下面是把实现顺序“整理成一个你可以照着做的路线图”，每一步都写清：**要做哪些模块、完成标准（验收点）、以及该不该写测试**。你可以把它当成你的毕设开发计划表。

Milestone 0：全局约定与配置（先钉合同）

目标：把所有模块的“接口/数据格式/配置来源”统一起来，后面不会互相打架。

实现模块

1. configs/config.yaml
2. rpg_story/config.py
3. rpg_story/models/world.py
4. rpg_story/models/turn.py
5. rpg_story/llm/schemas.py（只放 LLM 输出结构，别和 world 模型重复）

验收点

- 所有关键配置项都有默认值/注释：API base_url、model、timeout、retry、RAG top_k、summary_window、session_dir 等。
- models/ 里定义的数据结构能完整表达：
  - WorldSpec（世界观、地点、NPC）
  - GameState（玩家位置、NPC 位置、flags/quests、最近摘要等）
  - TurnOutput（叙事、台词、world_updates、memory_summary）
- 明确“单一真相”：npc 位置只看 npc_locations[npc_id]。

测试（✅必须）

- schema validate：缺字段/错类型会报错（最小单测）。

Milestone 1：持久化与会话（让游戏状态能活下来）

目标：让“下次去断桥还能遇见 NPC”在工程上成立。

实现模块

1. rpg_story/persistence/store.py
2. 约定目录：
    - data/sessions/<session_id>/state.json
    - data/sessions/<session_id>/turns.jsonl

验收点

- save_state/load_state/append_turn_log 能跑通。
- session_id 生成规则明确（例如时间戳+随机串）。
- state.json 能完整还原 GameState。

测试（✅必须）

- 保存→读取→对象等价（核心字段一致）。
- turns.jsonl 能追加多条。

Milestone 2：LLM API 客户端（可替换、可重试、可修复 JSON）

目标：后面 orchestrator 只管“要结果”，不管 HTTP 细节；并且输出能被稳定解析。

实现模块

1. rpg_story/llm/client.py
2. （建议）支持一个 Mock client（未来写测试用）

验收点

- 支持 .env/config 的 base_url/api_key/model。
- timeout + retry（至少 1 次重试）。
- JSON 解析失败能触发一次“fix-json”修复流程。

测试（✅建议）

- 用 Mock client 返回固定 JSON，验证解析流程。
- （可选）把真实 API 调用放到手动测试，不写自动化。

Milestone 3：GameState 与最小回合引擎（先不用 RAG/不用 UI）

目标：先做一个“CLI 一回合能跑”的最小闭环：输入一句话→得到结构化输出→更新 state→落盘。

实现模块

1. rpg_story/engine/state.py
2. rpg_story/engine/orchestrator.py（先做 TurnPipeline 最小版）
3. rpg_story/prompts/*（system/narrator/npc persona 模板读取）
4. rpg_story/cli.py + scripts/run_cli_demo.py

验收点

- CLI 能跑一回合：给定 player_text 和 npc_id，得到 TurnOutput。
- 能把非移动类更新（flags/quests/summary）写入 state。
- 每回合结束自动 persist + log。

测试（✅必须）

- orchestrator 单回合 e2e（用 MockLLMClient，不走网络）。

Milestone 4：世界生成 WorldGen（RPG 开局体验）

目标：Start 后先输入一句话生成世界（世界观+地点+NPC），再进入游戏。

实现模块

1. rpg_story/world/generator.py
2. rpg_story/world/consistency.py
3. rpg_story/world/content/{locations.py,map.py,npcs.py}（辅助操作/建图）
4. scripts/create_world.py

验收点

- 给定 world_prompt 能生成 WorldSpec（结构化 JSON）。
- 校验：
  - location_id 唯一
  - connected_to 指向存在的地点
  - starting_location 存在
- 生成后初始化 GameState（player_location、npc_locations）并保存 session。

