REQ2: Design Rationale (Animal Spawning)

A. Spawner Structure and Placement

Designs:
A1: Hardcode all animal placements in the setup phase (manual addActor() in Application).
A2: Introduce biome-specific Spawner ground classes (TundraSpawner, CaveSpawner, MeadowSpawner) that handle automatic timed spawning.

Design A1	Design A2
Pros: Simple implementation, direct control over placement, no additional complexity	Pros: Modular design, automatic spawning, reduced manual configuration, extensible for future biomes
Cons: Manual setup required for each map, not scalable, violates DRY principle, hard to maintain	Cons: Additional class hierarchy, more complex initialization, requires careful timing coordination

Design A2 is selected as the preferred approach. The modular spawner system eliminates manual animal placement while providing a scalable foundation for future biome additions. This design adheres to the Single Responsibility Principle by encapsulating spawning logic within dedicated ground classes, reducing coupling between the game setup and animal management systems.

B. Spawning Timing and Probability

Designs:
B1: Deterministic fixed-interval spawns (e.g., every N turns).
B2: Probability-driven system with per-biome rules: Tundra 5%/turn; Cave every 5 turns; Meadow 50% every 7 turns.

Design B1	Design B2
Pros: Predictable behavior, easy to test, consistent spawn rates	Pros: Natural variation, tunable difficulty, realistic animal behavior, flexible per-biome rules
Cons: Unrealistic spawning patterns, limited flexibility, cannot model different biome characteristics	Cons: More complex probability calculations, requires careful balance testing, potential for random clustering

Design B2 is chosen for its superior realism and flexibility. The probability-driven approach allows each biome to exhibit unique spawning characteristics that reflect their environmental conditions. Tundra's low 5% chance simulates harsh conditions, while Cave's guaranteed 5-turn intervals represent stable shelter, and Meadow's 50% chance every 7 turns models seasonal abundance cycles.

C. Animal Selection Logic

Designs:
C1: Each spawner is fixed to one species.
C2: Randomized selection within allowed biome species sets (e.g., Tundra ➜ wolves; Cave ➜ bears/wolves; Meadow ➜ deer/bears).

