



## Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorios de docencia

# Laboratorio de Computación Salas A y B

*Profesor(a):* René Adrián Dávila Pérez

*Asignatura:* Programación Orientada a Objetos

*Grupo:* 7

*No de Práctica(s):* Proyecto 3

*Integrante(s):* 322044339

322094152

322114461

425093384

322150984

*No. de brigada:* 6

*Semestre:* 2026-1

*Fecha de entrega:* 1 de diciembre de 2025

*Observaciones:* \_\_\_\_\_

**CALIFICACIÓN:** \_\_\_\_\_

# Índice

<b>1. Introducción</b>	<b>3</b>
<b>2. Marco Teórico</b>	<b>4</b>
2.1. Paradigma Orientado a Objetos . . . . .	4
2.2. Encapsulamiento y Modularidad . . . . .	4
2.3. Cohesión, Acoplamiento y Polimorfismo . . . . .	4
2.4. Modelado UML . . . . .	4
2.5. Tipos de Datos y Control de Flujo . . . . .	4
2.6. Herencia, Enumeraciones y Constructores . . . . .	5
2.7. Manejo de Excepciones . . . . .	5
2.8. Entrada y Salida de Datos . . . . .	5
2.9. Patrones de Diseño . . . . .	5
<b>3. Desarrollo</b>	<b>6</b>
<b>4. Integrante 1, 322150984</b>	<b>6</b>
4.1. Abstracción para representar el sistema . . . . .	6
4.2. Encapsulamiento y control de estado interno . . . . .	6
4.3. Modularidad y separación de responsabilidades . . . . .	6
4.4. Cohesión y acoplamiento en el diseño de clases . . . . .	7
4.5. Polimorfismo con enumeraciones y categorías . . . . .	7
4.6. Diseño estático y dinámico en UML . . . . .	7
4.7. Control de flujo y estructuras de decisión . . . . .	7
4.8. Manejo de excepciones . . . . .	8
4.9. Flujo de entrada/salida de datos persistente . . . . .	8
4.10. Patrón Singleton para control global . . . . .	8
<b>5. Integrante 2, 322044339</b>	<b>9</b>
5.1. Diagramas estáticos UI . . . . .	18
5.2. Diagramas estáticos UI . . . . .	19
<b>6. Integrante 3, 322114461</b>	<b>27</b>
<b>7. Integrante 4, 425093384</b>	<b>31</b>
7.1. Diseño de clases fundamentales y modelo de datos . . . . .	31
7.2. Implementación del prototipo en terminal . . . . .	31
7.3. Migración a un modelo basado en eventos . . . . .	31
7.4. Decisiones de diseño del modo de juego . . . . .	31
7.5. Diseño de la dificultad y sistema de ítems . . . . .	32
7.6. Ampliación del motor de batalla . . . . .	33
7.7. Ajustes y mejoras en fórmulas internas . . . . .	34
7.8. Integración de la primera interfaz gráfica (UI) . . . . .	34
7.9. Creación de widgets personalizados para el HUD . . . . .	34
7.10. Generación del roster dinámico mediante un Singleton . . . . .	34

7.11. Pantalla de selección de equipo . . . . .	34
7.12. Búsqueda y análisis de assets . . . . .	34
7.13. Desarrollo e integración del sistema de audio . . . . .	34
7.14. Menú de opciones para el control de audio . . . . .	35
7.15. Porteo a Android y ajustes multiplataforma . . . . .	35
7.16. Fase colaborativa de pruebas (testing) . . . . .	36
7.17. Parcheado y optimización . . . . .	36
7.18. Documentación técnica . . . . .	36
<b>8. Integrante 5, 322094152</b>	<b>37</b>
8.1. Polimorfismo en clases pokemon y ataques . . . . .	37
8.2. Batallas pokemon . . . . .	39
8.3. Portear a otros sistemas . . . . .	41
8.4. Importar audio . . . . .	43
8.5. Implementación de la mochila . . . . .	44
8.6. Estados de alteración . . . . .	45
<b>9. Resultados</b>	<b>48</b>
9.1. Pantalla inicial . . . . .	48
9.2. Opciones . . . . .	49
9.3. Nueva partida . . . . .	50
9.4. Selección de equipo . . . . .	51
9.5. Combates . . . . .	53
9.6. Pantallas finales . . . . .	60
<b>10. Conclusiones</b>	<b>62</b>

# 1. Introducción

En la ingeniería de software, los proyectos integradores son el eslabón entre la teoría y la práctica; son el lugar donde los conocimientos abstractos se transforman en algo tangible. Durante el semestre, la materia de Programación Orientada a Objetos nos dio muchas herramientas conceptuales y metodológicas para modelar problemas, estructurar sistemas y desarrollar aplicaciones robustas, modulares y sostenibles. Pero conocer estos contenidos aisladamente de poco sirve si no se integran en un proyecto real que requiera ponerlos en juego.

El Proyecto 3 nace con esa finalidad: integrar en una sola aplicación todos los conceptos principales vistos en el curso. Desde los conceptos elementales del paradigma orientado a objetos (abstracción, encapsulamiento, herencia, polimorfismo) hasta tópicos más avanzados como manejo de errores, control de flujo, diseño UML, entrada/salida, programación concurrente y una introducción a patrones de diseño. El desarrollo de un simulador de batallas como el que se propone en este trabajo es una oportunidad para reconocer cómo cada tema del curso se manifiesta en partes reales: clases que colaboran, estructuras que definen la lógica, patrones que simplifican el diseño, diagramas que revelan la arquitectura y mecanismos de control que hacen que el sistema reaccione a los eventos del usuario. Todo ello hace que el conjunto de conocimientos no se vea como piezas aisladas, sino como partes de un todo. El objetivo es que el estudiante logre reconocer estos conceptos en un contexto teórico, pero que también pueda aplicarlos coherentemente en la solución integral.

El desarrollo de un simulador de batallas como el que se propone en este trabajo es una oportunidad para reconocer cómo cada tema del curso se manifiesta en partes reales: clases que colaboran, estructuras que definen la lógica, patrones que simplifican el diseño, diagramas que revelan la arquitectura y mecanismos de control hacen que el sistema reaccione a los eventos del usuario. ello hace que el conjunto de conocimientos no se vea como piezas aisladas, sino partes de un todo.

De este modo, en este informe se aborda el desarrollo del proyecto desde un enfoque integrador, mostrando cómo los contenidos del curso se fueron integrando de manera orgánica en la construcción de la aplicación. De esta manera, más que el resultado final, este informe busca exponer el camino que permitió integrar los conocimientos funcionales en una única solución, un proyecto que representa el cierre del ciclo de aprendizaje a través de la práctica y que demuestra la habilidad de convertir teoría en software.

322044339

## **2. Marco Teórico**

### **2.1. Paradigma Orientado a Objetos**

El paradigma orientado a objetos es un modelo que organiza el software en términos de objetos, que encapsulan estado y comportamiento. Esta manera puede plasmar en una aplicación las cosas del mundo real y construir un diseño modular y escalable. Aquí la abstracción es aislar las propiedades esenciales de algo, dejando a un lado los detalles. En el proyecto, las clases como Pokemon, Movimiento e Item encapsulan la información de la mecánica de batalla sin preocuparse por detalles innecesarios. [7]

### **2.2. Encapsulamiento y Modularidad**

El encapsulamiento oculta la información interna de un objeto y proporciona formas de acceder a ella desde fuera. En el simulador, las clases gestionan su propio estado con métodos como recibirDaño o curar, sin poder manipular sus estadísticas internas. La modularidad organiza la aplicación en partes independientes que interactúan entre sí. En el proyecto, esta modularidad se evidencia en la estructura por carpetas: los modelos son entidades, los controladores son la lógica, la vista es la interfaz y los datos son catálogos y tablas. [11]

### **2.3. Cohesión, Acoplamiento y Polimorfismo**

La cohesión y el acoplamiento también son principios. La cohesión es una medida de cuán relacionadas están las funciones de una clase y el acoplamiento mide la dependencia entre clases. En el proyecto, los modelos solo guardan datos, el controlador de batalla maneja los turnos y la vista solo los muestra, disminuyendo las dependencias y generando código más limpio. El polimorfismo hace posible que tipos diferentes puedan ser tratados de forma indiferente bajo una misma interfaz. Aquí vemos cómo las enumeraciones (como CategoriaMovimiento) pueden definir comportamientos en métodos que aceptan cualquier categoría. [2]

### **2.4. Modelado UML**

El modelado en UML nos ayuda a representar la estructura y el comportamiento del sistema. Diseño estático: los diagramas de clases, mostrando cómo interactúan las clases (Pokemon, Movimiento, Item, ControlBatalla, GameManager, Pantallas). El patrón dinámico especifica la manera en que interactúan estas entidades en el tiempo (por ejemplo, el flujo de un turno de batalla donde la interfaz llama al controlador y éste le regresa eventos para animar). [4]

### **2.5. Tipos de Datos y Control de Flujo**

Los tipos de datos, expresiones y estructuras de control definen la lógica de la aplicación. En el simulador se usan condicionales para determinar el orden de ataque

y verificar el estado de los Pokémon, y bucles para iterar sobre listas de movimientos, eventos o equipos. Además, las listas y los diccionarios se usan para representar equipos, ataques e inventarios.

## **2.6. Herencia, Enumeraciones y Constructores**

El proyecto no usa herencia profunda, pero sí usa principios de diseño orientado a objetos. Las enumeraciones son una forma de enumerar conjuntos de valores relacionados, como EstadoAlterado, TipoItem o CategoriaMovimiento. Los constructores inicializan correctamente los objetos (por ejemplo, en Pokemon se calculan estadísticas y se asignan valores iniciales). [8]

## **2.7. Manejo de Excepciones**

El tratamiento de excepciones anticipa errores inesperados. En el proyecto se usa mayormente para archivos de audio, cuando abren archivos inexistentes o corruptos. El try-catch previene que la aplicación se detenga ante errores en recursos externos.

## **2.8. Entrada y Salida de Datos**

La entrada y salida de datos permite guardar y recuperar información de forma permanente. Esto se nota en cómo se usa SharedPreferences en GameManager para guardar el nombre, sprite, etc., para continuar donde lo dejó.

## **2.9. Patrones de Diseño**

En definitiva, los patrones de diseño son soluciones reutilizables a problemas comunes de arquitectura. En este proyecto se utiliza el patrón Singleton en clases como GameManager y AudioManager para asegurar que solo exista una instancia y así controlar el estado del juego y la música. [5]

322150984

### **3. Desarrollo**

#### **4. Integrante 1, 322150984**

El simulador de batallas fue un lugar donde los conceptos de la POO no solo se discutieron, sino que se codificaron. Todas las teorías aprendidas en clase se aplicaron en las decisiones de diseño para desarrollar una aplicación estructurada, modular, mantenable y consistente. A continuación, se describe cómo se aplicaron estos conceptos en el proyecto, sin mostrar código explícito, pero explicando el motivo de cada implementación en el sistema.

##### **4.1. Abstracción para representar el sistema**

Una de las cosas que más saltó a la vista del proyecto fue la abstracción. Para simular el mundo de batalla Pokémon solo se necesitó identificar lo que caracterizaba a cada entidad. Un Pokémon se caracteriza por sus estadísticas, movimientos, nivel, tipo y estado de salud. Un movimiento posee nombre, tipo elemental, categoría, poder, precisión y prioridad. Una cosa es lo que hace, la cantidad que se tiene y, si procede, su tipo de curación. Nunca se modelan detalles como rasgos biológicos, comportamientos fuera de combate o fragmentos de historia. Esta capacidad hizo posible generar pequeñas clases que sólo contienen lo necesario para la simulación y descartan el resto. La abstracción fue, entonces, una manera de delimitar el proyecto y no caer en complejidades innecesarias.

##### **4.2. Encapsulamiento y control de estado interno**

El encapsulamiento también fue clave para la calidad del diseño. Las clases base (en particular las clases Pokémon) tienen control absoluto sobre su propio estado interno. Un Pokémon no acepta ser manipulado en su vida, ataque o estado alterado. En cambio, proporciona maneras concretas de hacer daño, curarse o debilitarse. Asegura la integridad en las ecuaciones de combate: solo el método de combate puede cambiar los puntos de vida, solo los métodos internos pueden cambiar los estados alterados, y la lógica previene errores como curar sobre los puntos de vida máximos o infligir daño negativo. Gracias al encapsulamiento, el proyecto separa la lógica interna de los modelos del resto de la aplicación, evitando errores y garantizando la fiabilidad. [3]

##### **4.3. Modularidad y separación de responsabilidades**

El proyecto se organizó en módulos para facilitar su mantenimiento. Cada carpeta tiene una función: models son las clases entidad del juego, controllers es la lógica de batalla y control de flujo, ui son las pantallas, animaciones, widgets y la entrada del usuario, data son catálogos de Pokémon, movimientos, objetos y tablas de efectividad, y audio es la música y los efectos de sonido. Esta separación permite manipular cada parte sin afectar las demás. Por ejemplo, cambiar una animación de la interfaz no cambia la manera en que se calcula el daño, añadir un nuevo Pokémon no implica

cambiar las pantallas ni el controlador de batalla. La modularidad hace del proyecto un sistema resistente donde cada parte es independiente y reemplazable. [6]

#### **4.4. Cohesión y acoplamiento en el diseño de clases**

El diseño resultó muy cohesionado en cada clase; cada una es responsable de una tarea. La clase de un Pokémon solo controla sus estadísticas, estado y movimientos. El controlador de batalla solo se encarga del combate: turnos, cálculo de daño, efectos, KO, eventos, flujo. Las pantallas solo informan, no calculan el combate. Esto reduce el acoplamiento porque cada clase solo conoce la información que necesita. El controlador no sabe nada de la interfaz, solo devuelve eventos. La interfaz no hiera, solo refleja. Esta separación fomentó la claridad, el soporte y la facilidad de extensión con nuevas funcionalidades sin afectar el resto del sistema.

#### **4.5. Polimorfismo con enumeraciones y categorías**

El proyecto aplica el polimorfismo con estructuras como las enumeraciones. Por ejemplo, la categoría de un movimiento influye en el cálculo de daño, ya que un movimiento físico usa estadísticas diferentes a uno especial. No tiene subclases, pero el juego permite que en un mismo método se puedan hacer movimientos de distintos tipos y que los cambios de estado influyan de manera diferente en el Pokémon según su tipo. El polimorfismo hace que las cosas se comporten de forma particular en un flujo genérico. [9]

#### **4.6. Diseño estático y dinámico en UML**

UML guio la comprensión y planificación del sistema. El diseño estático, en un diagrama de clases, modeló las principales entidades, sus atributos y relaciones. La dependencia de Pokémon a sus movimientos, la dependencia entre entrenadores y equipos, y la dependencia entre vista y controlador se muestran en el diagrama final. En el diagrama de flujo se puede apreciar cómo transcurre un turno en la batalla, desde que el jugador selecciona una acción hasta las animaciones de daño. UML hizo posible visualizar la estructura, asignar responsabilidades y verificar que el flujo tuviera sentido antes de escribir una línea de código.

#### **4.7. Control de flujo y estructuras de decisión**

Cómo se lleva a cabo el combate lo controlan estructuras de control. El sistema determina qué Pokémon ataca primero (si el ataque acierta o falla, si algún estado impide atacar, si el daño derrotó al oponente y si debe cambiar automáticamente a otro Pokémon). Condicionales, chequeos por prioridades, comparaciones de velocidad y bifurcaciones por estados alterados son la carne de la simulación. También bucles pueden recorrer la lista de movimientos, los eventos de cada turno o los Pokémon de un equipo para hallar el siguiente disponible. Esta lógica controlada asegura que cada turno siga las reglas del sistema de combate. [1]

## **4.8. Manejo de excepciones**

El proyecto cuenta con control de errores sobre todo en la parte multimedia. Cuando se reproduce música o audio, puede faltar un archivo o estar corrupto. Para ello se utilizan capturadores de errores que impiden que la aplicación se cierre. Si algo falla, la aplicación sigue funcionando, mejorando la estabilidad y la experiencia del usuario.

