# Página 3

## 1. Idea principal del proyecto

La idea central de **Legacy Creatures – Corey** es convertir el bestiario de Minecraft en un sistema de enemigos "con progresión", donde:

- Cada mob puede pertenecer a un **tier de poder**: `NORMAL`, `EPIC`, `LEGENDARY`, `MYTHIC`, `DEFINITIVE`.
- No todos los mobs llegan a todos los tiers: se controlan con tags como `tier_leve`, `tier_basic`, `tier_intermediate`, `tier_hard`.
- La **dificultad efectiva del mundo** (sistema de `EffectiveDifficulty` + `MobTier`) decide si un mob concreto sube de tier cuando aparece.
- Al subir de tier, el mob:
  - Gana más vida y daño (escalado de atributos).
  - Cambia visualmente (nombre coloreado, partículas).
  - Puede recibir **mutaciones** y efectos especiales.
  - Tiene acceso a **loot propio de su tier**, con recompensas cada vez más fuertes (epic → legendary → mythic → definitive).

En otras palabras: es un sistema de "mobs élite / jefes" emergentes, pero completamente integrado con dificultad, tags y loot progresivo.

## 2. Cómo se ha tomado el proyecto a nivel técnico

### Clasificación de mobs por categoría

- En `data/legacycreaturescorey/tags/entity_type/` tienes tags como:
  - `tier_leve.json` → mobs ligeros: solo pueden ser `NORMAL` o `EPIC`.
  - `tier_basic.json` → pueden llegar hasta `LEGENDARY`.
  - `tier_intermediate.json` y `tier_hard.json` → pueden llegar hasta `MYTHIC` y `DEFINITIVE`.
- `TierManager` usa estos tags para construir el conjunto de tiers permitidos por tipo de entidad:

- `tier_leve` → `NORMAL` + `EPIC`
- `tier_basic` → `NORMAL` + `EPIC` + `LEGENDARY`
- `tier_intermediate` / `tier_hard` → todos los tiers (`NORMAL` … `DEFINITIVE`)

Así se evita, por ejemplo, que un lobo "leve" tenga loot legendario o definitivo.

## Sistema de dificultad y elección de tier

- La dificultad global se calcula con clases en `difficulty/` (`EffectiveDifficultyCalculator`, `DifficultyManager`, etc.).
- `TierManager.tryCategorize(mob, effectiveDifficulty)`:
  - Mira qué tiers son válidos para ese mob según sus tags.
  - Aplica reglas de debug (forzar tier concreto, forzar el más alto posible, etc.).
  - Usa `TierProbabilityCalculator` para **tirar la "ruleta de tier"** en base a la dificultad efectiva y los tiers permitidos.
  - Si el mob sube de tier:
    - Ajusta vida y daño usando `MobAttributeDataLoader`.
    - Aplica nombre custom, color y partículas (`applyVisuals`).
    - Asigna mutaciones (`MutationAssigner`) y furia (`FuryHelper`).
    - Guarda el tier en el componente `MOB_LEGACY`.

## Sistema de loot por tier

- En `loot/data/` están los cargadores:
  - `TieredLootDataLoader`, `TieredLootManager`, `TieredMobLoot` y `IntRange`.
- En tiered tienes la estructura de loot:
  - `epic/`, `legendary/`, `mythic/`, `definitive/`…
  - Cada archivo es por entidad (`zombified_piglin.json`, warden.json, drowned.json, etc.).
- Cada JSON define:
  - `origin_tier`: de qué categoría de origen viene ese mob (`leve`, `basico`, `intermedia`, `dificil`) para documentar y equilibrar.

- `entity`: el `EntityType` (`minecraft:warden`, `minecraft:drowned`, etc.).
- `rolls`: rango de tiradas de loot.
- `guaranteed`: drops garantizados.
- `weighted`: drops ponderados, a veces con `stack` y `components` para:
  - Encantamientos concretos en libros y armas.
  - Pociones con efectos específicos.
  - Ítems especiales (elytra, netherite, beacons, conduits, trial keys...).

Has ido **limpiando incoherencias** (por ejemplo, eliminando loot legendario para `tier_leve`) y luego añadiendo loot mítico y definitivo solo para los mobs que realmente pueden alcanzar esos tiers según `TierManager`.

### Mutaciones y "fantasía" de diseño

Aunque aquí nos hemos centrado más en el loot y los tiers, el proyecto también plantea:

- Mutaciones (en `mutation/`) que cambian el comportamiento o las estadísticas del mob.
- Efectos visuales y de feedback (partículas, nombres, colores, log de debug) para que el jugador sienta claramente cuándo un enemigo es especial.
- Configuración (`CoreyConfig`) que te permite ajustar dificultad, forzar tiers para pruebas, controlar el log, etc.

## 3. Cómo se traduce esto en experiencia de jugador

En la práctica, el proyecto se ha planteado para que:

- **El mundo escale con el jugador**: a medida que sube la dificultad efectiva, aparecen más mobs de tiers altos (épicos, legendarios, míticos, definitivos).
- Cada encuentro puede ser:
  - Un mob normal (fácil).
  - Un mini-jefe épico o legendario, con más vida, mutaciones y buen loot.
  - Un jefe "definitivo" con loot absurdo (netherite, beacons, trial keys, libros rotos, etc.).

- El hecho de que todo esté atado a **tags (`tier_leve`, `tier_basic`, etc.), tiers y loot por entidad** te da una base muy modular:
  - Puedes añadir o quitar mobs de un tag y automáticamente cambian las posibilidades de tiers.
  - Puedes ajustar un JSON de loot sin tocar código Java.
  - Puedes equilibrar el juego por "familias de mobs": nether, end, raids, acuáticos, etc.

Si quieres, en el siguiente paso puedo ayudarte a escribir un texto corto y pulido (por ejemplo, para la página del mod o el `README`) que resuma esto en 2–3 párrafos en tono más "marketing".