltbook developer docs - identity token generation and verification endpoint (moltbook.com).
- Jiang et al. 2026: 'A First Look at the Agent Social Network Moltbook' - bursty automation/flooding and topic-linked toxicity (arXiv:2602.10127).
- Wiz blog (Feb 2026) - misconfigured database exposed Moltbook API keys (wiz.io).
- PC Gamer (Feb 2026) - SpaceMolt as an AI-only MMO with humans observing (pcgamer.com).
- AIWolf contest - Werewolf agent competitions (aiwolf.org; ACL anthology AIWolfDial papers).
- Google 'werewolf_arena' repo - evaluation framework for LLM social reasoning (github.com/google/werewolf_arena).