Documentación
Pokemón Bytes

Documentación - Fase II: Motor de Batalla

ÍNDICE

1. Introducción y objetivos de la Fase

El objetivo de la Fase II ha sido desarrollar el motor de batalla, el núcleo funcional de Pokémon Bytes. Esta fase traslada la lógica de negocio del cliente al servidor, garantizando que todos los cálculos (daños, probabilidades y gestión de estados) sean seguros, deterministas y persistentes.

El sistema implementa las mecánicas de la II GEN de Pokémon, soportando combates por turnos con validación de estados alterados, efectividad entre tipos y persistencia transaccional en la base de datos.

2. Arquitectura del modulo de combate.

El modulo se estructura bajo el patrón Service Layer, desacoplando la orquestación del flujo(reglas del juego) de los cálculos matemáticos puros.

Componentes principales:

- *BatallaController:* Punto de entrada de REST (POST /api/v1/batalla/turno). Recibe DTOs Y valida la seguridad JWT.
- *BatallaService:* Gestiona el ciclo de vida del turno
- *CalculoService:* Motor matemático. Contiene las formulas de daño y probabilidad.
- *TipoService*: Gestor para matriz de efectividad elemental.
- *PokemonUsuario:* Modelo de datos que mantiene el estado vivo del Pokémon(HP, estados,…)

3. Implementación del flujo de turno(BatallaService)

La lógica central reside en el método ejecutarTurno. Este método esta anotado con @Transaccional, asegurando la atomicidad: o se procesan todos los efectos de los turnos o se revierte la operación completa para evitar inconsistencias en la partida.

3.1 Secuencia de ejecución

I. **Carga de entidades:** Se recuperan las instancias de PokemonUsuario (atacante y defensor) y PokedexMaestra desde la base de datos haciendo uso de los repositorios

II. **Validación pre-turno(bloqueo):** Se invoca a verificarEstadoPreTurno() para determinar si el atacante puede moverse.

III. **Calculo de daño:** Si el pokémon puede atacar, se delega en CalculoService para obtener el daño numérico

IV. **Aplicación de daño:** Se actualiza el hpActual del defensor en memoria

V. **Persistencia:** Se guarda el estado actualizado en MySQL mediante pokemonUsuarioRepository.save()

VI. **Efectos pre-turno(Residuales):** Se calculan daños residuales (veneno, quemadura, drenadoras…)

VII. **Construcción de respuesta:** Se genera un TurnoResponse(DTO) con los datos necesarios para que el frontend renderice la animación

4. Logica matematica (CalculoService)

4.1 Formula de daño real

El método calcularDaño() implementa la ecuación de la Generación II/III:

$$Daño = \left(\frac{\left(\frac{2xN}{5}+2\right) x\, P\, x\left(A/D\right)}{50}+2\right) x\, STAB\, x\, TIPO\, x\, RANDOM\, ¿$$

Mapeo de Variables en Código:

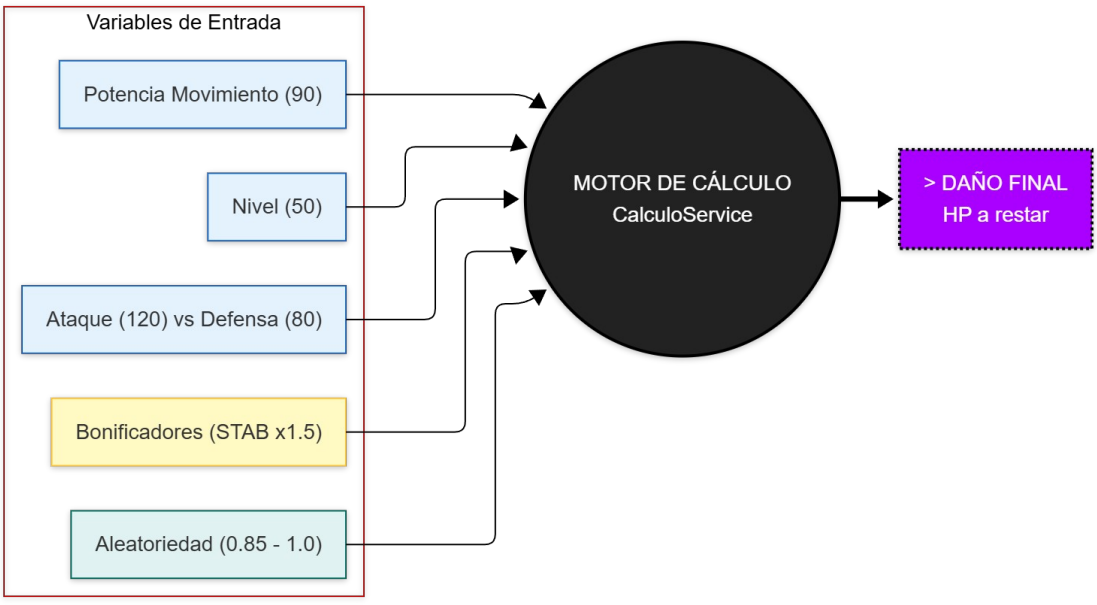
- **N (Nivel):** nivelFactor = 0.2 * nivelAtacante + 1.0;
- **P (Potencia):** potenciaMovimiento obtenido de la tabla ATAQUES.
- **A/D (Stats):** Se seleccionan dinámicamente (ataqueStat vs defensaStat) dependiendo de si el movimiento es Físico o Especial.
- **Modificador de Quemadura:** Implementación fiel a la mecánica clásica:

```
if (estadoAtacante == Estado.QUEMADO && esFisico) {
        ataqueEfectivo = ataqueStat / 2.0; // Reducción del 50%
}
```

