## **Game Battle System Architecture (Revision 2)**

### **1. Core Principles & System Overview**

This document outlines a turn-based combat system designed for a single player-controlled unit against one or more AI opponents (up to 8 active combatants). The architecture prioritizes strategic depth and player intuition by retaining a proven, deterministic damage calculation core while innovating on its surrounding systems. Key pillars include a streamlined four-stat model, a highly modular data-driven elemental system, a clear binary buff/debuff framework, and a "Graze" mechanic that replaces traditional "miss" chance to reduce player frustration.

### **2. Combatant Attributes and Stats**

Each combatant, player or enemy, is defined by a set of core attributes. These stats form the fundamental inputs for all combat calculations.

* **Level:** The primary scalar for a combatant's power, directly influencing damage output and potentially other secondary calculations.
* **HP (Health Points):** A combatant's vitality. When HP is reduced to 0, they are defeated.
* **Strength (STR):** Governs the potency of **Physical** moves. This stat is used as the primary offensive value for calculating damage with attacks designated as Physical.
* **Intelligence (INT):** Governs the potency of **Magical** moves. This stat is used as the primary offensive value for calculating damage with attacks designated as Magical.
* **Tenacity (TEN):** A combatant's universal resilience. This stat serves as the primary defensive value against all incoming damage, whether from Physical or Magical sources.
* **Agility (AGI):** A combatant's speed and reaction time. This stat is the sole determinant for action order among moves of the same priority level.

### **3. The Turn Sequence: A Step-by-Step Flow**

Combat proceeds in discrete turns, following a strict, unbreakable sequence of operations.

**Phase 1: Start of Turn**

1. **Environment Resolution:** Any persistent environmental effects (e.g., "Acid Rain" DoT) are resolved for all combatants.
2. **Duration Countdown:** All turn-based counters for buffs, debuffs, and environmental effects are decremented by 1. Effects whose duration reaches 0 are immediately removed.

**Phase 2: Action Selection**

1. The player selects an action (Use a Move, Use an Item, Flee).
2. The AI for each enemy combatant selects and locks in their actions.

**Phase 3: Action Resolution**

1. **Build & Sort Action Queue:** A list of all selected actions is compiled and sorted according to two criteria in order:  
   a. **Move Priority:** Higher priority integers execute before lower ones.  
   b. **Combatant Agility:** For moves of identical priority, the combatant with the higher Agility stat acts first. A random roll breaks any ties.
2. **Sequential Execution:** The system processes one action at a time from the top of the sorted queue. An action must be fully resolved (damage dealt, effects applied) before the next action begins.

**Phase 4: End of Turn**

1. **Victory/Defeat Check:** The system checks if the battle's termination conditions have been met.
2. **Loop:** If the battle continues, the system returns to Phase 1.

### **4. Action Resolution: The Damage Formula**

This is the granular, step-by-step process for resolving a single damaging move.

**Step 1: Buff/Debuff Pre-computation (Attacker)**At the start of its turn to act, before any other logic, the active combatant resolves its own status effects.

* **DoT/HoT Effects:** If afflicted with "Poison" (damage-over-time) or "Regen" (heal-over-time), the HP adjustment is made now.
* **Action-Altering Effects:** If afflicted with an effect like "Stun," the combatant's action is immediately cancelled, the effect is consumed, and their turn in the queue ends.

**Step 2: Accuracy Check & The Graze Mechanic**A move's accuracy determines its reliability. This step sets a critical flag for the final damage step.

* **-- Accuracy (Absolute Hit):** The move is flagged as a "True Hit." It **bypasses all accuracy calculations and modifiers**. It cannot be made to Graze by any means (e.g., Agility debuffs, environmental effects). It proceeds directly to Step 3.
* **100 or <100 Accuracy (Roll-Based Hit):** The move's reliability is subject to modifiers.
  1. Calculate CurrentAccuracy = Move\_Base\_Accuracy - Accuracy\_Debuffs.
  2. A random number is generated and compared against CurrentAccuracy.
  3. If the roll succeeds, the move is a **full hit**.
  4. If the roll fails, the move is a **Graze**. The system sets a boolean flag (isGrazed = true). A Grazed attack cannot become a Critical Hit.

