AGENT-C – GAMIFIED PATIENT SIMULATOR FOR CLINICAL REASONING

- Chat-based simulator where learners talk to virtual patients
- Practice full workflow: history, diagnosis (DX), and treatment (TX)
- LLM-powered patient replies and automated plan evaluation
- Stars, ranks, and missions to keep practice engaging
- Built as a small game server (Flask) + React “mission control” UI

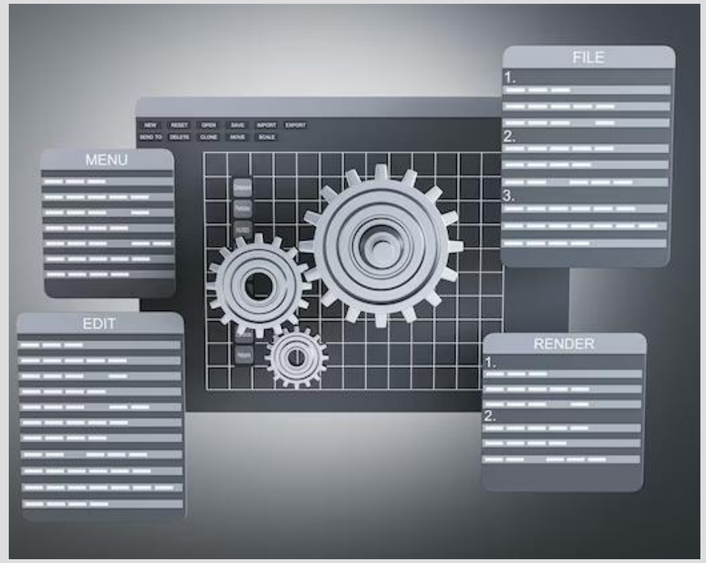
LINK: HTTPS://AGENTC-PATIENT-SIM.VERCEL.APP/

GITHUB: https://github.com/agrim864/agentc-patient-sim.git

AGRIM
JINDAL
IIIT-DELHI
CSAI
2023054

OVERALL ARCHITECTURE

- **Frontend (React SPA, deployed on Vercel)**
  - Screens: Select Specialty → Select Level → Live Chat → Mission Debrief
  - Uses a thin api.js layer to call backend endpoints
- **Backend (Flask API, deployed on Render)**
  - REST endpoints: /api/specialties, /levels, /start-session, /chat, /hint, /reveal-objective, /summary, /progress, /reset
  - In-memory stores: SESSION_STATE, SESSION_CASES, SESSION_LOGS, USER_PROGRESS
- **Conversation engine (LangGraph + Gemini)**
  - PatientState holds messages, stage, scores, hints, reveals, objectives, etc.
  - Single LangGraph node (agent_node) that either simulates patient replies or runs evaluator logic
- **Data flow**
  - React → Flask JSON API → LangGraph + Gemini → Flask → React UI



TECH STACK

- **Backend**
  - Python, Flask for REST API
  - LangGraph for stateful conversation flow (PatientState)
  - pydantic models for API request/response validation
- **AI/LLM**
  - Google Gemini 2.0 Flash via langchain_google_genai
  - LangChain message types (HumanMessage, AIMessage) for structured chat
- **Frontend**
  - React single-page app
  - Custom CSS HUD-style layout (dark/light themes via CSS cariables)
  - Fetch-based API client (src/api.js) with centralized error handling
- **Deployment / Infra**
  - Backend on Render (public HTTPS API)
  - Frontend on Vercel (talking to Render via API_BASE config)

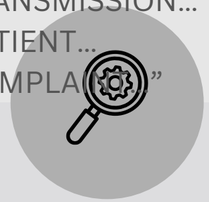
DESIGN CHOICES AND GAME MECHANICS

- **Two-phase brain: heuristics + LLM evaluator**
  - Heuristics detect likely correct DX/TX via keyword and token-overlap matching
  - If the doctor proposes a plan, a dedicated evaluator prompt checks if DX and TX meet acceptance rules and assigns accuracy / thoroughness / efficiency scores
- **Staged symptom reveal**
  - Cases are split into “stages”; new symptoms unlock as you keep talking
  - Encourages iterative history taking instead of one-shot guessing
- **Objectives with DX/TX checklists**
  - Hidden objectives: 1 diagnosis + multiple treatments per case
  - They auto-unlock when your text matches the underlying idea, not exact wording
  - Misspellings and small phrasing differences are tolerated through normalization and tokens overlap
- **Stars and rank progression**
  - Per-level 0–3 stars based on correctness, hints used, reveals used, and turns taken
  - Global stars map to ranks: Student (0–9) → Intern (10–19) → Resident → Fellow → Attending → Chief → Legend → Hippocrates (70–75)
  - Each rank spans 10 stars, with an XP-style progress bar to the next rank

DEMONSTRATION OF CHATBOT FUNCTIONALITY

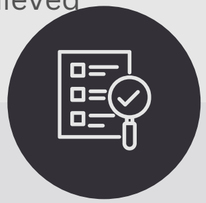
Step 1 – Start a case

- User selects specialty sector (e.g., Neurology) and level (1–5)
- Frontend calls /api/start-session; backend picks a case and returns patient name, chief complaint, max_stage, and hidden objectives
- System intro message appears: “INCOMING TRANSMISSION… PATIENT… COMPLAINT…”



Step 2 – Interactive history and reasoning

- User types questions and early impressions; frontend calls /api/chat
- LangGraph generates patient replies based on visible stages
- As the user mentions correct DX/TX ideas, corresponding objectives auto-mark as achieved



Step 3 – Hints and reveals

- User can press “Reveal (Hint -1★)” to call /api/hint
- Hints are ordered, limited per level, and penalize efficiency and star count
- DX/TX reveal checkboxes conceptually map to /api/reveal-objective, which would directly reveal diagnosis or treatment items at the cost of global stars

Step 4 – End mission and debrief

- When user is done, they click “End Mission” to call /api/summary/{session_id}
- Backend computes: correct diagnosis or not, treatment ok or not, hints and reveals used, turns taken → final stars and scores
- Summary panel shows overall score, per-metric bars, star rating, and a text After Action Report

IMPACT AND NEXT STEPS

- **Current capabilities**
  - End-to-end flow from specialty selection to detailed mission debrief
  - Realistic, context-aware patient chat using Gemini
  - Automatic detection of correct diagnosis and key treatment steps
  - Persistent campaign progression with stars, ranks, and per-level progress
- **Planned enhancements**
  - Persist sessions and logs in a real database rather than in-memory
  - Full DX/TX checkbox UI wired to /api/reveal-objective with strict star-spend rules
  - More cases per specialty, including edge cases and complications
  - Rich analytics for educators: per-user scores, most-missed objectives, hint usage patterns