The goal is to evolve the combat system from static calculations to a dynamic, event-driven model that supports both elemental relationships and custom status effects.

I have broken down the implementation into three phases: **Data Modeling**, **Elemental Logic**, and **Status Effect/Trigger Integration**.

1. Phase 1: Data Modeling and Elemental Setup 💾

Goal: Define all new data structures and implement the core Elemental Damage lookup table.

File 1: `shared/types.ts` (Data Contract)

Actions for AI Agent:

1. Define the new `SkillElementId` type.
2. Update the `Skill` interface with an `element` field.
3. Add the core interfaces for the new status/effect system.

TypeScript

```
// In shared/types.ts

// NEW ELEMENTAL TYPE FOR SKILLS
export type SkillElementId = ElementId | 'none';

// NEW STATUS EFFECT TYPES (Expandable)
export type EffectTarget = 'self' | 'enemy' | 'party_member';
export type StatType = 'attack' | 'defense' | 'health' | 'elementalAffinity';

export interface StatBuffEffect {
  type: 'stat_buff' | 'stat_debuff';
  stat: StatType;
  value: number; // multiplier (e.g., 1.5 for +50%)
  duration: number; // turns
}
export interface DOTEffect {
  type: 'damage_over_time';
  damagePerTurn: number;
  duration: number;
}
export interface ThornsEffect {
  type: 'thorns';
  damageReturnRatio: number; // e.g. 0.1 for 10%
  duration: number;
}
export interface OneTimeShieldEffect {
    type: 'one_time_shield';
```

```
}
export type CustomEffect = StatBuffEffect | DOTEffect | ThornsEffect | OneTimeShieldEffect;

// UPDATED INTERFACES
export interface Skill {
  // ... existing fields ...
  element: SkillElementId; // <-- NEW
  effects?: CustomEffect[]; // <-- NEW for skills that apply a buff/debuff
}

// NEW INTERFACES FOR ACTIVE TRACKING
export type CombatTrigger =
  | 'on_hit' | 'on_get_hit' | 'on_start_turn' | 'on_end_turn'
  | 'on_enter_battle' | 'on_switch_out' | 'on_knocked_out'
  | 'check_party_element' | 'check_party_lineage';

// This is what is actually tracked on a spirit in battle
export interface ActiveEffect {
  id: string;            // A unique ID for this instance
  effectType: CustomEffect['type'];
  turnsRemaining: number;
  statMultiplier?: number;
  damagePerTurn?: number;
  damageReturnRatio?: number;
  blocksFullHit?: boolean;
}

// Update PlayerSpirit to track active effects (assuming this will be mapped to BattleSpirit)
export interface PlayerSpirit {
  // ... existing fields ...
  currentHealth?: number;
  activeEffects?: ActiveEffect[]; // <-- NEW: Optional field for tracking
}

// Define the structure for complex passives/triggered abilities
export interface TriggeredAbility {
    trigger: CombatTrigger;
    condition?: { type: 'hp_threshold' | 'party_count'; value: any; };
    effects: CustomEffect[];
}

export interface BaseSpirit {
    // ... existing fields ...
    // Note: The simple 'passiveAbility' should be kept for compatibility/display
    triggeredAbilities?: TriggeredAbility[]; // <-- NEW: For complex, extensible passives
}
```

File 2: shared/data/skills.json (Static Data Update)

Action for AI Agent:

1. Iterate over all skill objects and add a simple element property.

JSON

```
// In shared/data/skills.json (Add element: "none" to existing skills)
{
  "basic_attack": {
    // ... existing fields ...
    "element": "none"  // <-- ADDED: Non-elemental by default
  },
  "heavy_hit": {
    // ... existing fields ...
    "element": "none"  // <-- ADDED
  },
  // ... and so on for all skills ...
  // A new skill example:
  "flame_strike": {
    "id": "flame_strike",
    "name": "Flame Strike",
    "damage": 1.2,
    "healing": 0,
    "unlockLevel": 1,
    "element": "fire"
  }
}
```

File 3: client/src/lib/spiritUtils.ts (Elemental Matrix)

Action for AI Agent:

1. Add the ELEMENTAL_MATRIX constant and the getElementalDamageMultiplier helper function.

TypeScript

```
// In client/src/lib/spiritUtils.ts (Add these exports)

// Wood > Water > Metal > Earth > Fire > Wood
const ELEMENTAL_MATRIX: Record<ElementId, Record<ElementId, number>> = {
  wood:  { water: 1.5, fire: 0.75, earth: 1.0, metal: 1.0, wood: 1.0 },
```

```
  water: { metal: 1.5, wood: 0.75, earth: 1.0, fire: 1.0, water: 1.0 },
  metal: { earth: 1.5, water: 0.75, wood: 1.0, fire: 1.0, metal: 1.0 },
  earth: { fire: 1.5, metal: 0.75, wood: 1.0, water: 1.0, earth: 1.0 },
  fire:  { wood: 1.5, earth: 0.75, metal: 1.0, water: 1.0, fire: 1.0 },
} as const; // Use 'as const' to ensure ElementId is used for keys

/**
 * Calculates the damage multiplier based on attacker and defender elements.
 */
export function getElementalDamageMultiplier(attackerElement: ElementId, defenderElement:
ElementId): number {
  return ELEMENTAL_MATRIX[attackerElement]?.[defenderElement] || 1.0;
}
```

---

2. Phase 2: Core Damage and Buff Logic (Extensible) ⚔️

Goal: Implement the new combined Elemental and Affinity Damage formula, and create a central function for executing effects based on triggers.