## **4.9. Flujo de entrada/salida de datos persistente**

La aplicación hace uso de almacenamiento persistente para guardar datos entre sesiones, como por ejemplo el nombre del jugador, el sprite escogido o la dificultad. Esto se hace con shared preferences para guardar y cargar datos simples y que la aplicación recuerde en qué punto se quedó una partida sin tener que volver a empezar. Esta aplicación práctica de entrada-salida muestra cómo la teoría se aplica en una aplicación móvil real. [10]

## **4.10. Patrón Singleton para control global**

Finalmente, el patrón Singleton se aplicó en dos partes principales: GameManager para la información global del juego y AudioManager para música y efectos de sonido en toda la aplicación. Ambos necesitan una única instancia activa a la que se puede acceder desde cualquier punto del sistema. Gracias al Singleton, asegura la consistencia, evita la duplicación de datos y centraliza las operaciones.

## 5. Integrante 2, 322044339

### Diagramas UML estático (de clases)

Pokemon
<pre>+ nombre: String + tipoPrimario: String + baseHP: int + baseAtk: int + baseDef: int + baseSpAtk: int + baseSpDef: int + baseSpeed: int + nivel: int + movimientos: List&lt;Movimiento&gt; + ivHP: int + ivAtk: int + ivDef: int + ivSpAtk: int + ivSpDef: int + ivSpeed: int + hpMaximo: int + ataque: int + defensa: int + ataqueEspecial: int + defensaEspecial: int + velocidad: int + hpActual: int - _rand: Random + estado: EstadoAlterado + turnosCongelado: int  + estaQuemado(): bool + estaParalizado(): bool + estaQuemado(): bool + estaEnvenenado(): bool + Pokemon(nombre: String, tipoPrimario: String, baseHP: int, baseAtk: int, baseDef: int, baseSpAtk: int, baseSpDef: int, baseSpeed: int, nivel: int, movimientos: List&lt;Movimiento&gt;) - _calcularHP(base: int, iv: int): int - _calcularStat(base: int, iv:int): int + proporcionHP(): double + estaDebilitado(): bool + recibirDanio(danio: int): void + curar(cantidad: int): void + curarAlMaximo(): void + toString(): String</pre>

Figura 1: Diagramas estáticos (de clases)

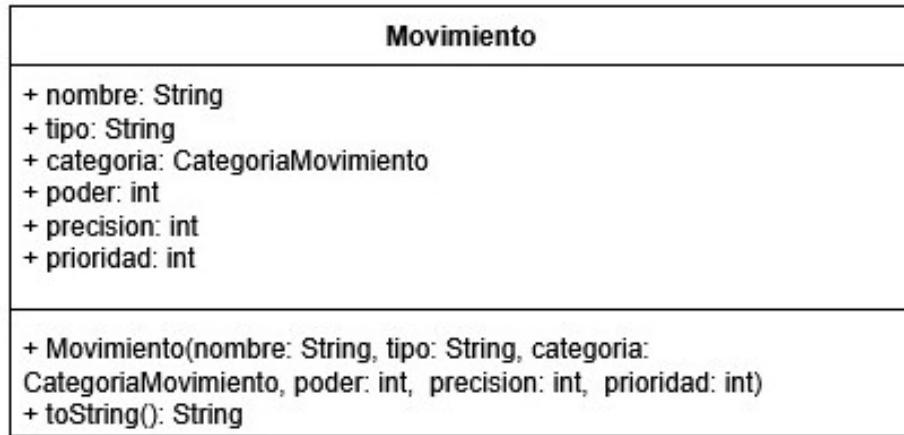


Figura 2: Diagramas estáticos (de clases)

Pikachu: Pokemon	Charmander: Pokemon	Garchomp: Pokemon	Salamence: Pokemon	Scopyle: Pokemon
<pre>+ nombre = "Pikachu" + tipoPrimario = "Eléctrico" + baseHP = 35 + baseAtk = 55 + baseDef = 40 + baseSpAtk = 50 + baseSpDef = 50 + baseSpeed = 90 + movimientos = [rayo, ondaTrueno, ondaElectrica]</pre>	<pre>+ nombre = "Charmander" + tipoPrimario = "Fuego" + baseHP = 39 + baseAtk = 52 + baseDef = 43 + baseSpAtk = 60 + baseSpDef = 50 + baseSpeed = 65 + movimientos = [lanzallamas, asciusEstado, ataqueRapido]</pre>	<pre>+ nombre = "Garchomp" + tipoPrimario = "Dragón" + baseHP = 108 + baseAtk = 130 + baseDef = 95 + baseSpAtk = 80 + baseSpDef = 85 + baseSpeed = 102 + movimientos = [garrDragon, terremoto, ataqueRapido]</pre>	<pre>+ nombre = "Salamence" + tipoPrimario = "Dragón" + baseHP = 70 + baseAtk = 135 + baseDef = 85 + baseSpAtk = 110 + baseSpDef = 80 + baseSpeed = 100 + movimientos = [garrDragon, lanzallamas, ataqueRapido]</pre>	<pre>+ nombre = "Scopyle" + tipoPrimario = "Planta" + baseHP = 70 + baseAtk = 85 + baseDef = 65 + baseSpAtk = 105 + baseSpDef = 85 + baseSpeed = 120 + movimientos = [hojaAguda, latigoCepa, ataqueRapido]</pre>
Bulbasur: Pokemon	Squirtle: Pokemon	Tyrantar: Pokemon	Dragonite: Pokemon	Aggron: Pokemon
<pre>+ nombre = "Bulbasur" + tipoPrimario = "Planta" + baseHP = 45 + baseAtk = 49 + baseDef = 49 + baseSpAtk = 65 + baseSpDef = 65 + baseSpeed = 45 + movimientos = [latigoCepa, hojaAtacada, polvoVeneno]</pre>	<pre>+ nombre = "Squirtle" + tipoPrimario = "Aqua" + baseHP = 44 + baseAtk = 48 + baseDef = 65 + baseSpAtk = 50 + baseSpDef = 64 + baseSpeed = 43 + movimientos = [pistolaAgua, surt, congelar]</pre>	<pre>+ nombre = "Tyrantar" + tipoPrimario = "Roca" + baseHP = 100 + baseAtk = 134 + baseDef = 110 + baseSpAtk = 95 + baseSpDef = 100 + baseSpeed = 61 + movimientos = [garrMetal, terremoto, punioHelio]</pre>	<pre>+ nombre = "Dragonite" + tipoPrimario = "Dragón" + baseHP = 91 + baseAtk = 134 + baseDef = 95 + baseSpAtk = 100 + baseSpDef = 100 + baseSpeed = 80 + movimientos = [garrDragon, punioFuego, ondaTrueno]</pre>	<pre>+ nombre = "Aggron" + tipoPrimario = "Acero" + baseHP = 70 + baseAtk = 110 + baseDef = 180 + baseSpAtk = 60 + baseSpDef = 60 + baseSpeed = 50 + movimientos = [garrMetal, terremoto, ataqueRapido]</pre>
Gengar: Pokemon	Alakazam: Pokemon	Metagross: Pokemon	Godra: Pokemon	Abro: Pokemon
<pre>+ nombre = "Gengar" + tipoPrimario = "Fantasma" + baseHP = 60 + baseAtk = 65 + baseDef = 60 + baseSpAtk = 130 + baseSpDef = 75 + baseSpeed = 110 + movimientos = [bolaSombra, psiquico, rayo]</pre>	<pre>+ nombre = "Alakazam" + tipoPrimario = "Psiquico" + baseHP = 55 + baseAtk = 50 + baseDef = 45 + baseSpAtk = 135 + baseSpDef = 95 + baseSpeed = 120 + movimientos = [psiquico, bolaSombra, rayo]</pre>	<pre>+ nombre = "Metagross" + tipoPrimario = "Acero" + baseHP = 80 + baseAtk = 135 + baseDef = 130 + baseSpAtk = 95 + baseSpDef = 90 + baseSpeed = 70 + movimientos = [garrMetal, psiquico, terremoto]</pre>	<pre>+ nombre = "Godra" + tipoPrimario = "Dragón" + baseHP = 90 + baseAtk = 100 + baseDef = 70 + baseSpAtk = 110 + baseSpDef = 150 + baseSpeed = 80 + movimientos = [garrDragon, poderOcular, ataqueRapido]</pre>	<pre>+ nombre = "Abro" + tipoPrimario = "Siniestro" + baseHP = 65 + baseAtk = 130 + baseDef = 60 + baseSpAtk = 75 + baseSpDef = 60 + baseSpeed = 75 + movimientos = [bolaSombra, ataqueRapido, garrMetal]</pre>

Figura 3: Diagramas estáticos (de clases)

Sceptile: Pokemon	Arcanine: Pokemon	Breloom: Pokemon	Swampert: Pokemon	Gigalith: Pokemon
+ nombre = "Sceptile" + tipoPrimario = "Planta" + baseHP = 70 + baseAtk = 85 + baseDef = 65 + baseSpAtk = 105 + baseSpDef = 85 + baseSpeed = 120 + movimientos = [hojaAguda, latigoCepa, ataqueRapido]	+ nombre = "Arcanine" + tipoPrimario = "Fuego" + baseHP = 90 + baseAtk = 110 + baseDef = 80 + baseSpAtk = 100 + baseSpDef = 60 + baseSpeed = 95 + movimientos = [lanzallamas, ataqueRapido, punioFuego]	+ nombre = "Breloom" + tipoPrimario = "Planta" + baseHP = 60 + baseAtk = 130 + baseDef = 80 + baseSpAtk = 60 + baseSpDef = 60 + baseSpeed = 70 + movimientos = [latigoCepa, hojaAfiada, ataqueRapido]	+ nombre = "Swampert" + tipoPrimario = "Aqua" + baseHP = 100 + baseAtk = 110 + baseDef = 90 + baseSpAtk = 85 + baseSpDef = 90 + baseSpeed = 60 + movimientos = [terremoto, surf, ataqueRapido]	+ nombre = "Gigalith" + tipoPrimario = "Roca" + baseHP = 85 + baseAtk = 135 + baseDef = 130 + baseSpAtk = 60 + baseSpDef = 80 + baseSpeed = 25 + movimientos = [avalancha, terremoto, garraMetal]
Aggron: Pokemon	Vaporeon: Pokemon	Magnezone: Pokemon	Tokekiss: Pokemon	Sylveon: Pokemon
+ nombre = "Aggron" + tipoPrimario = "Acero" + baseHP = 70 + baseAtk = 110 + baseDef = 180 + baseSpAtk = 60 + baseSpDef = 60 + baseSpeed = 50 + movimientos = [garrametal, terremoto, ataqueRapido]	+ nombre = "Vaporeon" + tipoPrimario = "Aqua" + baseHP = 130 + baseAtk = 65 + baseDef = 60 + baseSpAtk = 110 + baseSpDef = 90 + baseSpeed = 65 + movimientos = [surf, pistolaAgua, congelar]	+ nombre = "Magnezone" + tipoPrimario = "Eléctrico" + baseHP = 70 + baseAtk = 70 + baseDef = 115 + baseSpAtk = 130 + baseSpDef = 90 + baseSpeed = 60 + movimientos = [rayo, ondaElectrica, poderOculto]	+ nombre = "Tokekiss" + tipoPrimario = "Hada" + baseHP = 85 + baseAtk = 50 + baseDef = 95 + baseSpAtk = 120 + baseSpDef = 115 + baseSpeed = 80 + movimientos = [poderOculto, psiquico, ataqueRapido]	+ nombre = "Sylveon" + tipoPrimario = "Hada" + baseHP = 95 + baseAtk = 65 + baseDef = 65 + baseSpAtk = 110 + baseSpDef = 130 + baseSpeed = 60 + movimientos = [brilloMagico, psiquico, ataqueRapido]
Absol: Pokemon	Dorphan: Pokemon	Umbreon: Pokemon	Ferrothorn: Pokemon	Weavile: Pokemon
+ nombre = "Absol" + tipoPrimario = "Siniestro" + baseHP = 65 + baseAtk = 130 + baseDef = 60 + baseSpAtk = 75 + baseSpDef = 60 + baseSpeed = 75 + movimientos = [bolasSombra, ataqueRapido, garrameta]	+ nombre = "Dorphan" + tipoPrimario = "Tierra" + baseHP = 90 + baseAtk = 120 + baseDef = 120 + baseSpAtk = 60 + baseSpDef = 60 + baseSpeed = 50 + movimientos = [lanzallamas, ascuesEstado, ataqueRapido]	+ nombre = "Umbreon" + tipoPrimario = "Siniestro" + baseHP = 95 + baseAtk = 65 + baseDef = 110 + baseSpAtk = 60 + baseSpDef = 130 + baseSpeed = 65 + movimientos = [bolasSombra, ataqueRapido, psiquico]	+ nombre = "Ferrothorn" + tipoPrimario = "Planta" + baseHP = 74 + baseAtk = 94 + baseDef = 131 + baseSpAtk = 54 + baseSpDef = 116 + baseSpeed = 20 + movimientos = [bolaSombra, ataqueRapido, psiquico]	+ nombre = "Weavile" + tipoPrimario = "Hielo" + baseHP = 70 + baseAtk = 120 + baseDef = 65 + baseSpAtk = 45 + baseSpDef = 85 + baseSpeed = 125 + movimientos = [punioHielo, pulsoUmbral, garrameta]

Figura 4: Diagramas estáticos (de clases)

Infernape: Pokemon	Gardevoir: Pokemon	Venusaur: Pokemon	Typhlosion: Pokemon
+ nombre = "Infernape" + tipoPrimario = "Fuego" + baseHP = 76 + baseAtk = 104 + baseDef = 71 + baseSpAtk = 104 + baseSpDef = 71 + baseSpeed = 108 + movimientos = [lanzallamas, ataqueRapido, punioFuego]	+ nombre = "Gardevoir" + tipoPrimario = "Psiquico" + baseHP = 65 + baseAtk = 65 + baseDef = 65 + baseSpAtk = 125 + baseSpDef = 115 + baseSpeed = 80 + movimientos = [psiquico, bolaSombra, poderOculto]	+ nombre = "Venusaur" + tipoPrimario = "Planta" + baseHP = 80 + baseAtk = 82 + baseDef = 83 + baseSpAtk = 100 + baseSpDef = 100 + baseSpeed = 80 + movimientos = [hojaAfiada, latigoCepa, polvoVeneno]	+ nombre = "Typhlosion" + tipoPrimario = "Fuego" + baseHP = 78 + baseAtk = 84 + baseDef = 78 + baseSpAtk = 109 + baseSpDef = 85 + baseSpeed = 100 + movimientos = [lanzallamas, ataqueRapido, ascuesEstado]
Gliscor: Pokemon	Snorlax: Pokemon	Blastoise: Pokemon	Feraligatr: Pokemon
+ nombre = "Gliscor" + tipoPrimario = "Tierra" + baseHP = 75 + baseAtk = 95 + baseDef = 125 + baseSpAtk = 45 + baseSpDef = 75 + baseSpeed = 95 + movimientos = [terremoto, garrameta, ataqueRapido]	+ nombre = "Snorlax" + tipoPrimario = "Normal" + baseHP = 160 + baseAtk = 110 + baseDef = 65 + baseSpAtk = 65 + baseSpDef = 110 + baseSpeed = 30 + movimientos = [plajaje, ataqueRapido, poderOculto]	+ nombre = "Blastoise" + tipoPrimario = "Aqua" + baseHP = 79 + baseAtk = 83 + baseDef = 100 + baseSpAtk = 85 + baseSpDef = 105 + baseSpeed = 78 + movimientos = [surf, congelar, ataqueRapido]	+ nombre = "Feraligatr" + tipoPrimario = "Aqua" + baseHP = 85 + baseAtk = 105 + baseDef = 100 + baseSpAtk = 79 + baseSpDef = 83 + baseSpeed = 78 + movimientos = [surf, ataqueRapido, garraDragon]
Lucario: Pokemon	Greninja: Pokemon	Espeon: Pokemon	Meganium: Pokemon
+ nombre = "Lucario" + tipoPrimario = "Lucha" + baseHP = 70 + baseAtk = 110 + baseDef = 70 + baseSpAtk = 115 + baseSpDef = 70 + baseSpeed = 90 + movimientos = [garrameta, psiquico, ataqueRapido]	+ nombre = "Greninja" + tipoPrimario = "Aqua" + baseHP = 72 + baseAtk = 95 + baseDef = 67 + baseSpAtk = 103 + baseSpDef = 71 + baseSpeed = 122 + movimientos = [surf, psiquico, ataqueRapido]	+ nombre = "Espeon" + tipoPrimario = "Psiquico" + baseHP = 65 + baseAtk = 65 + baseDef = 60 + baseSpAtk = 130 + baseSpDef = 95 + baseSpeed = 110 + movimientos = [psiquico, bolaSombra, ataqueRapido]	+ nombre = "Meganium" + tipoPrimario = "Planta" + baseHP = 80 + baseAtk = 82 + baseDef = 100 + baseSpAtk = 83 + baseSpDef = 100 + baseSpeed = 80 + movimientos = [latigoCepa, hojaAfiada, toxico]

