REQ3 THE ORACLE

SacredScroll & Action Classes for Cohesion & DRY

- SacredScroll extends the engine's Item and carries either a Riddle or a ProphecyType flag.
- Two concrete Action subclasses: RiddleAnswerAction and ProphecyAction, both extend the engine's Action.
- This encapsulates all "what happens next" logic inside the scroll's associated Action, so the Oracle need only ask "which Action?" rather than embed the scroll logic directly.

Oracle as Director

- Oracle extends Actor and inspects the SacredScroll state with simple if(hasRiddle) / else guards.
- It creates the proper Action instance, handing off the execution of the action.

DialogueGenerator Interface for Separation of Concerns

- A single DialogueGenerator interface defines generate(ResultData). Three implementations: RiddleGenerator, ProphecyGenerator, ComplimentGenerator produce the final text and deal with the response from the API.
- RiddleGenerator specifically receives JSON from the API, this functionality is put in the processResponse() function.
- Both Action classes and Oracle depend only on the engine classes, not being responsible for dialogue.
- This allows swapping out generators, or adding new functionality, with other interactions or responses to the oracle without modifying Oracle.

ApiHandler for External Data Fetch

- All HTTP/network calls are isolated in ApiHandler under util/.
- DialogueGenerator implementations call into this handler when they need dynamic or procedurally generated content.
- ApiHandler also handles any potential JSON output from the API.
- Game code remains synchronous and testable, with a single replaceable mock point for network behaviour.

Trade-Offs & Limitations

- **Oracle Growth**: Every new kind of prophecy added to the Oracle will bloat the class.

Principles Applied

- **Single Responsibility**:
  - Action subclasses handle only their scroll outcome logic.
  - DialogueGenerator implementations handle only text construction.

- **Open/Closed**:
    - New scroll types or dialogue styles can be added by subclassing without modifying existing classes.
- **Dependency Inversion**:
    - High-level modules (Oracle, Actions) depend on abstractions (DialogueGenerator), not on concrete classes.
- **High Cohesion & Low Coupling**:
    - Each class has one focused purpose and minimal dependencies on others.