Design Document: "Project Viscera" (Retro Gore System)

1. Executive Summary

This feature introduces a high-impact, "Project Brutality" inspired gore system. Unlike modern 3D engines that use decals and meshes, we must respect the engine's retro raycasting/billboarding architecture. The system will generate procedural "Gibs" (meat chunks), volumetric blood sprays (particles), and persistent surface staining (decals) using efficient pooling to ensure memory safety.

2. Core Systems

A. `GoreManager` (New Manager)

A centralized system to handle the lifecycle, updating, and pooling of gore entities.

- **Responsibilities:**
  - Manage pools for `BloodParticle`, `Gib`, and `SurfaceDecal`.
  - Update physics for active particles.
  - Handle collision detection with the floor (y=0) and walls (grid boundaries).
  - Render order sorting (to ensure blood draws behind monsters but in front of walls).

B. `GoreParticle` (New Class)

Represents ephemeral fluids (blood drops, mist).

- **Attributes:** Position (Vector3), Velocity (Vector3), Color (RGBA), Size, Lifetime.
- **Behavior:** Simple gravity physics. Upon hitting the floor (y=0), it converts into a `SurfaceDecal` (puddle) or disappears.

C. `Gib` (Extension of CorpsePart)

Represents solid matter (entrails, eyeballs, bone fragments).

- **Extension:** We will modify or subclass `CorpsePart` to support "micro-sprites" (1x1 or 2x2 pixel chunks) rather than just full sprite quadrants.
- **Physics:** Higher bounce factor, rotation, and friction.

D. Persistent Decals (`SurfaceDecal`)

- **Floor Decals:** Simple XY coordinates with a specific sprite/color drawn at Y=0.
- **Wall Decals:** This is the trickiest part in a raycaster. We will store decals as "Attachments" to specific GridPoint2 coordinates and cardinal directions (North/South/East/West faces).

3. Visual Aesthetic (The "Project Brutality" Feel)

1. **Exaggerated Velocity:** Blood shouldn't just drop; it should *explode* outward from the impact vector.
2. **Color Variance:** Use a palette of reds—bright crimson for fresh spray, dark maroon for floor puddles, and nearly black for "dried" gore.
3. **Volume:** One hit should generate dozens of particles, not just one.

4. Technical Implementation Strategy

Since `EntityRenderer` performs manual 3D projection (translating World XYZ to Screen XY), we **cannot** use standard LibGDX `ParticleEffect` classes, as they won't respect the depth buffer or the pseudo-3D perspective. We must implement a custom renderer that utilizes the existing `transformX / transformY` logic found in `EntityRenderer`.

Project Plan & Milestones

Phase 1: Infrastructure & Particles

Goal: Get blood particles spraying and falling to the ground using the engine's projection math.
Changes:

1. Create `GoreManager` class.
2. Create `BloodParticle` class.
3. Update `Maze` to hold the reference to `GoreManager`.
4. Update `EntityRenderer` to draw these particles.

Milestone 1: Compile and Run. *Result: Hitting a monster generates a cloud of red squares that fall through the floor.*

Phase 2: Physics & Floor Pooling

Goal: Particles stop at the floor, pool into puddles, and old particles are recycled (Memory Safety). **Changes:**

1. Implement object pooling in `GoreManager`.
2. Add floor collision logic.
3. Create `SurfaceDecal` class for floor puddles.

Milestone 2: Compile and Run. *Result: Blood sprays, hits the ground, and turns into static flat puddles.*

Phase 3: "Gibs" & Dismemberment

Goal: Replace the current "4-quadrant crumble" with explosive distinct chunks. **Changes:**

1. Modify `CombatManager` to calculate "Overkill" damage.
2. Create `GibType` enum (Bone, Intestine, Eye, Meat).
3. Update `EntityRenderer` to handle small, rotating solid chunks.

Milestone 3: Compile and Run. *Result: Killing a monster causes it to explode into various distinct debris types.*

Phase 4: Wall Splatters (Advanced)

Goal: Blood hitting a wall stays there. **Changes:**

1. Implement raycasting in `GoreManager` to detect wall hits before floor hits.
2. Store wall decals in `Maze` keyed by GridCoordinate + Face.
3. Update `EntityRenderer` to draw wall overlays.

Milestone 4: Compile and Run. *Final Feature Complete.*