Design Document: Dynamic Weather System

1. Overview

The Weather System adds dynamic, biome-dependent atmospheric effects to the game. It is active **only** when the player is on **Level 1** (Ground Level). It manages visual effects (rain, snow, fog, lighting), audio ambience (wind, thunder), and gameplay modifiers (stat penalties, visibility).

2. New Classes & Enums

2.1 `WeatherType` (Enum)

Defines the distinct types of weather events.

- `CLEAR`: High visibility, standard skybox.
- `RAIN`: Reduced visibility, rain particles, wet sounds.
- `STORM`: Low visibility, heavy rain, lightning flashes, thunder, high wind.
- `SNOW`: Reduced visibility, snow particles, cold/wind sounds.
- `BLIZZARD`: Very low visibility, heavy snow, wind howling.
- `FOG`: Dense fog, muffled audio (optional).
- `TORNADO`: Special event. Visible funnel cloud entity, extreme wind, particle chaos.

2.2 `WeatherIntensity` (Enum)

Defines the severity of the weather, acting as a multiplier for effects.

- `LIGHT`, `MEDIUM`, `HEAVY`, `EXTREME` (Hurricane/Blizzard).

2.3 `WeatherManager` (Class)

The core brain of the system.

- **Responsibilities:**
  - Tracks current `WeatherType` and `WeatherIntensity`.
  - Manages timers for weather transitions (e.g., Clear -> Light Rain -> Storm).
  - Determines valid weather based on the current `Biome` (via `WorldManager`).

    - Calculates target fog distances and colors for `FirstPersonRenderer`.
    - Manages the "Flash" effect for lightning.
    - Triggers sounds via `SoundManager`.

2.4 `WeatherRenderer` (Class)

A helper rendering class used by `FirstPersonRenderer`.

- **Responsibilities:**
    - Renders precipitation (Rain/Snow) using LibGDX `ParticleEffect` or a custom lightweight quad system (for retro feel).
    - Renders global full-screen flashes for lightning.
    - Renders the Tornado visual (billboarded texture) if applicable.

3. Integration Points

3.1 `WorldManager`

- **Updates:** Must hold an instance of `WeatherManager`.
- **Logic:** In `update()`, if `currentLevel == 1`, tick the `WeatherManager`. If `currentLevel > 1`, disable weather.
- **Biome:** Pass current chunk's `Biome` to `WeatherManager` so it knows if it should snow (Mountains) or rain (Forest/Plains).

3.2 `FirstPersonRenderer`

- **Fog:** Instead of static fog constants, query `WeatherManager.getCurrentFogDistance()` and `.getCurrentFogColor()`.
- **Lighting:** Apply a global brightness modifier from `WeatherManager` (for darkening during storms or flashing during lightning).
- **Render Pass:** Call `weatherRenderer.render(...)` after drawing the maze but before the HUD.

3.3 `SoundManager`

- **Ambient Loops:** Needs new functionality to handle cross-fading ambient tracks (e.g., fading out "Forest Ambience" and fading in "Heavy Rain").
- **One-Shot:** Play thunder or wind gusts.

3.4 `Player` / `PlayerStats`

- **Modifiers:** `WeatherManager` may apply temporary status effects (e.g., `WET`, `COLD`, `UNSTABLE_FOOTING`) reducing dexterity or movement speed.

Project Plan

We will implement this feature in **4 Milestones**.

Milestone 1: Core Logic & Data Structure

Goal: The game "knows" it is raining, but we can't see or hear it yet. The console logs weather state changes.

1. Create `WeatherType` and `WeatherIntensity` enums.
2. Create `WeatherManager` class with the state machine logic.
3. Integrate `WeatherManager` into `WorldManager` (tick it only on Level 1).
4. **Compile & Run:** Verify console logs show "Weather changed to RAIN" when walking in a Forest on Level 1.

Milestone 2: Audio & Atmosphere

Goal: We can hear the weather and the fog visuals react.

1. Update `SoundManager` to support looping sound layers (Rain, Wind).
2. Update `FirstPersonRenderer` to use dynamic fog values from `WeatherManager`.
3. Implement Lightning logic (random bright flashes in `WeatherManager` affecting render brightness).
4. **Compile & Run:** Player hears rain/thunder. Fog gets closer/darker during storms. Screen flashes white for lightning.

Milestone 3: Visual Precipitation (Particles)

Goal: We can see rain and snow falling.

1. Create `WeatherRenderer`.
2. Implement a simple "falling droplets/flakes" system (retro-style lines or dots).
3. Hook `WeatherRenderer` into `FirstPersonRenderer.render`.
4. **Compile & Run:** Visual rain overlays the 3D view.

Milestone 4: Advanced Features (Tornado & Gameplay Stats)

Goal: The Tornado event and player stat penalties.

1. Implement Tornado logic (rare state in `WeatherManager`).
2. Implement Tornado rendering (a scaling sprite in the distance or a "wandering" entity).
3. Update `Player` to check `WeatherManager` for debuffs (e.g., reduced bow accuracy in high wind).
4. **Compile & Run:** Full feature set active.
1.