REQ3 – Design Rationale: NPC Dialogue

REQ3 introduces special NPCs (Sellen, Kale, and Guts) capable of generating contextual monologues based on the player's status (e.g., health, inventory) and surrounding ground (e.g., BLIGHT). The system must support multiple types of dialogue logic while remaining modular, extensible, and cleanly separated from presentation or behavior logic. Additionally, REQ3 involves hostile actors that require cleanly defined, reusable combat behavior logic.

Design 1: Hardcoded dialogue directly in NPC

Pros	Cons
Simple and fast to implement	High code duplication; not scalable
No extra classes or wiring required	Logic and presentation tightly coupled
Easy to debug when NPC count is low	Difficult to test or update behavior independently

Design 2: Centralized ListenAction

Primary Design

Pros	Cons

Cleanly separates output (Action) from logic (NPC)	Slightly more setup per NPC
Highly reusable across multiple NPCs	Each NPC must still prepare appropriate lines
Centralized formatting and interaction contract	Custom behaviors still live inside NPC

Design 3: Passive

MonologueBehaviour

Used by Sellen

Pros	Cons
Adds atmospheric, lore-rich worldbuilding	Not interactive; no player-triggered response
Fully decoupled from player actions	Cannot respond to changing player or map state
Lightweight and modular	Limited contextual flexibility

Design 4: Inline contextual logic in NPC (Used by Kale & Guts)

Pros	Cons
Supports dynamic dialogue based on player stats or environment	Logic tightly coupled to each NPC class
Enables immersive, responsive interactions	Low reusability unless abstracted
Simple and readable for small-scale use	Harder to share logic between NPCs

---

Design 5: Interface-based Attack Logic via

HostileAttacker

Combat Behavior Design

Pros	Cons
Removes instanceof and downcast in AttackBehaviour	Requires extra interface and constructor setup
Follows OOP principles (OCP, DIP)	Slightly more boilerplate per hostile actor

Behavior is now injected, not hardcoded	Requires coordination between NPC and behavior
Enables clean testing, reuse, and logic separation	Must manage capability + logic explicitly

Justification

The final design of Requirement 3 integrates four layered strategies to support scalable NPC interaction, dialogue delivery, and hostile behavior management. It combines Design 2 (ListenAction) as the reusable interaction mechanism, Design 3 (MonologueBehaviour) for passive ambient dialogue, Design 4 for contextual monologue generation within individual NPCs, and Design 5, which introduces an interface-driven attack logic system via HostileAttacker.

To support modular dialogue presentation, ListenAction is used across all interactable NPCs to decouple the *what* (message generation) from the *how* (rendering and formatting). This enables NPCs to follow a unified interaction contract while centralizing output logic in a single reusable class. For instance, characters like Kale and Guts dynamically generate monologues based on player state (e.g., health, inventory), but offload the presentation to ListenAction, thereby separating logic from UI. For non-interactive ambiance, MonologueBehaviour enables characters like Sellen to periodically speak without direct player involvement, enriching world immersion with minimal coupling.

In terms of combat logic, a key architectural enhancement is the introduction of the HostileAttacker interface. Hostile NPCs such as Guts implement this interface to define custom canAttackTarget() logic (e.g., only attacking conscious players with sufficient HP). This interface is injected into AttackBehaviour, allowing hostile logic to be reused or extended without modifying the behavior controller. This approach aligns with the Dependency Inversion Principle (DIP) and eliminates the need for instanceof or downcasting, promoting polymorphic flexibility and reducing long-term coupling.

The design strongly adheres to SOLID principles:

- Single Responsibility Principle (SRP): Dialogue generation, rendering, and combat logic are each handled by separate, focused modules (NPC, ListenAction, HostileAttacker, etc.).

- Open/Closed Principle (OCP): New types of dialogue or combat logic can be introduced without altering existing code.

- Liskov Substitution Principle (LSP): All NPCs using ListenAction or MonologueBehaviour conform to the same behavioral contract and can be substituted interchangeably.

- Interface Segregation Principle (ISP): Only relevant NPCs implement HostileAttacker; passive ones remain unaffected, ensuring minimal interface overhead.

- Dependency Inversion Principle (DIP): High-level behavior logic in AttackBehaviour depends on abstract interfaces rather than concrete implementations like Guts.

This architecture also effectively avoids several code smells:

- Duplicated Code: Dialogue output and combat validation are centralized.

- Shotgun Surgery: Changes to dialogue formatting only affect ListenAction.

- God Classes: NPCs maintain single, focused responsibilities without becoming bloated.

From a connascence perspective, this design reduces both Connascence of Name and Connascence of Algorithm between NPCs and behaviors. Behavior classes like AttackBehaviour do not need to know the internal logic of their actors, but simply depend on well-defined contracts such as canAttackTarget(). This lowers coupling, enhances readability, and simplifies future refactoring.

Lastly, the system aligns with Design-by-Contract principles. NPCs commit to providing valid outputs (dialogue or attack decisions), while ListenAction and AttackBehaviour guarantee consistent and structured execution of those outputs. Each component operates under clearly defined preconditions and postconditions, ensuring predictable behavior and facilitating robust testing and extension.