测试（✅必须）

- connected_to 非法时报错/拒绝。
- consistency 能命中禁词并触发重写（至少流程可跑）。

Milestone 5：validators（移动合法性 + 可达性）

目标：LLM 不能随便“传送”NPC；移动必须合法。

实现模块

1. rpg_story/engine/validators.py

验收点

- npc_move 校验包含：
  - npc 存在
  - from_location == 当前 npc_locations
  - to_location 存在
  - BFS 可达（connected_to 图）
- 不合法时：拒绝该 move，并写入 turn log。

测试（✅必须）

- BFS reachability 测试。
- to_location 不存在 → reject。
- from_location 不匹配 → reject。

Milestone 6：agency（NPC 不一定听话，可持续人格）

目标：执拗 NPC 会拒绝搬家；引擎决定是否执行 move（不信 LLM）。

实现模块

1. rpg_story/engine/agency.py
2. engine/orchestrator.py 升级：在 apply move 前走 agency gate

验收点

- 同一个 NPC（stubbornness 高）在相同请求下结果稳定（确定性）。
- 拒绝时：
  - state 不更新 npc_locations
  - turn log 记录 refusal_reason
  - 对话里能反映拒绝（由 LLM 或引擎插入）

测试（✅必须）

- stubbornness=0.9 的 NPC 对“去别处等我”必拒绝（或高概率拒绝但你要做成确定性更好写报告）。
- obedient NPC 在同样请求下接受。

Milestone 7：RAG（先接口固定，再接向量库）

目标：长程记忆与地点/NPC 相关检索；每回合强制注入世界规则与关键上下文。

实现模块

1. rpg_story/rag/sources/*（把 world bible/location/npc/summary 变成 docs）
2. rpg_story/rag/retriever.py
3. rpg_story/rag/index.py
4. rpg_story/rag/stores/*（可先空实现/占位）

验收点

- “强制注入列表”每回合必包含：
  1. world bible
  2. current location doc
  3. npc profile doc（对话时）
  4. last N summaries
  5. top_k filtered memories（location_id/npc_id）
- metadata 规范化：doc_type/session_id/location_id/npc_id/turn_id/timestamp。

测试（✅建议）

- always-include 的文档一定出现。
- filter by location_id/npc_id 正确。

Milestone 8：UI（Streamlit，可演示）

目标：真正像“文字对话 RPG”的体验：开始→生成世界→地图→选地点→选NPC→聊天。

实现模块

1. rpg_story/ui/streamlit_app.py

验收点

- World creation page：输入世界 prompt→创建 session→进入游戏页。
- Game page：
  - sidebar 地图切换 player_location（并落盘）
  - 显示该地点 NPC 列表
  - 聊天框调用 orchestrator.turn_phase
- UI 每次交互都从 persistence 同步 state（避免刷新丢状态）。

测试（手动为主）

- UI 自动化难做，建议写“手动验收 checklist”。

Milestone 9：安全与评估（收尾加分项）

目标：毕设写作更稳、更合规、更好展示。

实现模块

1. rpg_story/safety/filters.py
2. rpg_story/eval/*

验收点

- safety：关键词/规则触发后能拒绝或柔性引导。
- eval：能从 turns.jsonl 统计 latency、回合数、基本一致性指标。

测试（可选）

- safety 关键词命中测试。
- eval 脚本跑通即可。

总体注意事项（保证模块能协作）

- **数据模型优先**：所有模块都围绕 models/ 作为唯一真相。
- **orchestrator 不写细节**：它只编排 client/retriever/validators/agency/store。
- **LLM 不可信**：必须有 parse+validate、fix-json、违规重写、move reject 的机制。
- **状态一致性**：npc_locations/player_location 是权威；UI 展示通过计算得到。
- **每个 milestone 都要能“独立验收”**：避免最后一次性集成爆炸。

“每实现一个模块需不需要测试？”

建议遵循这个规则：

- **涉及契约/状态/规则**的模块：models、persistence、validators、agency、orchestrator → **必须写测试**
- **涉及 UI / 脚本**：以手动验收为主
- **RAG/向量库**：先写“接口正确性测试”，效果评估后置

如果你愿意，我可以把这些 Milestone 再压缩成一个**“两周/三周的时间安排表”**（每天做什么、做到什么算完成），更像毕设开发计划。