Figura 5: Diagramas estáticos (de clases)

Ampharos: Pokemon	Empoleon: Pokemon	Charizard: Pokemon	Moltac: Pokemon	Crobat: Pokemon
+ nombre = "Ampharos" + tipoPrimario = "Eléctrico" + baseHP = 90 + baseAtk = 75 + baseDef = 85 + baseSpAtk = 115 + baseSpDef = 90 + baseSpeed = 55 + movimientos = [rayo, ondaElectrica, ataqueRapido]	+ nombre = "Emeleon" + tipoPrimario = "Aqua" + baseHP = 84 + baseAtk = 86 + baseDef = 88 + baseSpAtk = 111 + baseSpDef = 101 + baseSpeed = 60 + movimientos = [surf, garrameta, congelar]	+ nombre = "Charizard" + tipoPrimario = "Fuego" + baseHP = 78 + baseAtk = 84 + baseDef = 78 + baseSpAtk = 109 + baseSpDef = 85 + baseSpeed = 100 + movimientos = [lanzallamas, garrDragon, ataqueRapido]	+ nombre = "Moltac" + tipoPrimario = "Aqua" + baseHP = 95 + baseAtk = 60 + baseDef = 79 + baseSpAtk = 100 + baseSpDef = 125 + baseSpeed = 81 + movimientos = [surf, brillMagic, congelar]	+ nombre = "Crobat" + tipoPrimario = "Veneno" + baseHP = 85 + baseAtk = 90 + baseDef = 80 + baseSpAtk = 70 + baseSpDef = 80 + baseSpeed = 130 + movimientos = [bombaLodo, ataqueAereo, zumbido]
Heracross: Pokemon	Luxray: Pokemon	Hydrogren: Pokemon	Sizzler: Pokemon	Ninetales: Pokemon
+ nombre = "Heracross" + tipoPrimario = "Lucha" + baseHP = 80 + baseAtk = 125 + baseDef = 75 + baseSpAtk = 40 + baseSpDef = 95 + baseSpeed = 85 + movimientos = [garrMetal, ataqueRapido, latigoCepa]	+ nombre = "Luxray" + tipoPrimario = "Eléctrico" + baseHP = 80 + baseAtk = 120 + baseDef = 79 + baseSpAtk = 95 + baseSpDef = 79 + baseSpeed = 70 + movimientos = [rayo, ataqueRapido, poderOculto]	+ nombre = "Hydrogren" + tipoPrimario = "Siniestro" + baseHP = 92 + baseAtk = 105 + baseDef = 90 + baseSpAtk = 125 + baseSpDef = 90 + baseSpeed = 98 + movimientos = [bolaSombra, psiquico, ataqueRapido]	+ nombre = "Sizzler" + tipoPrimario = "Bicho" + baseHP = 73 + baseAtk = 130 + baseDef = 100 + baseSpAtk = 55 + baseSpDef = 80 + baseSpeed = 65 + movimientos = [jumblido, paraMetal, golpeCruzado]	+ nombre = "Ninetales" + tipoPrimario = "Fuego" + baseHP = 76 + baseAtk = 75 + baseDef = 81 + baseSpAtk = 100 + baseSpDef = 100 + baseSpeed = 100 + movimientos = [lanzallamas, asciuaEstado, psiquico]
Torterra: Pokemon	Roserade: Pokemon	Spiritomb: Pokemon	Staraptor: Pokemon	
+ nombre = "Torterra" + tipoPrimario = "Planta" + baseHP = 95 + baseAtk = 109 + baseDef = 105 + baseSpAtk = 75 + baseSpDef = 85 + baseSpeed = 56 + movimientos = [latigoCepa, hojaAlada, terremoto]	+ nombre = "Roserade" + tipoPrimario = "Planta" + baseHP = 60 + baseAtk = 70 + baseDef = 65 + baseSpAtk = 125 + baseSpDef = 105 + baseSpeed = 90 + movimientos = [hojaAlada, latigoCepa, toxico]	+ nombre = "Spiritomb" + tipoPrimario = "Fantasma" + baseHP = 50 + baseAtk = 92 + baseDef = 108 + baseSpAtk = 92 + baseSpDef = 108 + baseSpeed = 35 + movimientos = [bolaSombra, pulsoUmbrío, ataqueRapido]	+ nombre = "Staraptor" + tipoPrimario = "Volador" + baseHP = 85 + baseAtk = 120 + baseDef = 70 + baseSpAtk = 50 + baseSpDef = 60 + baseSpeed = 100 + movimientos = [ataqueAereo, hiperColmillo, golpeCruzado]	

Figura 6: Diagramas estáticos (de clases)

ondaTrueno: Movimiento	hojaAlada: Movimiento	asciuaEstado: Movimiento	puntoTrueno: Movimiento	golpeCruzado: Movimiento
+ nombre = "Onda Trueno" + tipo = "Eléctrico" + categoria = CategoríaMovimiento estado + poder = 0 + precision = 90 + prioridad = 0	+ nombre = "Hoja Alada" + tipo = "Planta" + categoria = CategoríaMovimiento fisico + poder = 45 + precision = 95 + prioridad = 0	+ nombre = "Quemadura" + tipo = "Fuego" + categoria = CategoríaMovimiento estado + poder = 55 + precision = 85 + prioridad = 0	+ nombre = "Punto Trueno" + tipo = "Eléctrico" + categoria = CategoríaMovimiento fisico + poder = 75 + precision = 100 + prioridad = 0	+ nombre = "Golpe Cruzado" + tipo = "Lucha" + categoria = CategoríaMovimiento fisico + poder = 100 + precision = 80 + prioridad = 0
ondaElectrica: Movimiento	pistolaAgua: Movimiento	congelar: Movimiento	poderOscuro: Movimiento	avalancha: Movimiento
+ nombre = "Onda Eléctrica" + tipo = "Eléctrico" + categoria = CategoríaMovimiento estado + poder = 0 + precision = 95 + prioridad = 0	+ nombre = "Pistola Agua" + tipo = "Aqua" + categoria = CategoríaMovimiento especial + poder = 40 + precision = 100 + prioridad = 0	+ nombre = "Congelar" + tipo = "Hielo" + categoria = CategoríaMovimiento estado + poder = 0 + precision = 90 + prioridad = 0	+ nombre = "Poder Oscuro" + tipo = "Normal" + categoria = CategoríaMovimiento especial + poder = 60 + precision = 100 + prioridad = 0	+ nombre = "Avalancha" + tipo = "Roca" + categoria = CategoríaMovimiento fisico + poder = 75 + precision = 90 + prioridad = 0
asciua: Movimiento	surf: Movimiento	colaDragon: Movimiento	pulsoUmbrío: Movimiento	belloMagico: Movimiento
+ nombre = "Asciua" + tipo = "Fuego" + categoria = CategoríaMovimiento especial + poder = 40 + precision = 100 + prioridad = 0	+ nombre = "Surf" + tipo = "Aqua" + categoria = CategoríaMovimiento especial + poder = 90 + precision = 100 + prioridad = 0	+ nombre = "Cola Dragón" + tipo = "Dragón" + categoria = CategoríaMovimiento fisico + poder = 60 + precision = 90 + prioridad = 0	+ nombre = "Pulso Umbrío" + tipo = "Siniestro" + categoria = CategoríaMovimiento especial + poder = 80 + precision = 100 + prioridad = 0	+ nombre = "Bello Mágico" + tipo = "Hielo" + categoria = CategoríaMovimiento especial + poder = 80 + precision = 100 + prioridad = 0

Figura 7: Diagramas estáticos (de clases)

latigoCepa: Movimiento	psiquico: Movimiento	garrDragon: Movimiento	vientoCortante: Movimiento	hiperColmillo: Movimiento
+ nombre = "Latigo Cepa" + tipo = "Planta" + categoria = CategoríaMovimiento fisico + poder = 45 + precision = 100 + prioridad = 0	+ nombre = "Psiquico" + tipo = "Psiquico" + categoria = CategoríaMovimiento especial + poder = 90 + precision = 100 + prioridad = 0	+ nombre = "Garras Dragón" + tipo = "Dragón" + categoria = CategoríaMovimiento fisico + poder = 80 + precision = 100 + prioridad = 0	+ nombre = "Viento Cortante" + tipo = "Volador" + categoria = CategoríaMovimiento especial + poder = 60 + precision = 100 + prioridad = 0	+ nombre = "Hiper Colmillo" + tipo = "Normal" + categoria = CategoríaMovimiento fisico + poder = 80 + precision = 90 + prioridad = 0
torico: Movimiento	polveoVeneno: Movimiento	terremoto: Movimiento	ataqueAereo: Movimiento	bombaLodo: Movimiento
+ nombre = "Torico" + tipo = "Veneno" + categoria = CategoríaMovimiento estado + poder = 0 + precision = 90 + prioridad = 0	+ nombre = "Polve Veneno" + tipo = "Veneno" + categoria = CategoríaMovimiento estado + poder = 0 + precision = 75 + prioridad = 0	+ nombre = "Terremoto" + tipo = "Tierra" + categoria = CategoríaMovimiento fisico + poder = 100 + precision = 100 + prioridad = 0	+ nombre = "Ataque Aéreo" + tipo = "Volador" + categoria = CategoríaMovimiento fisico + poder = 90 + precision = 95 + prioridad = 0	+ nombre = "Bomba Lodo" + tipo = "Veneno" + categoria = CategoríaMovimiento especial + poder = 90 + precision = 100 + prioridad = 0
garrMetal: Movimiento	puncioFuego: Movimiento	punioHielo: Movimiento	zumbido: Movimiento	confusion: Movimiento
+ nombre = "Garr Metal" + tipo = "Acero" + categoria = CategoríaMovimiento fisico + poder = 50 + precision = 95 + prioridad = 0	+ nombre = "Puncio Fuego" + tipo = "Fuego" + categoria = CategoríaMovimiento fisico + poder = 75 + precision = 100 + prioridad = 0	+ nombre = "Punio Hielo" + tipo = "Hielo" + categoria = CategoríaMovimiento fisico + poder = 75 + precision = 100 + prioridad = 0	+ nombre = "Zumbido" + tipo = "Siniestro" + categoria = CategoríaMovimiento especial + poder = 90 + precision = 100 + prioridad = 0	+ nombre = "Confusión" + tipo = "Psiquico" + categoria = CategoríaMovimiento especial + poder = 50 + precision = 100 + prioridad = 1

Figura 8: Diagramas estáticos (de clases)

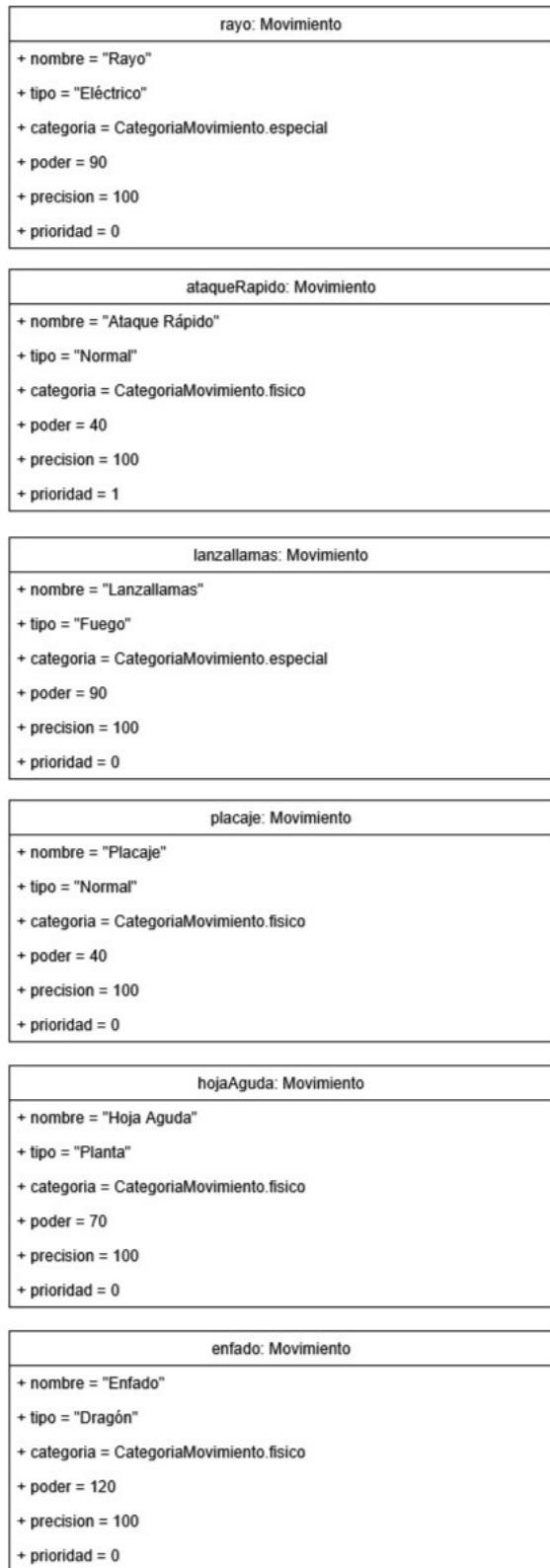


Figura 9: Diagramas estáticos (de clases)

## AudioManager

```
- _instance: AudioManager (static)
- _music: ja.AudioPlayer
+ fx: ap.AudioPlayer
- _rng: Random
- _musicVolume: double

+ AudioManager()
+ musicVolume(): double
+ setVolume(v: double): Future<void>

+ cargarVolumen(): Future<void>

+ playMusic(asset: String, loop: bool = true): Future <void>
+ playRandomMusic(lista: list<String>): Future<void>
+ playFX(file: String): Future<void>
+ stopMusic(): Future<void>
+ fadeOut(): Future<void>
```

Figura 10: Diagramas estáticos (de clases)

