**1. PlayerControllerEffect**

* **Responsabilidad:** Mantener el estado de mutaciones activas para el jugador.
* Contiene **flags** como MicrowavesMajorActive y MicrowavesMinorActive.
* También guarda los valores de cada mutación (burnDuration, damagePerTick, bonusDamageIfAlreadyBurned) que serán usados para aplicar DoT a los enemigos.
* Método relevante:

public void SetMicrowavesMajor(bool active, float burnDuration, float damagePerTick, float bonusDamage) { ... }

public void SetMicrowavesMinor(bool active, float minBurn, float maxBurn, float damagePerTick) { ... }

* Estos métodos se llaman desde los ScriptableObjects de las mutaciones para **activar la mutación** y almacenar sus parámetros.

**2. MicrowavesNervousMajorEffect (Scriptable)**

* **Responsabilidad:** Define los parámetros de la mutación mayor y activa su flag en el PlayerControllerEffect.
* **No aplica DoT directamente**. Solo activa el flag:

controller.SetMicrowavesMajor(true, burnDuration, damagePerTick, bonusDamageIfAlreadyBurned);

* Cuando se activa (por ejemplo con la tecla M en el tester), queda registrado en PlayerControllerEffect.

**3. MicrowavesNervousMinorEffect (Scriptable)**

* Similar al mayor, pero para la mutación menor.
* También **activa el flag** en PlayerControllerEffect y guarda sus parámetros (minBurnDuration, maxBurnDuration, damagePerTick).
* Los valores de duración se seleccionan aleatoriamente entre minBurnDuration y maxBurnDuration.

**4. Bullet**

* **Responsabilidad:** Detectar impactos contra enemigos.
* En su OnTriggerEnter o OnCollisionEnter:

var collectable = other.GetComponent<IProjectileCollectable>();

if (collectable != null)

collectable.OnHitByProjectile(playerControllerEffect);

* Pasa la referencia de PlayerControllerEffect para que el enemigo sepa qué mutaciones están activas y pueda aplicar DoT según los flags.

**5. IProjectileCollectable**

* **Interfaz** que implementan objetos que pueden recibir impactos de proyectiles (ej. enemigos).
* Contiene:

void OnHitByProjectile(PlayerControllerEffect playerEffect);

* Permite que cualquier objeto que reciba un disparo pueda reaccionar al mismo sin depender de su tipo concreto.

**6. EnemyStatusHandler**

* **Responsabilidad:** Manejar los efectos de estado en el enemigo (DoT, quemaduras, etc.).
* Implementa IStatusAffectable y IProjectileCollectable.
* Métodos importantes:

public void ApplyStatusEffect(StatusEffect newEffect) { ... }

public bool HasStatusEffect<T>() { ... }

public void OnHitByProjectile(PlayerControllerEffect playerEffect)

{

if (playerEffect.MicrowavesMajorActive) {

// Aplica BurnEffect mayor y bonus si ya estaba quemado

}

if (playerEffect.MicrowavesMinorActive) {

// Aplica BurnEffect menor

}

}

* **Logica DoT:**
  + Comprueba los flags del PlayerControllerEffect.
  + Aplica BurnEffect según corresponda.
  + Evita duplicar DoT del mismo tipo usando HasStatusEffect.

**7. StatusEffect**

* Clase abstracta base para todos los efectos de estado.
* Contiene:
  + Duration → cuánto dura el efecto
  + TickInterval → cada cuánto aplica daño
  + Initialize(IDamageable damageable) → referencia al enemigo para aplicar daño
  + Update(deltaTime) → avanza el tiempo y ejecuta OnTick()
  + IsSameType(StatusEffect other) → permite evitar stacking de efectos del mismo tipo

**8. BurnEffect (hereda de StatusEffect)**

* **Responsabilidad:** Aplica daño por segundo (DoT) al enemigo.
* Sobrescribe OnTick() para hacer:

\_damageable?.TakeDamage(damagePerTick);

* Puede tener lógica adicional como reducción gradual, visuales, etc.

**9. Flujo completo**

1. **Mutación activa:**
   * El jugador activa la mutación mayor o menor (p.ej. con M en el tester) → Scriptable llama SetMicrowavesMajor/Minor en PlayerControllerEffect.
2. **Disparo:**
   * El jugador dispara un Bullet.
   * Bullet detecta OnTriggerEnter con el enemigo.
   * Obtiene IProjectileCollectable y llama OnHitByProjectile(playerControllerEffect).
3. **Aplicación de DoT:**
   * EnemyStatusHandler revisa los flags de playerControllerEffect.
   * Si MicrowavesMajorActive → Aplica BurnEffect mayor + daño extra si ya estaba quemado.
   * Si MicrowavesMinorActive → Aplica BurnEffect menor.
4. **Actualización por frame:**
   * Cada Update() de EnemyStatusHandler llama a Update(deltaTime) de cada StatusEffect.
   * BurnEffect aplica daño periódicamente al enemigo.

✅ **Notas importantes:**

* **No hay stack automático:** Si un enemigo ya tiene el mismo BurnEffect, se reinicia el contador en ApplyStatusEffect.
* **Flags del PlayerControllerEffect** permiten manejar el DoT solo cuando la mutación está activa, **no al presionar M directamente**.
* **Minor y Major pueden coexistir**, creando **dos DoTs independientes** si ambos están activos.

DIAGRAMA

[PlayerControllerEffect]

│

│ (Activación con M o inventario)

▼

+----------------------+

| Flags de Mutaciones |

| MicrowavesMajorActive|

| MicrowavesMinorActive|

| burnDuration / damage|

+----------------------+

│

│ (Jugador dispara)

▼

[BULLET] ----------------------+

│ │

│ OnTriggerEnter │

▼ │

[IProjectileCollectable] │

EnemyStatusHandler implements │

│ │

│ OnHitByProjectile(PlayerControllerEffect)

▼

+-------------------------------+

| EnemyStatusHandler |

| - Comprueba flags de Player |

| - MicrowavesMajorActive → |

| Aplica BurnEffect Mayor |

| + Bonus Damage si ya |

| estaba quemado |

| - MicrowavesMinorActive → |

| Aplica BurnEffect Menor |

+-------------------------------+

│

│ Update() por frame

▼

+-------------------------------+

| StatusEffect (abstract) |

| - Duration, TickInterval |

| - Initialize(IDamageable) |

| - Update(deltaTime) |

| - OnTick() |

+-------------------------------+

│

│

▼

+-------------------------------+

| BurnEffect : StatusEffect |

| - Aplica daño periódico |

| al IDamageable |

+-------------------------------+

│

│

▼

[IDamageable] → enemigo recibe daño

**Explicación de flujo**

1. **Activación de mutación:**
   * PlayerControllerEffect guarda flags y parámetros de Microwaves.
   * Esto se hace mediante SetMicrowavesMajor o SetMicrowavesMinor, que activa la mutación pero **no aplica DoT directamente**.
2. **Disparo:**
   * Bullet detecta colisión con el enemigo.
   * Llama OnHitByProjectile(PlayerControllerEffect) del enemigo.
3. **Aplicación de DoT:**
   * EnemyStatusHandler verifica flags de PlayerControllerEffect.
   * Aplica BurnEffect mayor o menor según corresponda.
4. **Actualización de efecto:**
   * Cada frame, EnemyStatusHandler.Update() llama StatusEffect.Update(deltaTime).
   * BurnEffect.OnTick() aplica daño periódico al enemigo.
5. **Bonus damage:**
   * Solo se aplica si la mutación mayor ya estaba activa y el enemigo ya tenía quemadura.