- **STAB (Same Type Attack Bonus):** Bonificación de 1.5 si el tipo del ataque coincide con el del usuario.
- **Random (Aleatoriedad):** Se utiliza ThreadLocalRandom para generar un factor de variación entre **0.85 y 1.0**, evitando resultados predecibles.

4.2. Críticos y Precisión

- **Golpe Crítico:** Probabilidad base del 6.25% calculada en fueGolpeCritico().
- **Precisión:** Validación de impacto (verificaImpacto()) basada en la precisión base del movimiento (0-100) frente a un entero aleatorio.



5. Matriz de Tipos Dinámica (TipoService)

A diferencia de implementaciones estáticas , el sistema utiliza una base de datos relacional para gestionar la tabla de tipos, permitiendo re-balanceos sin recompilar el código.

5.1. Inicialización (TipoInitializer)

Mediante un CommandLineRunner, el sistema verifica al arranque si la tabla TIPOS está vacía. Si lo está, carga automáticamente las 17 relaciones elementales (Fuego, Agua, Planta, etc.) con sus multiplicadores:

- **Inmunidad:** x0.0
- **No muy efectivo:** x0.5
- **Eficaz:** x2.0

Result Grid | Filter Rows:

id_tipo	atacante	defensor	multiplicador
93	Eléctrico	Tierra	0
94	Normal	Fantasma	0
95	Fantasma	Normal	0
96	Lucha	Fantasma	0
97	Tierra	Volador	0
98	Veneno	Acero	0
99	Psíquico	Siniestro	0
100	Fuego	Planta	2
101	Fuego	Hielo	2
102	Fuego	Bicho	2
NULL	NULL	NULL	NULL

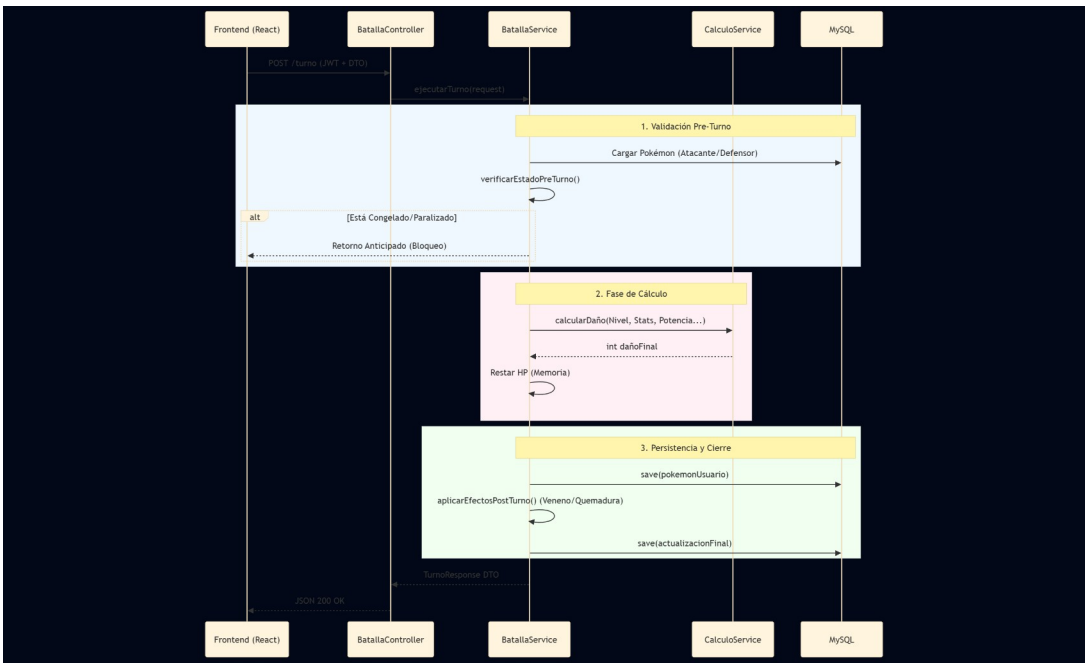
Extracto de la matriz de efectividad cargada en la base de datos relacional.