<p><b>EventoBatalla</b></p> <ul style="list-style-type: none"> <li>- tipo: TipoEvento</li> <li>- mensaje: String?</li> </ul> <p>+ EventoBatalla(tipo: TipoEvento, mensaje: String?)</p>	<p><b>ResultadoTurno</b></p> <ul style="list-style-type: none"> <li>- eventos: List&lt;EventoBatalla&gt;</li> </ul> <p>+ ResultadoTurno(eventos: List&lt;EventoBatalla&gt;)</p>
<p><b>ControlBattalla</b></p> <ul style="list-style-type: none"> <li>- equipoA: List&lt;Pokemon&gt;</li> <li>- equipoB: List&lt;Pokemon&gt;</li> <li>+ activoA: Pokemon</li> <li>+ activoB: Pokemon</li> <li>- rng: Random</li> <li>+ mochila: List&lt;item&gt;</li> </ul> <p>+ ControlBatalla(equipoA: List&lt;Pokemon&gt;, equipoB: List&lt;Pokemon&gt;)</p> <p>+ resolverTurno(idxMovJugador: int): Resultado Turno</p> <p>- _eventosAtaque(atacante: Pokemon, defensor: Pokemon, movimiento: Movimiento, esJugador: bool): List&lt;EventoBatalla&gt;</p> <p>- _resolverMovimientoEstado(atacante: Pokemon, defensor: Pokemon, movimiento: Movimiento, esJugador: bool, eventos: List&lt;EventoBatalla&gt;): void</p> <p>- _eventosEstadoSecundario(movimiento: Movimiento, atacante: Pokemon, defensor: Pokemon, esJugador: bool): List&lt;EventoBatalla&gt;</p> <p>- _nombreEstado(e: EstadoAlterado): String</p> <p>- _eventoFinDeTurno(p: Pokemon, esJugador: bool): List&lt;EventoBatalla&gt;</p> <p>- _bloqueoEstado(p: Pokemon, esJugador: bool): EventoBatalla?</p> <p>- _calcularDanio(atacante: Pokemon, defensor: Pokemon, mov: Movimiento):int</p> <p>- _decidirOrden(movJ: Movimiento, , pokeJ: Pokemon, movE: Movimiento, pokeE: Pokemon): bool</p> <p>+ usarItemSobreJugador(item: Item, idxItem: int): ResultadoTurno</p> <p>+ jugadorCambia(idx: int): String</p> <p>+ cambioAuto(esJugador: bool): Pokemon?</p>	

Figura 11: Diagramas estáticos (de clases)

<b>GameManager</b>
<ul style="list-style-type: none"> <li>- _instance: GameManager(static)</li> <li>+ jugadorNombre: String</li> <li>+ spriteEntrenador: String</li> <li>+ dificultad: String</li> <li>+ itemsJugador: List&lt;Item&gt;</li> <li>+ roster30: List&lt;Pokemon&gt;</li> <li>+ equipoJugador: List&lt;Pokemon&gt;</li> <li>+ torre: List&lt;String&gt;</li> </ul>
<ul style="list-style-type: none"> <li>+ GameManager()</li> <li>+ nuevaPartida(nombre: String, sprite: String, dificultad, String): Future&lt;void&gt;</li> <li>+ generarRoster20(): List&lt;Pokemon&gt;</li> <li>+ generarEquipo6(roster: List&lt;Pokemon&gt;): List&lt;Pokemon&gt;</li> <li>+ generarTorre(): List&lt;String&gt;</li> <li>- _clonarPokemon(p: Pokemon): Pokemon</li>   <li>+ generarItemsFacil(): List&lt;Item&gt;</li> <li>+ generarItemsMedio(): List&lt;Item&gt;</li> <li>+ generarItemsDificil(): List&lt;Item&gt;</li> </ul>

Figura 12: Diagramas estáticos (de clases)

<b>Item</b>
<ul style="list-style-type: none"> <li>+ nombre: String</li> <li>+ tipo: Tipolitem</li> <li>+ cantidad: int</li> <li>+ montoHP: int?</li> <li>+ cura: estadoAlterado?</li> </ul>
<ul style="list-style-type: none"> <li>+ Item(nombre: String, tipo: tipolitem, cantidad: int, montoHP: int?, cura, estadoAlterado?)</li> </ul>

Figura 13: Diagramas estáticos (de clases)

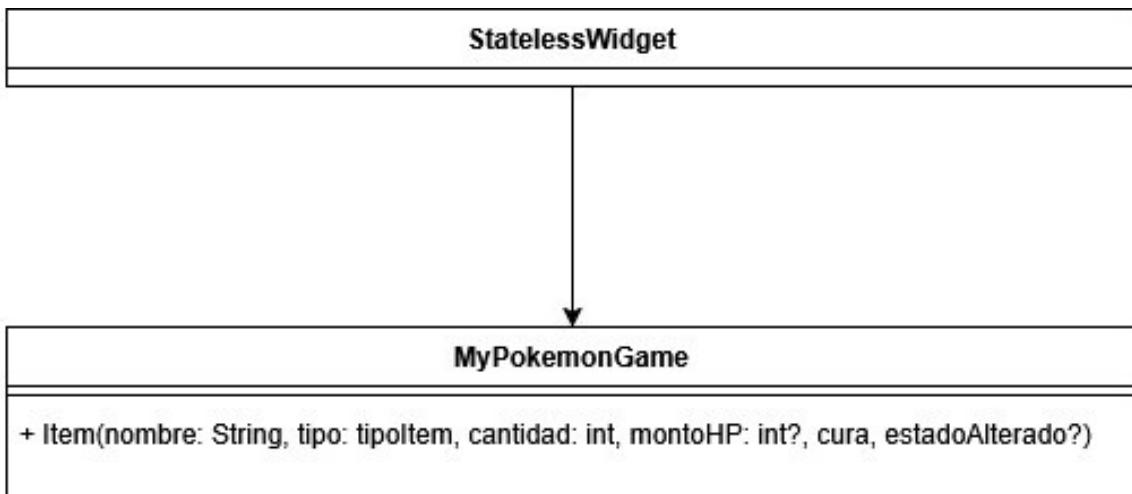


Figura 14: Diagramas estáticos (de clases)

<b>EntrenadorData</b>	<b>entrenadorOswaldo: EntrenadorData</b>
+ nombre: String + sprite: String + equipo: List<Pokemon>	+ nombre = "Oswaldo" + sprite = "oswaldo.png" + equipo = [arcaine, vaporeon, donphan, breloom, magnezone, umbreon]
+ EntrenadorData(nombre: String, sprite: String, equipo: List<Pokemon>)	
<b>entrenadorAbdiel: EntrenadorData</b>	<b>entrenadorAndres: EntrenadorData</b>
+ nombre = "Abdiel" + sprite = "abdiel.png" + equipo = [garchomp, tyranitar, dragonite, metagross, hydreigon, goodra]	+ nombre = "Andres" + sprite = "andres.png" + equipo = [metagross, swampert, tokekiss, ferrothorn, infernape, gliscor]
<b>entrenadorAlan: EntrenadorData</b>	<b>entrenadorRoberto: EntrenadorData</b>
+ nombre = "Alan" + sprite = "alan.png" + equipo = [charizard, tyranigar, gengar, dragonite, lucario, gardevoir]	+ nombre = "Roberto" + sprite = "roberto.png" + equipo = [sceptile, alakazam, salamence, aggron, absol, ninetales]
<b>entrenadorAsh: EntrenadorData</b>	<b>entrenadorRojo: EntrenadorData</b>
+ nombre = "Ash" + sprite = "ash.png" + equipo = [pikachu, charizard, bulbasur, squirtle, snorlax, greninja]	+ nombre = "Rojo" + sprite = "rojo.png" + equipo = [pikachu, charizard, venusaur, blastoise, snorlax espeon]
<b>entrenadorEthan: EntrenadorData</b>	<b>entrenadorSinooh: EntrenadorData</b>
+ nombre = "Ethan" + sprite = "ethan.png" + equipo = [tylosion, feraligtr, magenium, ampharos, heracross, tyranitar]	+ nombre = "Cynthia" + sprite = "cynthia.png" + equipo = [spiritomb, roserade, tokekiss, lucario, milotic, garchomp]

Figura 15: Diagramas estáticos (de clases)

## 5.1. Diagramas estáticos UI

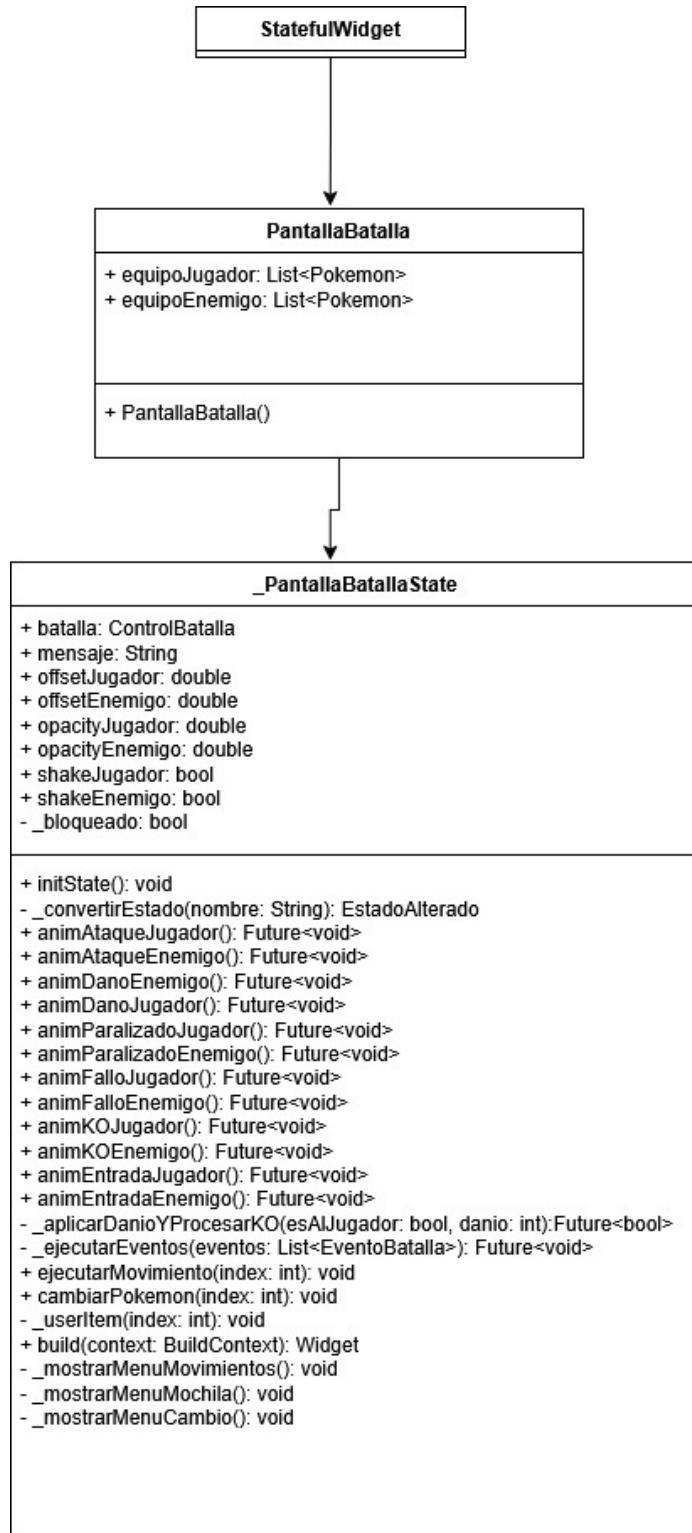


Figura 16: Diagramas estáticos (UI)

## 5.2. Diagramas estáticos UI

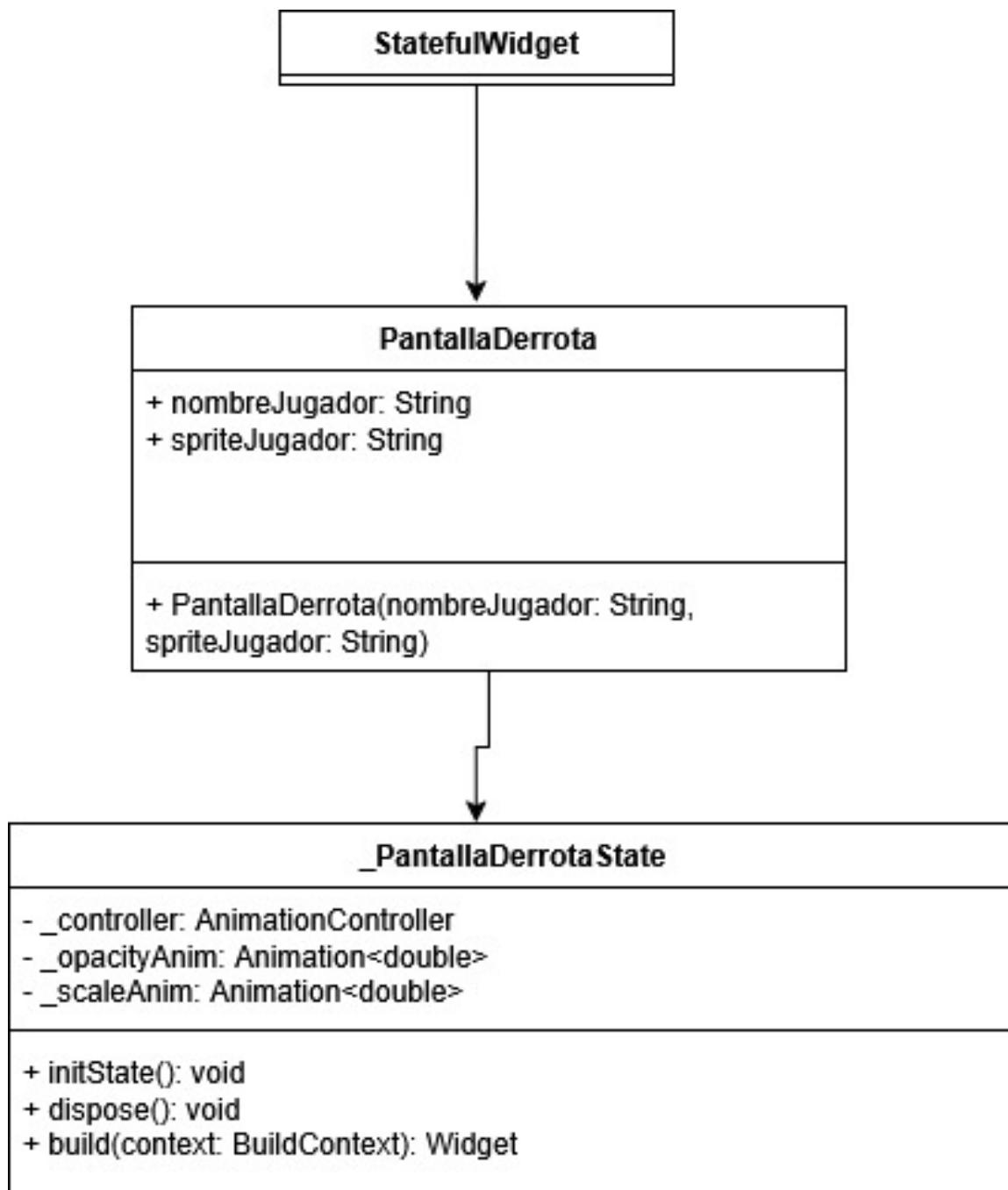


Figura 17: Diagramas estáticos (UI)

