CASE STUDIES

WizWor

By Austen Tucker

Why I built it

I wanted to learn what it takes to make an agent feel like part of a product, not a chat box pasted onto one. WizWor interviews a player in an original arcade-terminal voice, learns what kind of game they want, and reveals a small set of matches from a real catalog.

OpenAI later released a product experience that handled part of this discovery problem better. That does not make WizWor disposable. It was a serious learning project: I built the system, found its failure modes, and turned those failures into engineering practices I can reuse.

Build snapshot

- Interface: Next.js, React, TypeScript, keyboard controls, browser speech and sound
- Agent layer: OpenAI Agents SDK with typed output and explicit tool contracts
- Grounding: Local classic-game catalogs plus deterministic scoring and ID validation
- Verification: Unit tests, turn-level eval traces, and desktop/mobile Playwright flows

What I did

I split judgment from enforcement

The agent can interpret free-form preferences, decide when it has enough signal, and choose which grounded recommendation to reveal. Ordinary code owns the catalog search, scoring thresholds, accepted IDs, and final display state. The interface never has to trust a model-invented game or match percentage.

I engineered the context, not just the prompt

Each turn receives the selected consoles, a compact preference profile, durable session notes, recent conversation, and a bounded set of scored candidates. The current implementation limits recent mes-

sages and candidate lists so the model sees useful evidence without repeatedly swallowing the full catalog or transcript.

I gave the agent narrow tools

The workflow exposes separate tools for scoring candidates, searching the full catalog by title, and opening the recommendation showcase. The reveal tool validates its inputs against current scoring before the interface changes. Tool use became an observable product event instead of an implication hidden in prose.

I turned failures into eval cases

The eval suite starts from user and job stories, expresses them as turn-level traces, and combines deterministic assertions with rubrics for language and behavior. Fixed bugs become regression cases. Browser flows separately check focus, keyboard use, console selection, reset behavior, and recommendation reveals.

What I learned

Context engineering is product engineering

A model cannot make a good decision from “more context” in the abstract. It needs the right state, in a stable shape, at the moment of decision. Selecting, bounding, and naming that context changed behavior more reliably than piling on extra instructions.

Agent orchestration is a contract, not an agent count

WizWor uses one decision-making agent, typed tools, and deterministic guardrails. That was enough orchestration for this job. The hard part was assigning ownership: the model interprets and chooses; software validates, computes, and renders.

The last mile needs code

An agent saying “I recommend this” is not the same as the product showing a verified result. A dedicated reveal tool, schema validation, scoped retry behavior, and a deterministic fallback close the gap between a plausible answer and a working flow.

Evals need to resemble use

Recommendation scoring alone did not catch conversational stalls, ignored typed input, focus bugs, or a reveal that never opened. The useful suite crosses layers: trace checks for the agent, unit tests for deterministic rules, and browser checks for what the player can actually do.

Current best

WizWor is evidence of a working engineering loop: build a thin product, observe where the agent and interface disagree, move fragile behavior into explicit contracts, and preserve each fix as a test. That learning remains useful even when a larger product later solves part of the original problem better.

Limits and open questions

- There are no public adoption or business-impact results to claim.
- Recommendation quality still depends on catalog metadata and the coverage of the eval cases.
- The live experience uses a hosted model, so a successful session can create API cost.
- A genuine, publication-ready screenshot still needs to be captured and added to this site.