**Step 3: Base Damage Calculation**

1. Determine OffensiveStat: Strength for Physical moves, Intelligence for Magical moves.
2. Determine DefensiveStat: Always the target's Tenacity.
3. Calculate BaseDamage:
4. code Code
5. IGNORE\_WHEN\_COPYING\_START
6. IGNORE\_WHEN\_COPYING\_END
7. BaseDamage = (((2 \* Attacker\_Level/5 + 2) \* Move\_Power \* (OffensiveStat / DefensiveStat)) / 50) + 2

**Step 4: Multiplicative Modifier Application**Modifiers are applied sequentially to the BaseDamage value.

1. **Weather:** Apply environmental multipliers if applicable (e.g., \*= 1.5 for a Fire move in "Scorching Sun").
2. **Critical Hit:** If the move was not a Graze, a roll is performed. On a success, damage is multiplied by 1.5x and the check for attacker debuffs and defender buffs (Modifier #3) is skipped.
3. **Attacker/Defender Stat Buffs:** If not a critical hit, apply buffs/debuffs (e.g., attacker's "Strength Up" \*= 1.5; defender's "Tenacity Up" \*= 0.66).
4. **Elemental Effectiveness:** The system queries the **Elemental Interaction Matrix** (see Section 6.1) to get the final multiplier. Damage \*= Final\_Elemental\_Multiplier.
5. **Random Variance:** Damage \*= random\_float(0.85, 1.00).
6. **Graze Application:** If the isGrazed flag is true, this is the final multiplier applied: Damage \*= 0.5.

**Step 5: Final Damage**The final damage value (floored to the nearest integer) is subtracted from the target's HP.

### **5. Status Architecture: Buffs & Debuffs**

The system employs a non-stacking, binary buff/debuff model. Applying an existing effect resets its duration.

* **Format:** (Effect\_ID, Duration\_in\_Turns)
* **Examples:** "Strength Up", "Intelligence Down", "Tenacity Up", "Agility Down", "Poison", "Stun", "Regen".

### **6. Elemental System**

The system separates elements into Offensive (properties of a move) and Defensive (properties of a combatant). A move or combatant can have multiple elements. The final damage multiplier is the product of all interactions. The entire system is designed to be data-driven.

#### **6.1. Data-Driven Elemental System Architecture**

To ensure maximum flexibility, scalability, and ease of iteration, the elemental system is not hard-coded. Instead, it is driven by external data files (e.g., JSON, CSV, or database tables) that the game engine reads at runtime.

**Core Data Structure 1: Elements Table**This is a simple definition list for all possible elements in the game. It allows for easy addition of new elements without touching game code.

* **Schema:** ElementID (integer, unique key), ElementName (string), ElementType (enum: 'Offensive' or 'Defensive').

|  |  |  |
| --- | --- | --- |
| ElementID | ElementName | ElementType |
| 1 | Fire | Offensive |
| 2 | Metal | Offensive |
| 101 | Creature | Defensive |
| 102 | Stone | Defensive |

**Core Data Structure 2: ElementalInteractionMatrix Table**This is the heart of the system. It defines the relationships between elements. The game logic simply queries this table to determine multipliers.

* **Schema:** InteractionID (unique key), AttackingElementID (foreign key), DefendingElementID (foreign key), Multiplier (float).

|  |  |  |  |
| --- | --- | --- | --- |
| InteractionID | AttackingElementID | DefendingElementID | Multiplier |
| 1 | 1 (Fire) | 101 (Creature) | 1.5 |
| 2 | 1 (Fire) | 102 (Stone) | 0.5 |
| 3 | 2 (Metal) | 101 (Creature) | 2.0 |
| 4 | 2 (Metal) | 102 (Stone) | 0.5 |

**Runtime Logic & Scalability:**When a move is used, the game engine performs the following lookup:

1. Initialize FinalMultiplier = 1.0.
2. Retrieve the list of OffensiveElementIDs from the move.
3. Retrieve the list of DefensiveElementIDs from the target.
4. For each OffensiveElementID in the move's list:
   * For each DefensiveElementID in the target's list:
     + Query the ElementalInteractionMatrix for a matching pair.
     + If a match is found, FinalMultiplier \*= result.Multiplier.
     + **If no entry is found, the multiplier is assumed to be 1.0 (neutral).** This is critical, as it means new elements can be added and they will be neutral to all existing elements by default, preventing bugs and simplifying expansion.
5. The FinalMultiplier is then passed to Step 4 of the damage calculation.

This data-driven approach allows a designer to add a new "Water" offensive element and a "Plant" defensive element, define their interaction in the matrix, and have it work in-game instantly without a single line of new code.