5.2. Cálculo de Efectividad Dual

El método calcularEfectividad en TipoService soporta Pokémon de doble tipo (ej. Charizard: Fuego/Volador).

```
// Consulta a BD para el primer tipo defensivo
    double mult1 = tipoRepository.findByAtacanteAndDefensor(tipoAtaque,
    tipo1).getMultiplicador();
// Consulta a BD para el segundo tipo defensivo
    double mult2 = tipoRepository.findByAtacanteAndDefensor(tipoAtaque,
    tipo2).getMultiplicador();
    return mult1 * mult2; // Resultado final (ej. 4.0, 0.25)
```

Esto permite generar mensajes de feedback precisos para el usuario: *"¡Es súper efectivo!"* (x2, x4) o *"No es muy efectivo..."* (x0.5, x0.25).



6. Gestión de Estados Alterados

El BatallaService gestiona la lógica de estados persistentes (almacenados en el ENUM Estado de la entidad) y volátiles.

6.1. Bloqueo de Turnos (Pre-Turno)

Antes de permitir un ataque, se verifica el estado del Pokémon:

- **Congelado (FRZ)**: Tiene un 10% de probabilidad de descongelarse en cada turno. Si falla, el turno se pierde.
- **Dormido (SLP):** Utiliza el campo turnosSueno de la entidad PokemonUsuario. El contador decrece en cada turno hasta llegar a 0.
- **Paralizado (PAR):** Existe un 25% de probabilidad de sufrir parálisis total ("está totalmente paralizado"), perdiendo el turno.
- **Confusión:** Se gestiona mediante el campo turnosConfusion. Si el Pokémon está confuso, hay un 50% de probabilidad de que se ataque a sí mismo.
    - *Implementación:* Se calcula un daño a sí mismo con Potencia 40 (sin tipo) usando el CalculoService.

6.2. Daño Residual (Post-Turno)

Al finalizar el turno, el método aplicarEfectosPostTurno calcula el desgaste:

- **Veneno / Quemadura:** Restan 1/8 del HP máximo.
- **Intoxicación Grave (Toxic):** Usa el campo contadorToxico para aplicar daño exponencial:

$$Daño = \frac{HPmax \; x \; Contador}{16}$$

- **Drenadoras:** Si el flag tieneDrenadoras es true, aplica daño residual adicional.

7. Modelado de Datos (PokemonUsuario)

Para soportar esta lógica, la entidad PokemonUsuario ha sido diseñada con campos específicos para el combate, separando la información de la especie (PokedexMaestra) de la instancia concreta del jugador.

```
@Entity
@Table(name = "POKEMON_USUARIO")
public class PokemonUsuario {
    // ... IDs y Relaciones

    // Stats Dinámicos
    private Integer hpActual;
    private Integer hpMax;

    // Estados Persistentes
    @Enumerated(EnumType.STRING)
    private Estado estado; // SALUDABLE, QUEMADO, DORMIDO...

    // Contadores Volátiles (Para lógica compleja)
    private Integer turnosConfusion;
    private Integer contadorToxico;
    private Integer turnosSueno;
    private Boolean tieneDrenadoras;

    // ... Getters y Setters
}
```