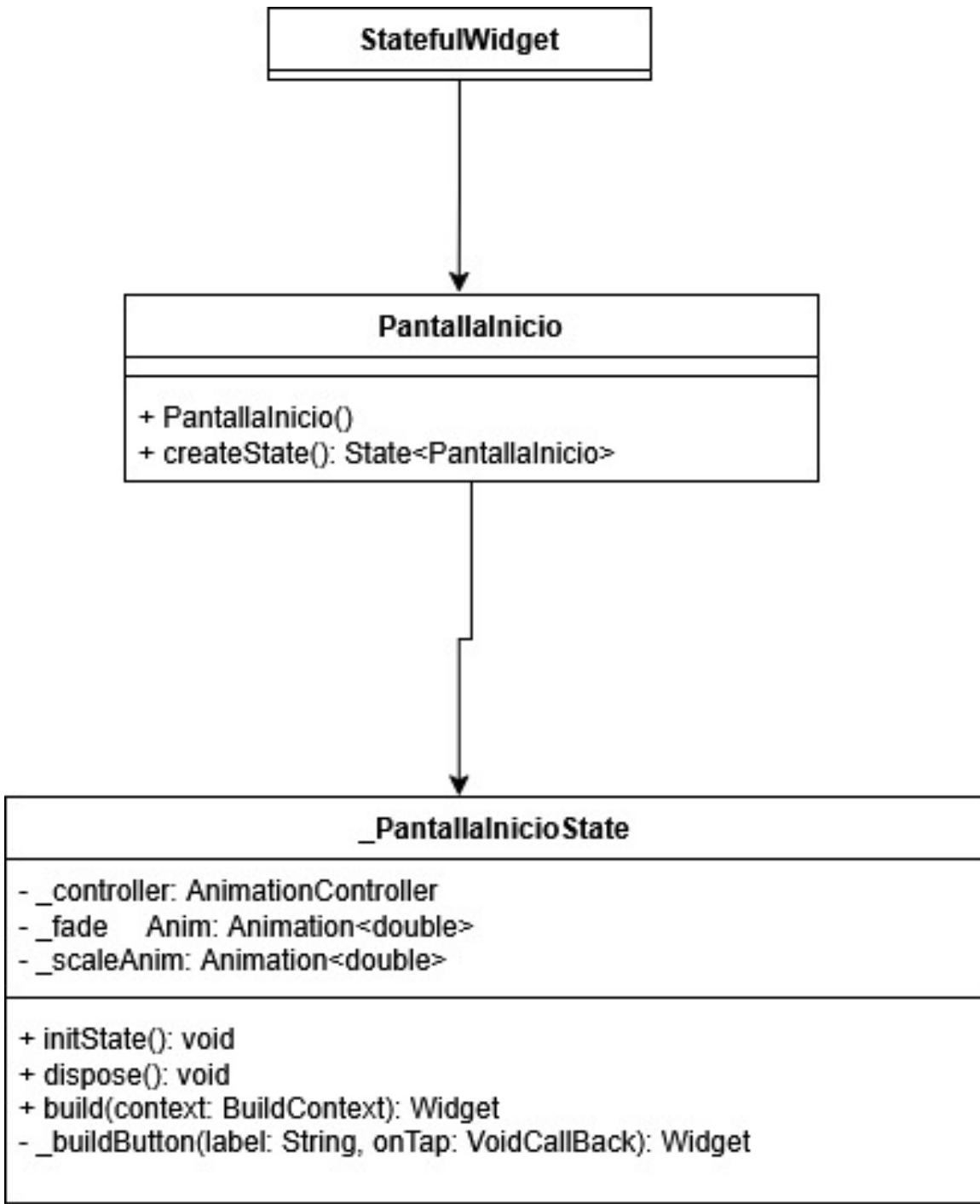


Figura 18: Diagramas estáticos (UI)

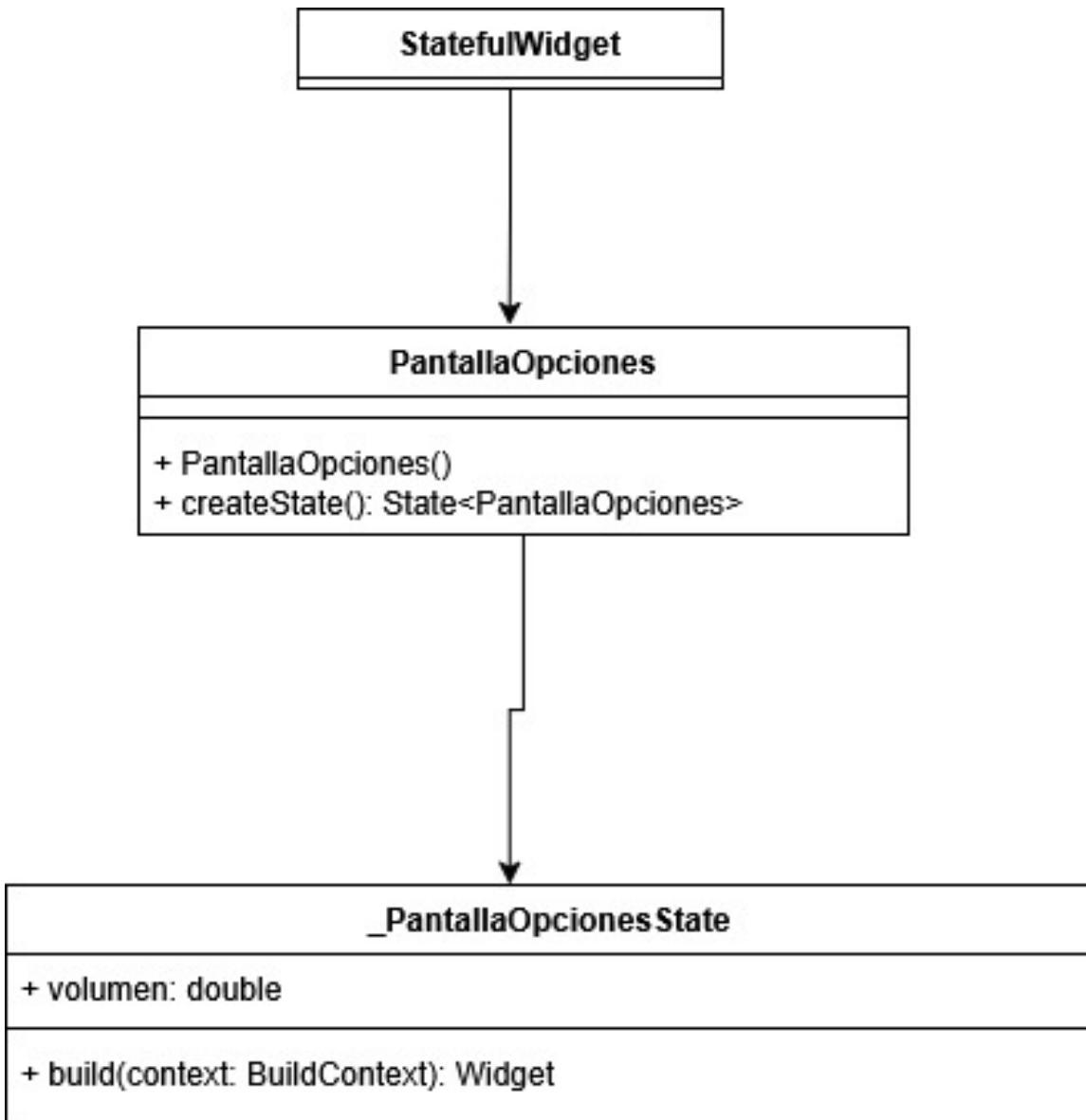


Figura 19: Diagramas estáticos (UI)

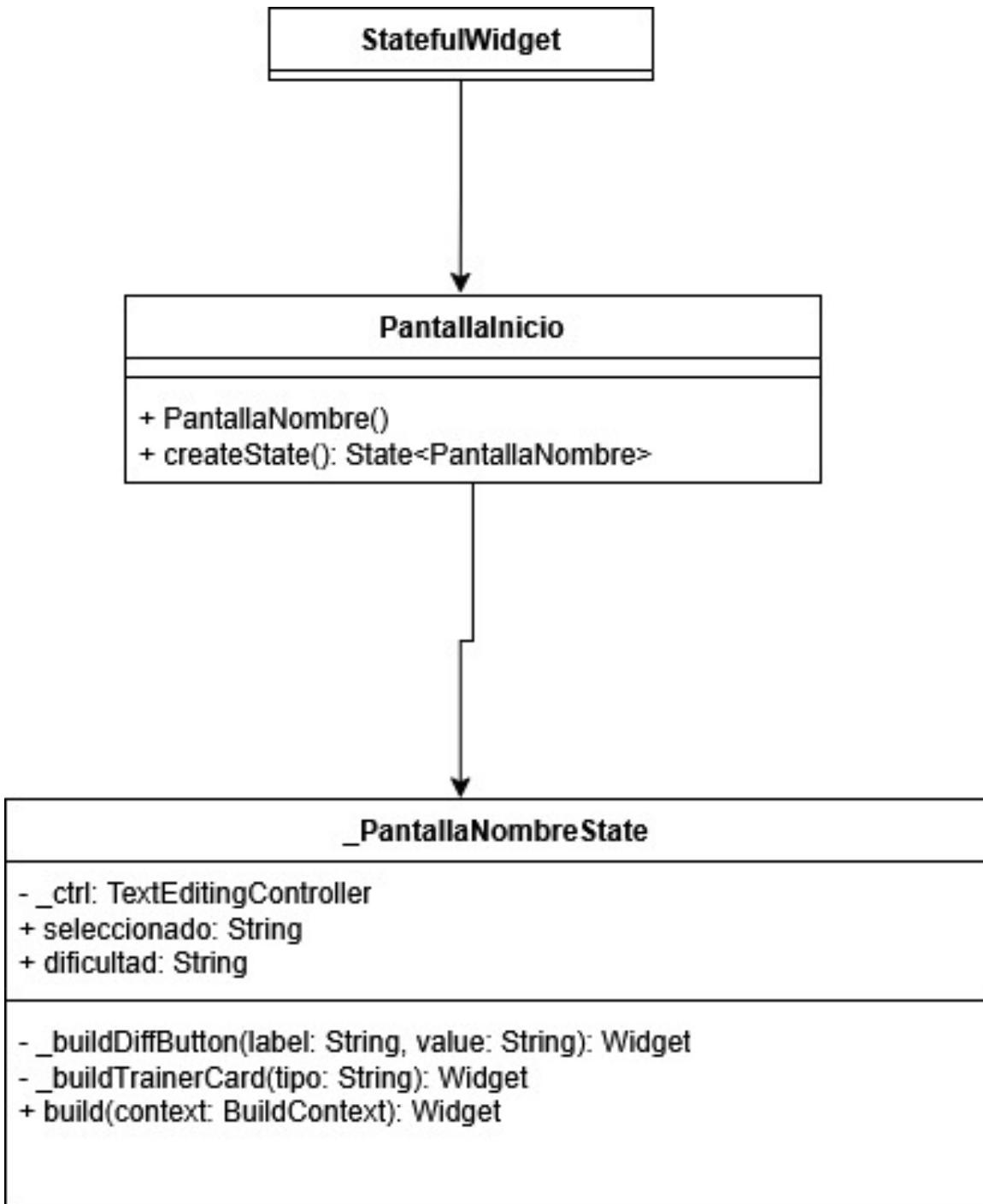


Figura 20: Diagramas estáticos (UI)

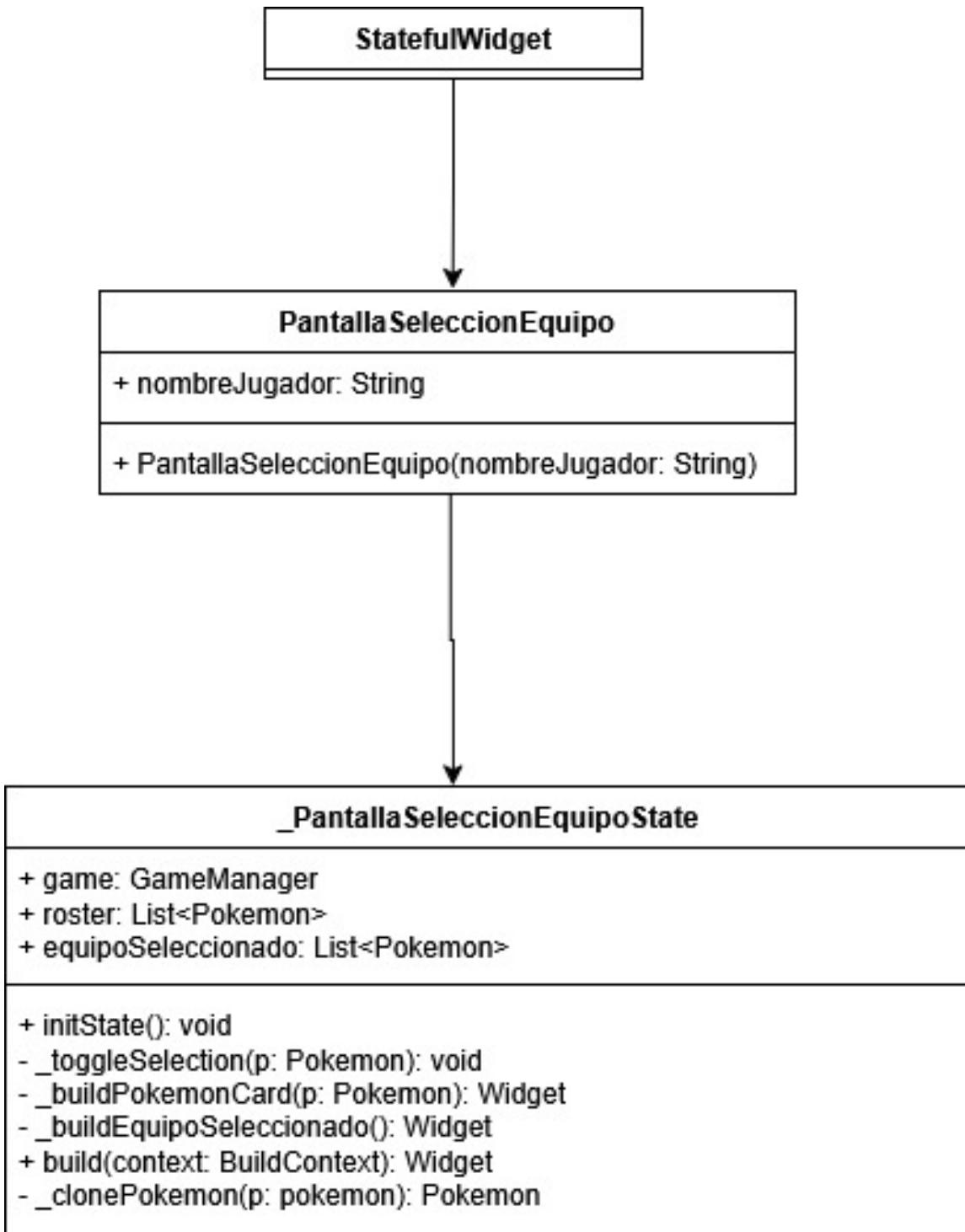


Figura 21: Diagramas estáticos (UI)