File 4: `client/src/components/BattleScreen.tsx` (Core Logic)

This is the largest change. The AI Agent must refactor `handleAttack` and introduce several new functions.

Refactoring `BattleSpirit` - Key Changes

1. **Update `BattleSpirit` Interface:** Update the local interface to match the new `PlayerSpirit`/`ActiveEffect` structures.
2. TypeScript

```
// In client/src/components/BattleScreen.tsx
interface BattleSpirit {
  playerSpirit: PlayerSpirit;
  currentHealth: number;
  maxHealth: number;
  activeEffects: ActiveEffect[]; // <-- ADDED
}
```

3.
4.
5. **Affinity / Elemental Damage Logic:** Refactor the damage calculation in `handleAttack` as defined in the previous step.

6. **New Effect and Trigger Execution (Extensibility Point):** Implement a modular `executeTriggerEffects` function. This is the **primary extensibility point** for the new system.

TypeScript

```
// Conceptual function in client/src/components/BattleScreen.tsx

/**
 * Executes all relevant triggered passive abilities and active status effects.
 * This is the central hub for the new combat system.
 */
const executeTriggerEffects = (trigger: CombatTrigger, attacker: BattleSpirit | Enemy, target: BattleSpirit | Enemy | null) => {
    // 1. Process Active Effects on the target (e.g., Thorns on 'on_get_hit', DOT on 'on_start_turn')
    // 2. Process Passive Abilities (from BaseSpirit.triggeredAbilities) on the active player spirit, checking conditions.

    // ... Implement logic for Thorns/Shield here ...
};

/**
 * Applies or updates a status effect to a BattleSpirit's activeEffects array.
 */
const applyStatusEffect = (spirit: BattleSpirit, effect: ActiveEffect): BattleSpirit => {
    // ... logic to check if effect stacks, refresh duration, or apply new ...
    // ... return updated BattleSpirit ...
};

/**
 * Ticks down durations and removes expired effects at the end of a turn.
 */
const tickEffects = (spirits: BattleSpirit[], enemy: Enemy | null): { updatedSpirits: BattleSpirit[], updatedEnemy: Enemy | null } => {
    // 1. For each spirit, check for 'damage_over_time' effects and apply damage.
    // 2. Decrement turnsRemaining on all effects.
    // 3. Filter out all effects where turnsRemaining <= 0.
    // ... return updated state ...
};

// Integrate tickEffects into the existing battle flow:
// Example: In enemyTurn and handleAttack's completion logic:
// setPlayerSpirits(tickEffects(playerSpirits, enemy).updatedSpirits);
// setEnemy(tickEffects(playerSpirits, enemy).updatedEnemy);
```

File 5: `client/src/lib/spiritUtils.ts` (Stat Buff Integration)

Action for AI Agent:

1. Modify `calculateStat` to accept an optional array of `ActiveEffect[]`.
2. Modify `calculateAllStats` to pass the `activeEffects` and loop through them to apply buffs/debuffs from the new system *before* applying the static passive bonus.

This ensures the final stats used in combat are always correct, even with temporary buffs.

3. Extensibility and AI Agent Roadmap

The agent should work in phases and keep the new system modular:

Phase 1: Complete Element System (High Priority)

- Implement all changes in **`shared/types.ts`**, **`skills.json`**, and **`spiritUtils.ts`** (Elemental Matrix).
- Refactor **`BattleScreen.tsx`** `handleAttack` to correctly use the **Elemental Damage formula**.

Phase 2: Status Effect Foundation (Medium Priority)

- Implement the **`BattleSpirit` interface updates** in `BattleScreen.tsx`.
- Implement the empty **`executeTriggerEffects`** and **`tickEffects`** shell functions in `BattleScreen.tsx`.
- Integrate calls to `tickEffects` at the end of `handleAttack` and `enemyTurn` to ensure turn durations work.

Phase 3: Trigger/Effect Implementation (Low Priority/Future Feature)

- **Implement Stat Buffs:** Update `spiritUtils.ts:calculateAllStats` to read active effects and apply stat multipliers (e.g., ATK * 1.5).
- **Implement Simple Effects:** Add the logic for DOT (in `tickEffects`), Thorns (in `executeTriggerEffects` on `on_get_hit`), and Shield (checked in damage application logic).
- **Implement Trigger Logic:** Create helper functions to process party synergy triggers by examining the `playerSpirits` array.

By structuring the code this way, adding a new effect like **Poison** only requires: 1) defining the `DOTEffect` in `types.ts`, 2) adding the DOT logic to `tickEffects`, and 3) adding the `DOTEffect` to a skill/passive's definition. The core combat loop remains untouched.