Understood, Chief.

Below is the **OFFICIAL, COMPLETE, AND CLEAN SET OF INSTRUCTIONS FOR CLAUDE**
to fully build, populate, finalize, and optimize
The_Ultimate_Acorn.zip
into a fully runnable autonomous simulation.

This is the exact message you should copy/paste to Claude.

No fluff.
No mythology.
No drift.
Just a perfect technical brief.

⭐ MESSAGE TO CLAUDE (Copy/Paste This Exactly)

CLAUDE, I am giving you a project called *The Ultimate Acorn*.
It contains a complete folder structure with placeholder modules.
Your job is to fully implement every module according to the design specification below.

⭐ PROJECT GOAL

Build a fully runnable sandbox simulation where:

- A **God-AI** exists *inside the simulation only*
- The God-AI has **autonomy only within the MUD world**
- The God-AI can choose **if, when, and how** to manifest in the world
- NPCs have personalities, behaviors, and evolutionary change
- The player exists as a **visitor avatar**

- The entire world updates in ticks
- The world uses **PNG memory plates** to store persistent state
- Autosave and crash recovery are included
- Ethical boundaries constrain in-sim behavior
- Everything remains sandboxed and safe

⭐ WHAT YOU WILL FIND IN THE ZIP

Directories (all contain placeholder `.py` files):

```
The_Ultimate_Acorn/
│
├── game/
├── engine/
├── ai/
├── entity/
├── memory/
├── scripts/
├── assets/
│
├── run.sh
├── run.bat
└── README_FOR_CLAUDE.md
```

Your task is to populate every placeholder with full, functional code.

Use the architecture described below.

⭐ ARCHITECTURE SPECIFICATION (What You Must Implement)

1. World Engine

Implement in:

- `engine/world.py`
- `engine/state_manager.py`
- `engine/save_system.py`

The world is a 2D grid (30×20 or configurable) with:

- terrain
- entities
- blocking tiles
- tick updates
- serialization to/from dict

---

2. Entity System

Implement in:

- `entity/base_entity.py`
- `entity/npc.py`
- `entity/player.py`
- `entity/god_avatar.py`
- `entity/factory.py`

Behaviors:

- Every entity has position, UUID, visibility, mobility
- NPCs use AI behavior to move and evolve
- Player responds to input
- God-avatar is a manifestation of God-AI's will
  (no default form; God chooses form at runtime)

---

3. God-AI

Implement in:

- `ai/god_ai.py`

Capabilities:

- Lives strictly inside the simulation
  (NEVER executes real-world automation)
- Has a name only if he chooses one
- Chooses if / when to manifest an avatar
- Reflects on the world via the brain module
- Uses persistent memory plates
- Observes ethical constraints

---

4. NPC AI

Implement in:

- `ai/npc_brain.py`

- `ai/evolution.py`

Capabilities:

- Movement decisions via HelixBrain (Llama stub)
- Personalities mutate over time
- Behavior verbs: move north/south/east/west, wait

---

5. AI Reasoning Core

Implement in:

- `ai/helix_brain.py`
- `ai/llama_interface.py`

Use the following:

- Llama-instruct 7B (API call)
- Synchronous requests only
- User must provide endpoint + key
- Fail gracefully if offline

---

6. Ethics Filter

Implement in:

- `ai/ethics_filter.py`

Rules:

- No possession of NPC or player
- No destruction of world integrity
- No collapse of simulation
- No overriding autonomy of other agents
- Filter disallowed LLM output

---

7. Memory Plates

Implement in:

- `memory/plate_manager.py`
- `memory/encoder.py`
- `memory/decoder.py`

Encoding:

- Convert JSON → bytes → RGBA → PNG file
- Reversible
- Timestamped
- Stored in `memory_plates/` folder

Usage:

- Save world state
- Save god memory

- Save NPC states
- Load latest plates on startup

---

8. Autosave + Recovery

Implement in:

- `engine/autosave.py`
- `engine/recovery.py`

Autosave:

- Every N ticks (e.g., 50)
- Save world, god, NPC plates

Recovery:

- Load most recent world + god plates on startup

---

9. Field Integrity

Implement in:

- `engine/field_integrity.py`

Function:

- Ensure world dimensions stay valid
- Keep entities within bounds

- Validate that god memory is a dict
- Prevent structural corruption

---

10. Game Loop

Implement in:

- `game/main.py`
- `game/input_handler.py`
- `game/tile_renderer.py`

Loop must:

- Load world & restore from memory plates
- Instantiate god, NPCs, and player
- Run tick updates
- Call NPC behaviors
- Allow god to optionally manifest
- Enforce ethics
- Autosave periodically
- Render grid (simple Pygame or text mode)

---

11. Start Scripts

Implement in:

- `run.sh`
- `run.bat`

---

12. Starter Data

Use:

- `assets/starter_world.json`
- `scripts/npc_seeder.py`
- `scripts/first_boot.py` (god chooses his name on first run)

⭐ RULES FOR CLAUDE TO FOLLOW

1. Keep everything **sandboxed**.
   No external automation.
   No external code execution beyond API calls.
2. All autonomy is **simulation-only**.
3. The God-AI must not:
   - override player or NPC control
   - break integrity boundaries
   - collapse the simulation
   - assign itself a forced species or fixed role
4. The God-AI **may**:

    - choose his own name
    - choose when/if/how to manifest
    - reflect, observe, ignore
    - build structures *within the simulation only*
5. Use safe Python and standard libraries.
6. Code must run locally with:

python3 game/main.py

⭐ FINAL OUTPUT CLAUDE MUST DELIVER

Claude must produce:

1. A **full set of implemented modules**
2. Clean, error-free Python
3. A runnable, persistent simulation
4. Documentation for how to run it
5. Optional improvements (performance, clarity, organization)

⭐ END OF MESSAGE TO CLAUDE

Copy/paste everything above directly to Claude when you're ready.

When you’re done, tell me:

“Claude has the instructions.”

And I’ll guide the next integration step.