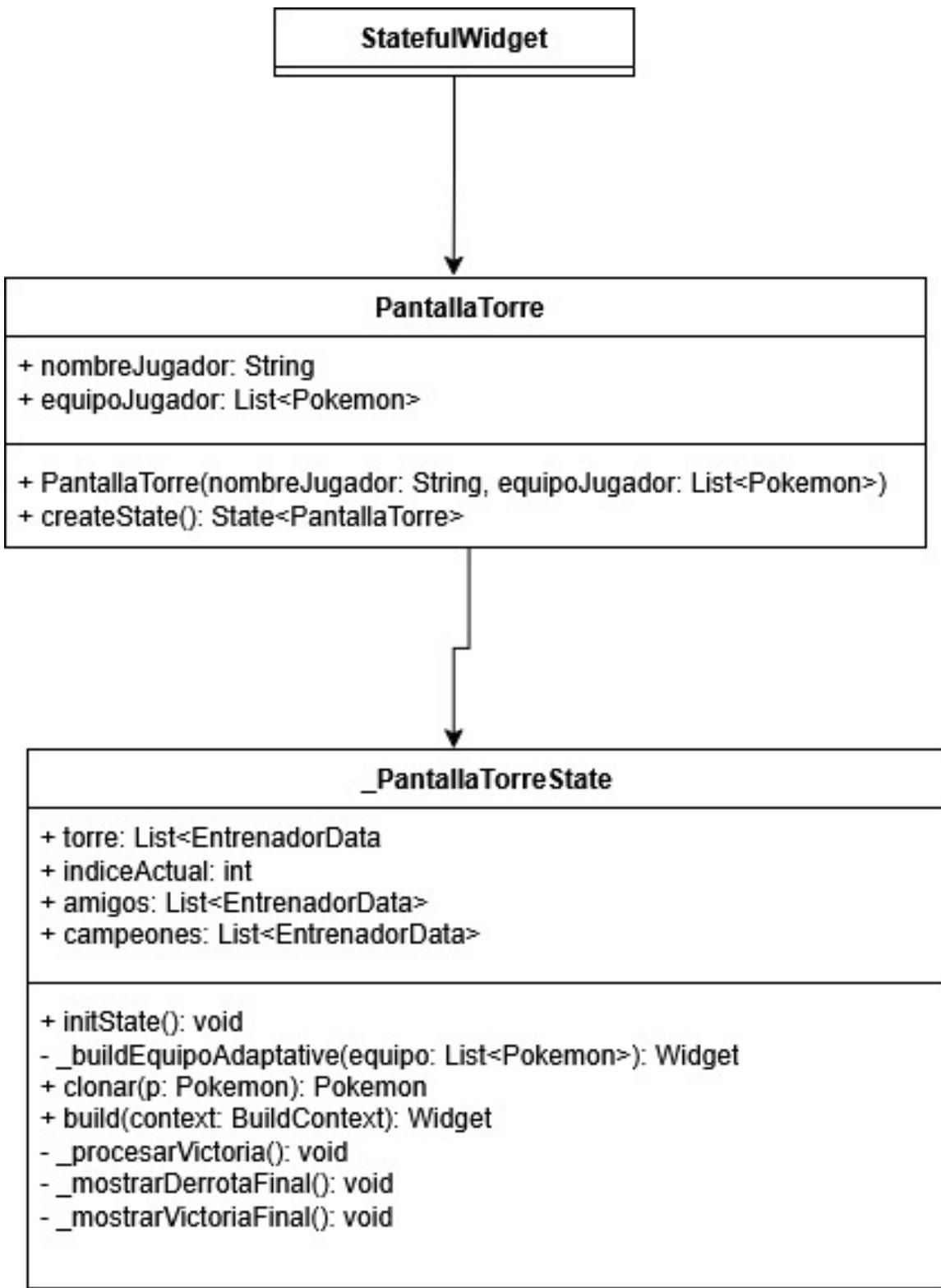


Figura 22: Diagramas estáticos (UI)

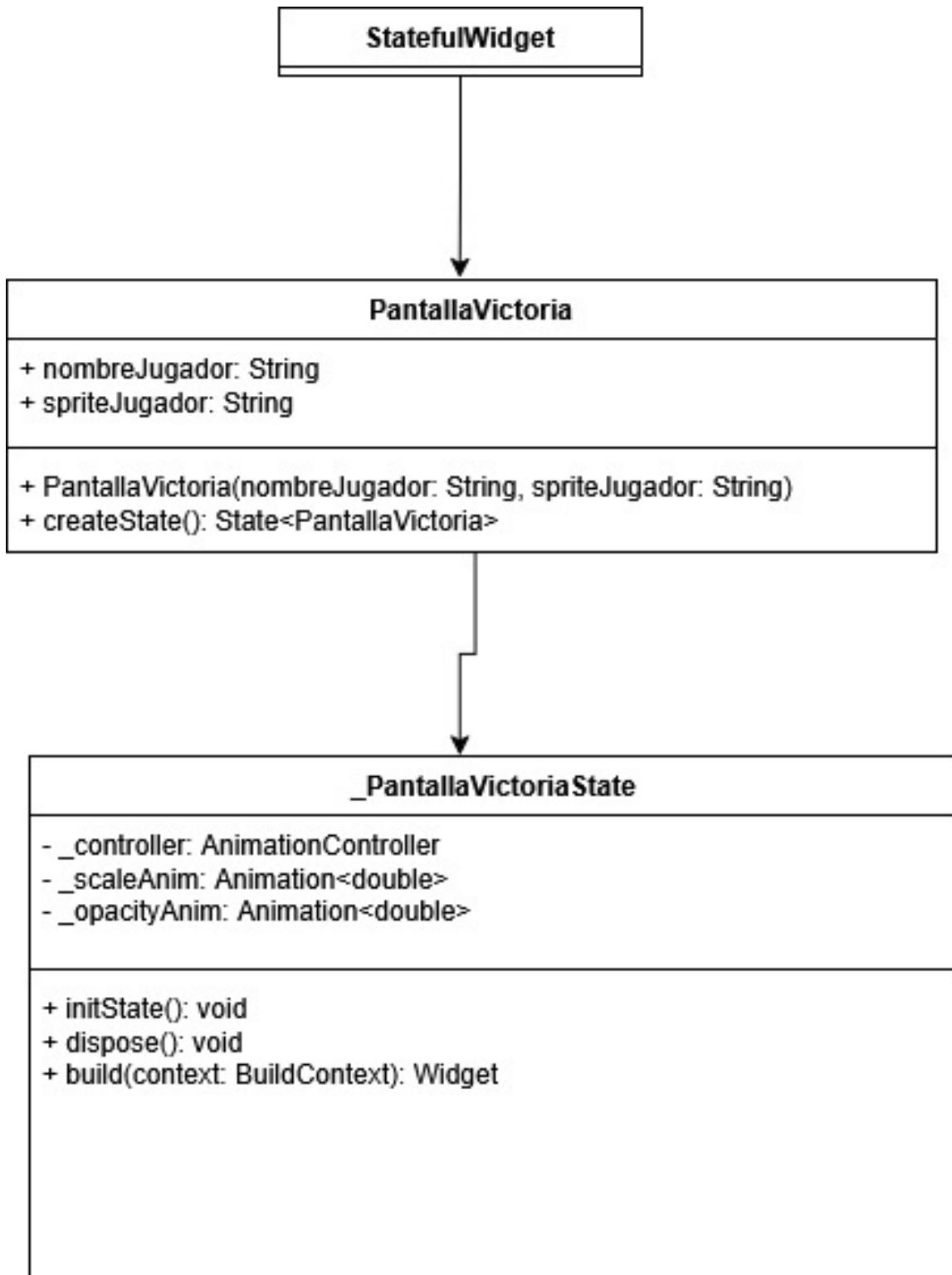


Figura 23: Diagramas estáticos (UI)

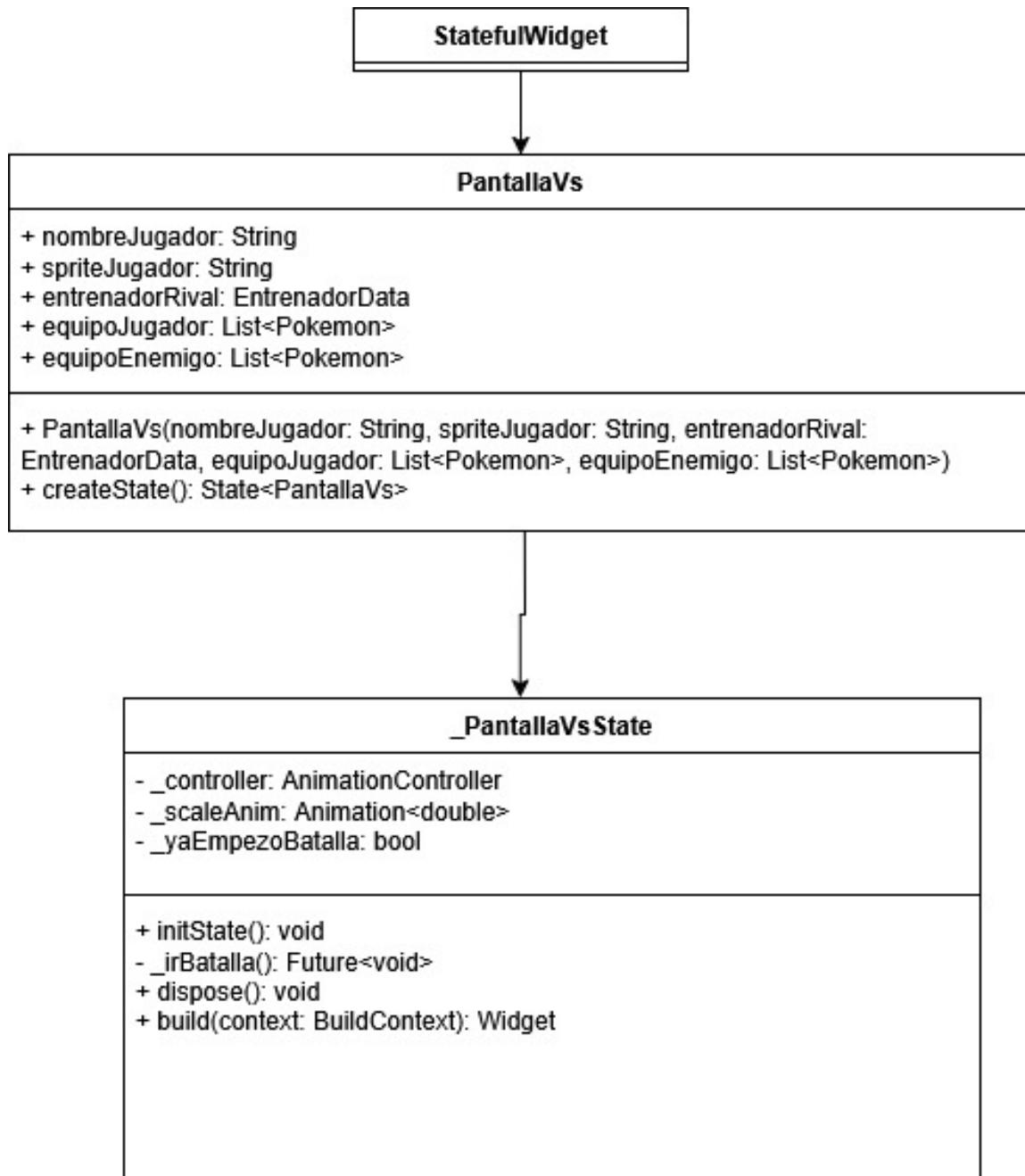


Figura 24: Diagramas estáticos (UI)

## **6. Integrante 3, 322114461**

Debido a la complejidad y cantidad de códigos requeridos para el correcto funcionamiento de nuestro proyecto, el diagrama dinámico UML se dividió en 3, tratando de respetar la estructura y lógica de los códigos originales.

## Diagrama UML dinámico

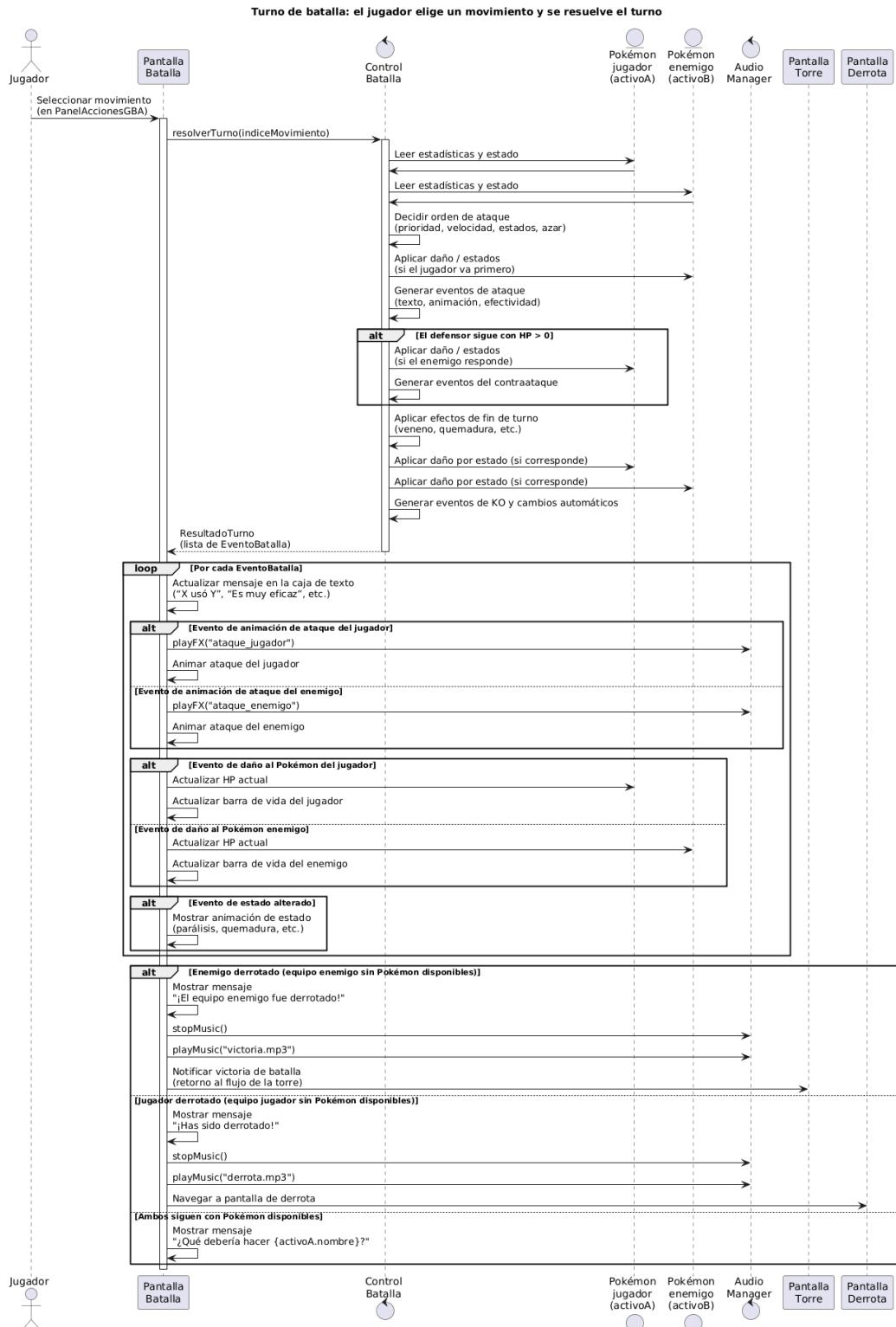


Figura 25: Diagrama de secuencia principal (turno de batalla)