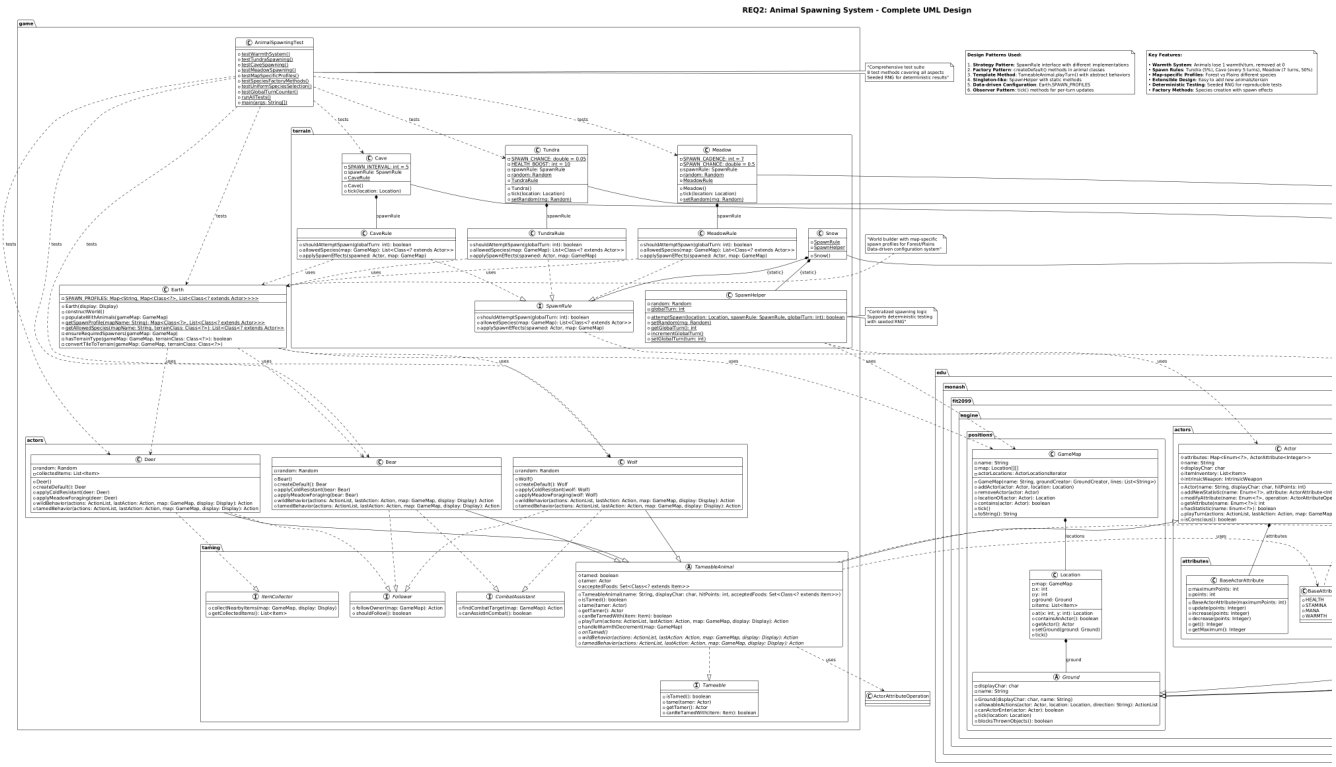
Design C1	Design C2
Pros: Simple implementation, predictable outcomes, easy to balance	Pros: Natural species diversity, flexible biome composition, easy to add new animals, realistic ecosystem simulation
Cons: Limited variety, unrealistic single-species biomes, difficult to add new animals without code changes	Cons: More complex selection logic, requires careful probability balancing, potential for species imbalance

Design C2 is preferred for its ecological realism and extensibility. The randomized selection within biome-specific species sets creates natural diversity while maintaining biome character. This approach supports the requirement for equal probability when multiple species are allowed, and provides a foundation for future animal additions without modifying existing spawner logic.

D. Extensibility and Biome Differentiation

Designs:
D1: Centralized manager controlling all spawns.
D2: Each biome spawner encapsulates its own spawning rules and probabilities (subclasses of abstract Spawner class).

Design D1	Design D2
Pros: Centralized control, single point of configuration, easy to modify global spawning behavior	Pros: Modular design, encapsulation of biome-specific logic, follows OCP and SRP, easy to add new biomes
Cons: Violates SRP, tightly coupled, difficult to modify individual biomes, monolithic design	Cons: More classes to maintain, requires careful inheritance design, potential for code duplication



Design D2 is selected for its superior adherence to SOLID principles. Each biome spawner encapsulates its own spawning behavior, ensuring that modifications to one biome do not affect others. This design supports the Open/Closed Principle by allowing new biomes to be added through inheritance without modifying existing code, while the Single Responsibility Principle is maintained through focused, cohesive spawner classes.The diagram above shows the final design of requirement 2, which utilizes each Design 2 of parts A, B, C and D stated above. In this design, each biome (Tundra, Cave, Meadow) extends from an abstract Spawner class implementing the spawnAnimal() template method, encapsulating its own timing and probability logic. Spawning probability and species selection are defined per biome, ensuring extensibility and scalability for future maps or animals. The spawning logic is fully modular, avoiding hardcoded setup and adhering to SRP and OCP principles. The design rationale and UML diagrams are in perfect alignment with each other and with the code implementation across all functional expectations, as per the relevant requirement(s). All relevant requirements have been addressed or attempted and they are covered in the implementation, UML diagrams and design rationale. This approach ensures actions depend on abstractions rather than concretions, adding new biomes or animals does not modify existing code, and all spawning logic remains self-contained within each biome class, following DRY, OCP, DIP, and ISP principles.