Design Document: Ranged Combat & Ballistics Overhaul

1. Executive Summary

Current ranged combat is restricted to adjacent tiles, effectively functioning as melee combat with different sprites. This feature will decouple ranged attacks from the "Combat State," allowing players and monsters to fire projectiles across the map. This requires implementing Line-of-Sight (LOS) raycasting for hit detection, ballistic animations that respect depth, and a new RPG calculation system for Accuracy/Evasion.

2. Systems Analysis & Changes

A. Combat Manager (The Core Logic)

- **Current State:** Checks `checkForAdjacentMonsters()`. If adjacent, enters `CombatState`. Attacks are instant and guaranteed.
- **New State:**
    - Needs a `playerRangedAttack(Direction dir)` method.
    - This method fires a "Logical Projectile" (Raycast) along the vector.
    - If it hits a wall: The attack ends there (wasted arrow).
    - If it hits a monster: Calculate Hit/Miss chance. If Hit, apply damage and aggro.
    - Removes the requirement for `CombatState` to be active to attack.

B. Monster AI Manager (Enemy Behavior)

- **Current State:** Moves randomly or seeks player via A*.
- **New State:**
    - Before moving, check `hasRangedAttack` capability.
    - Perform a LOS check to the player.
    - If visible and in range: Fire projectile instead of moving.

C. Player & Monster Data (RPG Stats)

- **Player:** Needs a `Dexterity` attribute to calculate accuracy.
- **Monster:** Needs `Dexterity` (evasion/aim), `isRanged`, and `attackRange` properties.
- **Items:** Weapons need an `accuracyModifier` (Tier).

D. Visuals (Animation Manager)

- **Current State:** Projectiles interpolate from A to B over a fixed time.
- **New State:**
    - Projectiles must visually stop at the point of impact (wall or entity).
    - We cannot just animate to the target's *center* if we miss; we must animate past them or to the wall behind them.

3. Hit/Miss Calculation Formula

We will implement a standard d100 roll system.

Java
```
HitChance = BaseChance (60%)
        + (Attacker.Dexterity * 2)
        + (Weapon.Tier * 5)
        - (Target.Dexterity * 1.5)
        - (Distance * 5)
```

If `Random(0-100) < HitChance`, damage is applied.

Project Plan

I have broken this down into 5 distinct milestones. We will verify compilation and runtime stability after every step.

Milestone 1: Data Structure Foundation

Objective: Add necessary stats (Dexterity, Range flags) to Player and Monsters so the math can function later. **Tasks:**

1. Update `PlayerStats` to include `dexterity`.
2. Update `MonsterTemplate` to include `dexterity`, `hasRangedAttack`, and `attackRange`.
3. Update `Monster` to read these new fields.

Milestone 2: The Ballistics Logic (Raycasting)

Objective: Create the math to determine *what* is hit when firing a projectile, without playing animations yet. **Tasks:**

1. Implement a `raycastBullet` method in `CombatManager` (or `WorldManager`) that traverses the grid until it hits a Wall, Door, or Entity.
2. Return the coordinate of impact and the entity hit (if any).

Milestone 3: Player Ranged Implementation

Objective: Allow the player to press a button and fire into the dark, consuming arrows. **Tasks:**

1. Update `CombatManager` to handle `performRangedAttack`.
2. Implement the Hit/Miss formula.
3. Hook up Input (I will guide you on where to place the key listener).

Milestone 4: Visuals & Animation Sync

Objective: Make the projectile sprite actually stop where the logic said it hit. **Tasks:**

1. Update `AnimationManager` / `Projectile` to accept a specific `targetWorldPos` rather than just an entity.
2. Refine the flight speed based on distance.

Milestone 5: Monster Retaliation

Objective: Enable monsters to shoot back. **Tasks:**

1. Update `MonsterAiManager` to check for LOS.
2. Trigger the `monsterAttack` logic from a distance.