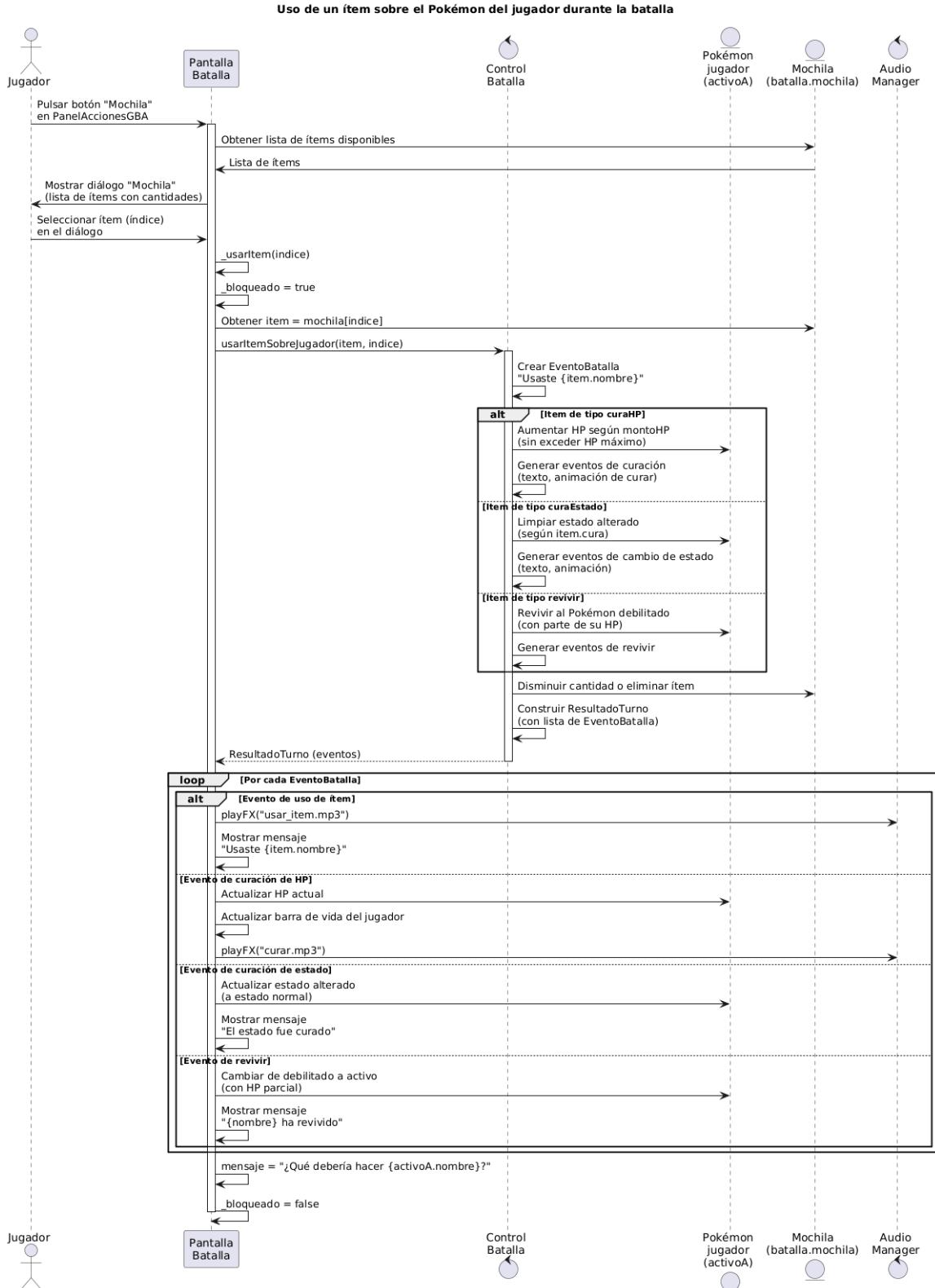


Figura 26: Diagrama de secuencia complementario (uso de ítem de la mochila)

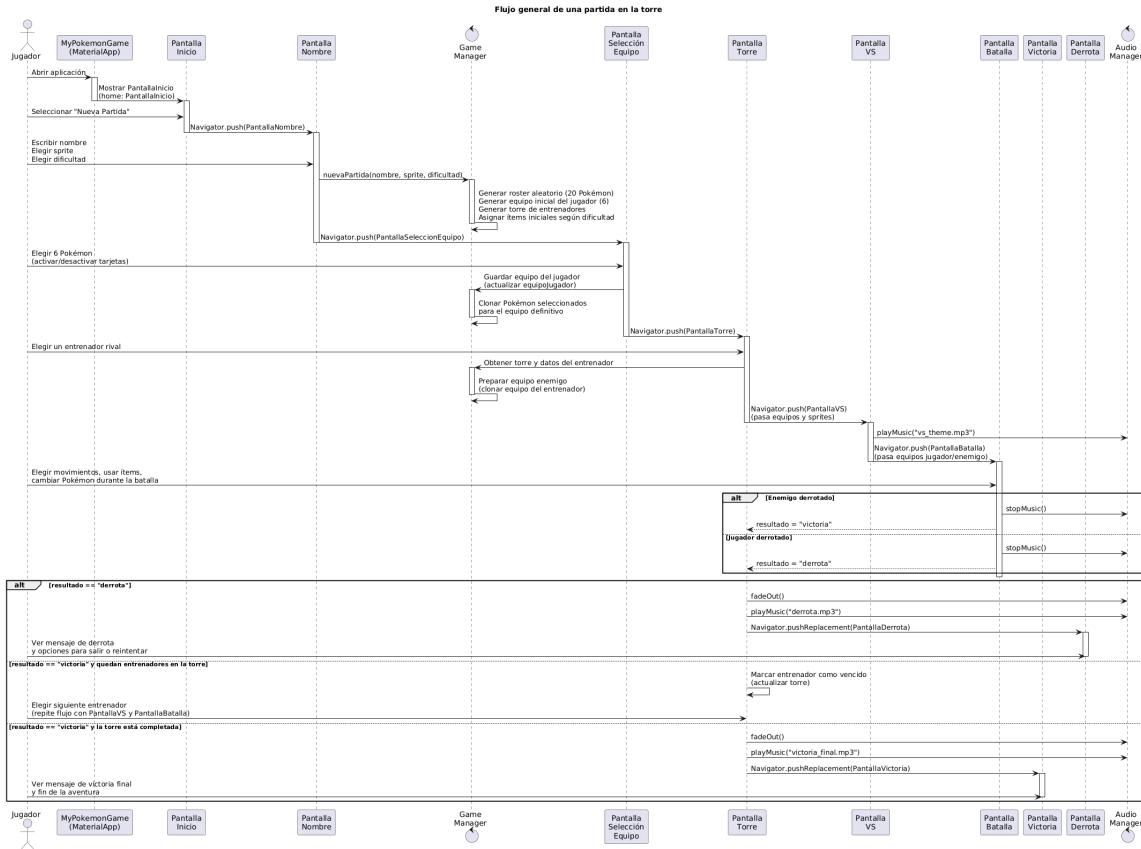


Figura 27: Diagrama de secuencia macro (flujo de pantallas durante la partida)

## **7. Integrante 4, 425093384**

### **Desarrollo en serie de pasos**

El desarrollo de nuestro juego se llevó a cabo mediante una metodología iterativa y modular, permitiendo que la arquitectura creciera de forma ordenada desde un prototipo en consola hasta una aplicación con interfaz gráfica, animaciones y soporte multiplataforma. A continuación se describen detalladamente las etapas abordadas durante este proceso.

### **7.1. Diseño de clases fundamentales y modelo de datos**

El proyecto inició con la definición de las estructuras esenciales del juego: `Pokémon`, `Movimiento`, `Item`, la tabla de tipos y módulos auxiliares para estadísticas y cálculo de daño. Se buscó un modelo flexible que permitiera añadir nuevas mecánicas sin comprometer la estabilidad del sistema.

### **7.2. Implementación del prototipo en terminal**

Se construyó una primera versión completamente funcional en consola. Este prototipo permitió validar la lógica de combate, las fórmulas de daño, el manejo de turnos, condiciones de victoria y uso de movimientos. Fue un paso crucial para construir la lógica sin involucrar aún interfaz gráfica.

### **7.3. Migración a un modelo basado en eventos**

Con el objetivo de hacer independientes la lógica del juego y la interfaz, se adoptó un sistema basado en eventos. Se creó el enumerador `TipoEvento` y la estructura `EventoBatalla`, lo que permitió que la UI actuara únicamente como lectora de eventos emitidos por el motor de la batalla. Este diseño habilitó un flujo de animaciones mucho más controlado y escalable.

### **7.4. Decisiones de diseño del modo de juego**

Tras evaluar diferentes enfoques, se definió un formato tipo Torre de Batalla. El usuario ingresa su nombre, una dificultad, selecciona un equipo de seis Pokémon a partir de un roster generado aleatoriamente por el `GameManager`, buscando variedad y rejugabilidad. Este diseño simplificó la estructura de niveles y concentró el desarrollo en el sistema de batalla.



Figura 28: Roster dinámico

## 7.5. Diseño de la dificultad y sistema de ítems

Además del modo de juego, se definió un sistema de dificultad que el jugador elige al inicio de la partida. La dificultad afecta solamente a la cantidad de Ítems que posee el jugador al iniciar.



Figura 29: Usuario, dificultad y sprite

## 7.6. Ampliación del motor de batalla

Se implementaron mecánicas para emular con mayor fidelidad la experiencia de combate de los juegos originales: prioridad por velocidad, fallos de movimientos, daño por estados alterados, efectos secundarios, curaciones y objetos en batalla.

## **7.7. Ajustes y mejoras en fórmulas internas**

Asimismo, se realizaron ajustes a distintas fórmulas de daño, precisión y efectos para mejorar la jugabilidad general y evitar resultados inconsistentes.

## **7.8. Integración de la primera interfaz gráfica (UI)**

Se construyó la pantalla principal de batalla en Flutter, incorporando animaciones como desplazamientos horizontales, variación de opacidad, sacudidas (shake), transiciones escalonadas y manejo de tiempos para dar ritmo a los enfrentamientos. La comunicación UI-controlador quedó completamente basada en la interpretación de eventos.

## **7.9. Creación de widgets personalizados para el HUD**

Se desarrollaron múltiples componentes reutilizables que conforman el estilo visual del juego: `SpritePokemon`, `CajaTextoGBA`, `PanelAccionesGBA`, `PokemonHudGBA`, entre otros. Cada widget fue diseñado para emular la estética de los juegos oficiales de Game Boy Advance, manteniendo coherencia visual con la franquicia original.

## **7.10. Generación del roster dinámico mediante un Singleton**

Se implementó el `GameManager` como un patrón Singleton, encargado de generar un roster único de veinte Pokémon para cada partida, guardar el nombre y sprite del jugador, además de regir el cambio entre pantallas y la lógica del juego en general(no del sistema de batalla).

## **7.11. Pantalla de selección de equipo**

Se construyó una pantalla interactiva para permitir al jugador elegir su equipo final de seis Pokémon. Se añadieron validaciones, retroalimentación visual y comunicación directa con el administrador de partida para continuar con el modo torre de batalla.

## **7.12. Búsqueda y análisis de assets**

Se investigaron y seleccionaron sprites, fondos y tipografías adecuadas para la estética del proyecto.

## **7.13. Desarrollo e integración del sistema de audio**

Para el audio se realizó una investigación de distintas librerías y formas de reproducir sonidos en Flutter, tanto en escritorio como en Android. Tras varias pruebas y errores se llegó a un `AudioManager` propio que se encarga de:

- reproducir música de fondo en bucle,

- lanzar efectos de sonido durante la batalla,
- cambiar pistas sin cortes bruscos,

Este módulo se probó repetidamente hasta lograr que la música y los efectos se escucharan de forma estable y sincronizada con las acciones del juego.

### 7.14. Menú de opciones para el control de audio

A partir del sistema de audio se creó un menú de opciones sencillo, enfocado únicamente en el control del volumen. Desde este menú el jugador puede ajustar el volumen general,

Este menú se conecta directamente con el `AudioManager`, lo que permite reflejar los cambios de forma inmediata durante la partida.



Figura 30: Slider de volúmen

### 7.15. Porteo a Android y ajustes multiplataforma

Una vez establecida la lógica y la UI, se realizó la exportación del proyecto hacia Android. Se analizaron:

- la legibilidad del HUD,
- el comportamiento de los sprites animados,
- la estabilidad del audio,
- el escalado de la interfaz según resolución.

Durante esta etapa se identificaron diferencias importantes frente a la versión de escritorio (Web y Ubuntu), lo que llevó a optimizaciones adicionales en animaciones y modificaciones del tamaño de letra.

### **7.16. Fase colaborativa de pruebas (testing)**

Todos los integrantes probaron el juego en Android para localizar errores, caídas inesperadas, problemas de animación o desincronización de eventos. Se generó una lista de fallos, se priorizaron y se registraron para su corrección.

### **7.17. Parcheado y optimización**

Se corrigieron errores relacionados con la aplicación del daño, métodos faltantes, problemas de actualización del HUD, eventos que no alcanzaban la interfaz y cálculos incorrectos en ciertas interacciones.

### **7.18. Documentación técnica**

La documentación se generó durante todas las fases del desarrollo, pero se consolidó al final. Incluyó diagramas UML explicados anteriormente, uso del patrón Singleton, diseño MVC, descripción del motor de eventos, y un registro con comentarios a lo largo de todo el proyecto.

## 8. Integrante 5, 322094152

### Resultados Relevantes:

Con este proyecto en Flutter hemos logrado aprender nuevos conceptos al cambiar de lenguaje pero también reforzamos otros sobre las bondades que nos ofrece la programación orientada a objetos. Los resultados relevantes van orientados tanto al desarrollo del juego para que sea funcional como los requisitos para puntaje extra propuestos, por ello se mostrará como es que se manejaron los pokemons y sus tipos, como es que estos pueden luchar, como es que se porteo a Android, el añadir audio, la implementación de la mochila con objetos y el manejo de los estados alterados:

### 8.1. Polimorfismo en clases pokemon y ataques

Como se vio, un pokemon tiene nombre, tipo primario, HP, ataque, defensa, ataque especial, defensa especial, velocidad de ataque y nivel además de una lista de movimientos para cada uno. Entonces para aprovechar el paradigma tenemos que hallar los puntos de intersección de los pokemons, para empezar, todos son precisamente pokemons, así que es conveniente crear una clase de pokemons, pero hay diferentes tipos, así que de acuerdo a su tipo algunos tendrán ciertas debilidades o fortalezas, es ahí donde entra la herencia y el polimorfismo, pues ya con eso solo quedaría modificar estadísticas y darle nombre propio a cada objeto pokemon teniendo almacenados 53 de estos para utilizarse, cada uno con ciertas estadísticas, tipo y set de movimientos.

Para poder usar los movimientos se da un caso similar, pues estos pueden ser físicos, especiales o de estado, cada uno ejercerá un daño diferente teniendo en cuenta la defensa o el tipo del pokemon, e inclusive si es que tiene prioridad de atacar primero como lo es el ataque rápido. Entonces para poder tener una correcta gestión de elementos, sus pros y sus contra debemos hacer uso de tablas de tipo.

Al momento de iniciar una nueva partida, el juego nos arrojara 20 pokemons para elegir, por lo que desde el inicio podremos contar con usar mecánicas similares a los juegos originales buscando un balance de tipos en nuestros pokemons para evitar las penalizaciones de decantarnos por un solo tipo.

Tu equipo:



Venusaur



Garchomp



Magnezone



Goodra



Salamence



Torterra

¡COMBATIR!

Figura 31: Equipo seleccionado por el usuario



Figura 32: Roster dinámico y selección de pokémon

## 8.2. Batallas pokemon

Como ya se vio en el desarrollo las batallas son el eje del juego y tiene en cuenta varios factores como la velocidad de los pokemons, la barra de HP, los tipos de movimientos a emplear y más adelante se destacará los estados secundarios que puede experimentar un pokémon y el uso de objetos.

Los sucesos ocurridos durante el gameplay son denominados eventos habiendo 12 en total y aunque unos desencadenan a otros, todos y cada uno de ellos son indispensables para la fluidez de la batalla. Todos los eventos tienen un tipo y posible mensaje siendo procesados por la UI. Durante un combate ocurren varias cosas aunque podrían pasar desapercibidas, por ejemplo, primero se decide al atacante de acuerdo a

la velocidad, después, se genera un rol entre ser un atacante y defensor, el uso de un movimiento seleccionado y analizar si este ataque es efectivo, no efectivo, neutro, genera algún estado o incluso si es capaz de fallar, posteriormente detectar si el pokémon recibe daño de un estado o es debilitado y así seleccionar otro.

Los pokémon pueden ser cambiados a medio combate, aunque sin la penalización de perder el turno, porque inmediatamente después de cambiar se comparan las velocidades y no es hasta un usar un movimiento valido cuando gasta su turno.

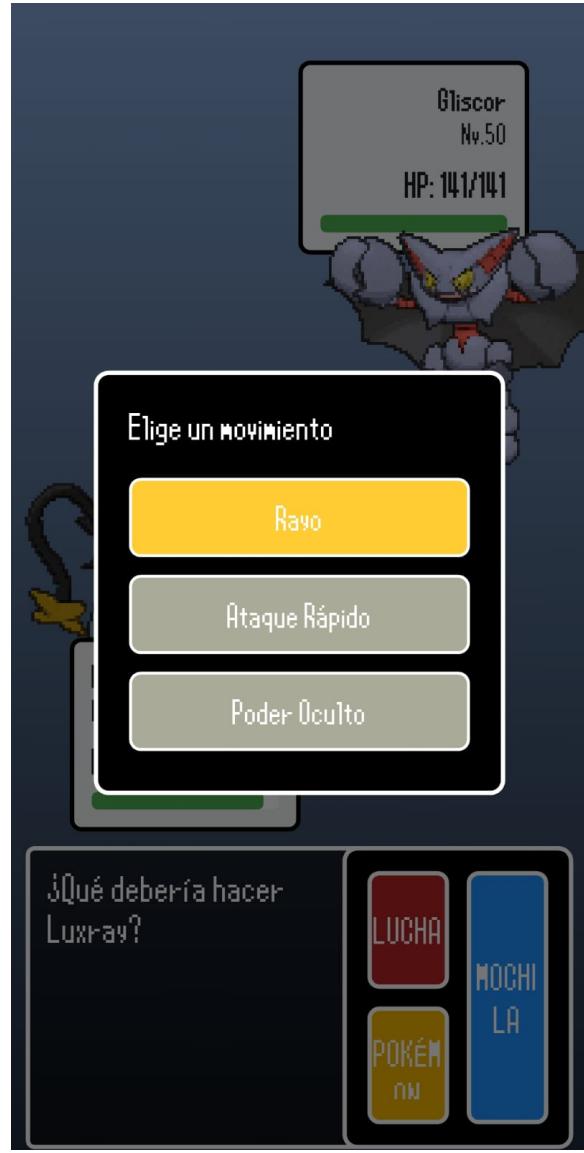


Figura 33: Menú de ataques

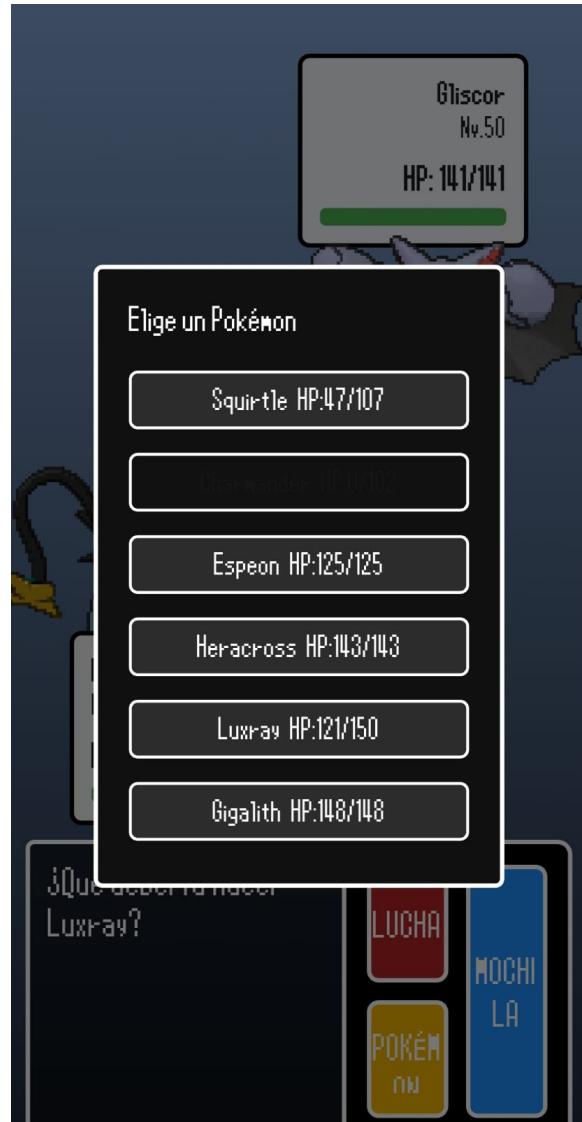


Figura 34: Cambio de pokémon

### 8.3. Portear a otros sistemas

Gracias a las características de Flutter nos es posible portear el proyecto a otros sistemas como lo serían Android, un formato portátil típico de los juegos originales. Así que la manera más fácil para portear hacia Android es utilizar el comando flutter build apk –realeas, compilar y generar nuestro proyecto apk listo para portear hacia otros sistemas.

Simplemente se compartió en el grupo de trabajo y ya estaba todo listo para operar como un juego.

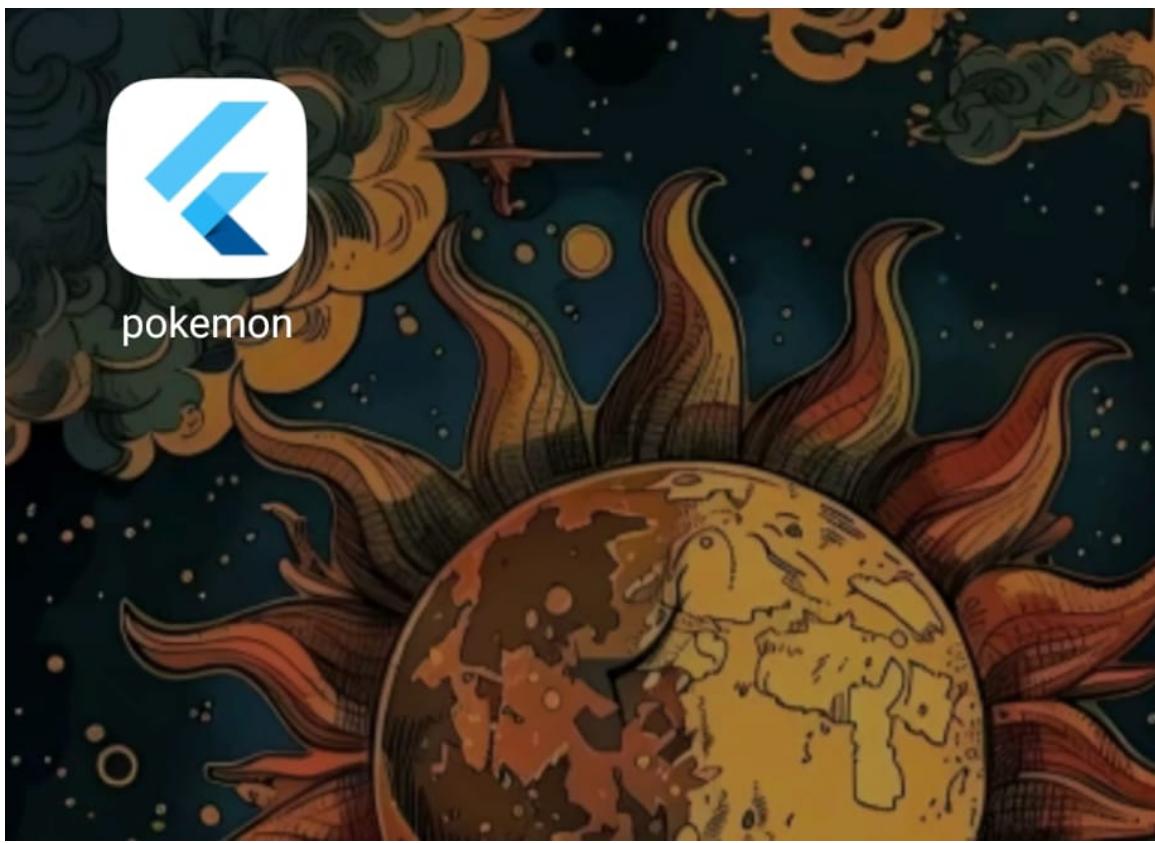


Figura 35: Aplicación



Figura 36: Pantalla principal de la aplicación

#### 8.4. Importar audio

Para la importación de audio se tuvo que hacer uso de varias librerías y paquetes como se mencionó, esto con el objetivo de tener un apartado sonoro no tan monótono, pues uno esperaría que la música del menú sea diferente en el menú, durante un combate e inclusive cuando has ganado o perdido. Entonces, hay que tener en cuenta los diferentes tipos de recursos auditivos que podemos usar, por lo que retomamos fragmentos de OST de juegos de pokémon para así dar un aura más inmersa.

Aunque en un reporte no puede escucharse, simplemente al momento de iniciar el juego salta la música junto con la pantalla de título.

A través del audio manager manejamos la música, efectos de sonido y volumen, con justaudio para música, audioplayers para efectos FX, shredpreferences para guardar el volumen. Se utilizó singleton para su operación, manejándolo como un servidor global de audio. Para mantener la variedad se optó por cierta aleatoriedad en la música por lo que a veces cambiará dependiendo de la partida que estés jugando, aunque si es que pasas un considerable tiempo en un fase será notable que la música está en loop.



Figura 37: Volumen

## 8.5. Implementación de la mochila

La mochila como tal solo es una lista que almacena items, en nuestra aventura nuestra mochila podrá tener diferentes de estos objetos que si bien comparten la característica de ser consumibles, no todos actúan de la misma manera, pues dependiendo

de su uso o pueden ser un salvavidas o ser desperdiciados.

La dificultad está estrechamente relacionada con la generación de items en nuestra mochila así como que tanto surten efecto, en estos objetos también se puede ver rasgos de herencia y polimorfismo pues no importa cual dificultad elijas todos son un tipo de Item el cual dependiendo de su función es de tipo curaH, tipo curaEstado o tipo revivir, cada uno posee un nombre, una cantidad dentro de la bolsa y un monto si es que recupera vida o repara un estado. Entonces podemos decir que montoHP y cura son opcionales porque cada tipo de item usa solo uno de ellos.

Ya en el gameplay podemos acceder a ellos presionando un botón de las opciones y simplemente seleccionar el objeto que deseemos usar, después de unos segundos nuestro objeto se gastará adecuadamente y surtirá efecto.



Figura 38: Mochila

## 8.6. Estados de alteración

Los estados de alteración aparecen después de cierto ataque con probabilidad de infringirlo o que justamente su fin es provocar un estado en el oponente, para ello.

Por ejemplo, rayo es un movimiento tipo eléctrico el cual puede paralizar al oponente, para decidir su probabilidad se tiene que generar un número entero del 0 al 99, si este número es menor que 10, entonces el estado alterado se habrá añadido al rival, si embargo, como primero se analiza el tipo del pokémon, entonces simplemente pokemons con inmunidad a rayo como los tipo Tierra no surtirán efecto. Esta misma exemplificación funciona con movimientos de fuego y veneno.

En el caso de Congelar, el cual es un ataque tipo hielo y su fin es generar el estado Congelado.<sup>entonces</sup> se procesa como un ataque normal sin daño pero con efecto. Este ataque es especial, debido a que como no hace daño al final del turno simplemente retorna el evento.

Los estados incluidos en el juego son congelado, paralizado, quemadura y veneno.

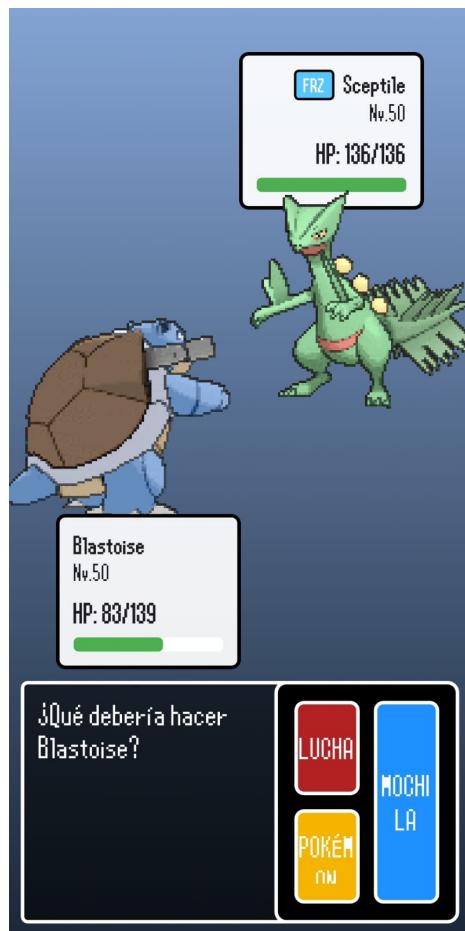


Figura 39: Batalla entre dos pokemones

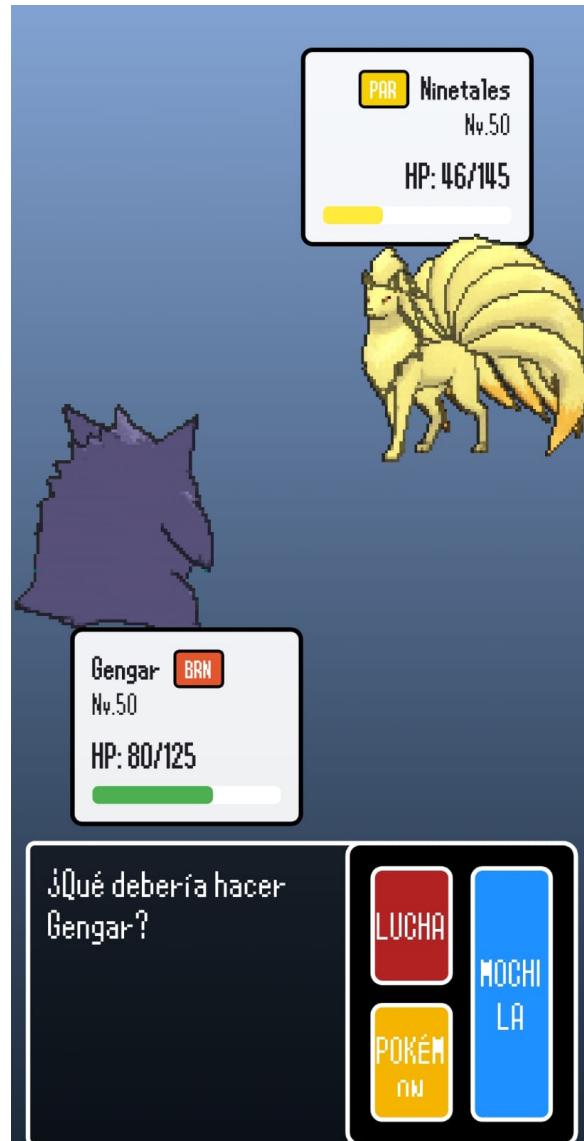


Figura 40: Batalla entre dos pokemones

## 9. Resultados

### 9.1. Pantalla inicial



Figura 41: Pantalla inicial al abrir el juego

Se muestra un menú con las opciones de modular el volumen o iniciar una partida

## 9.2. Opciones

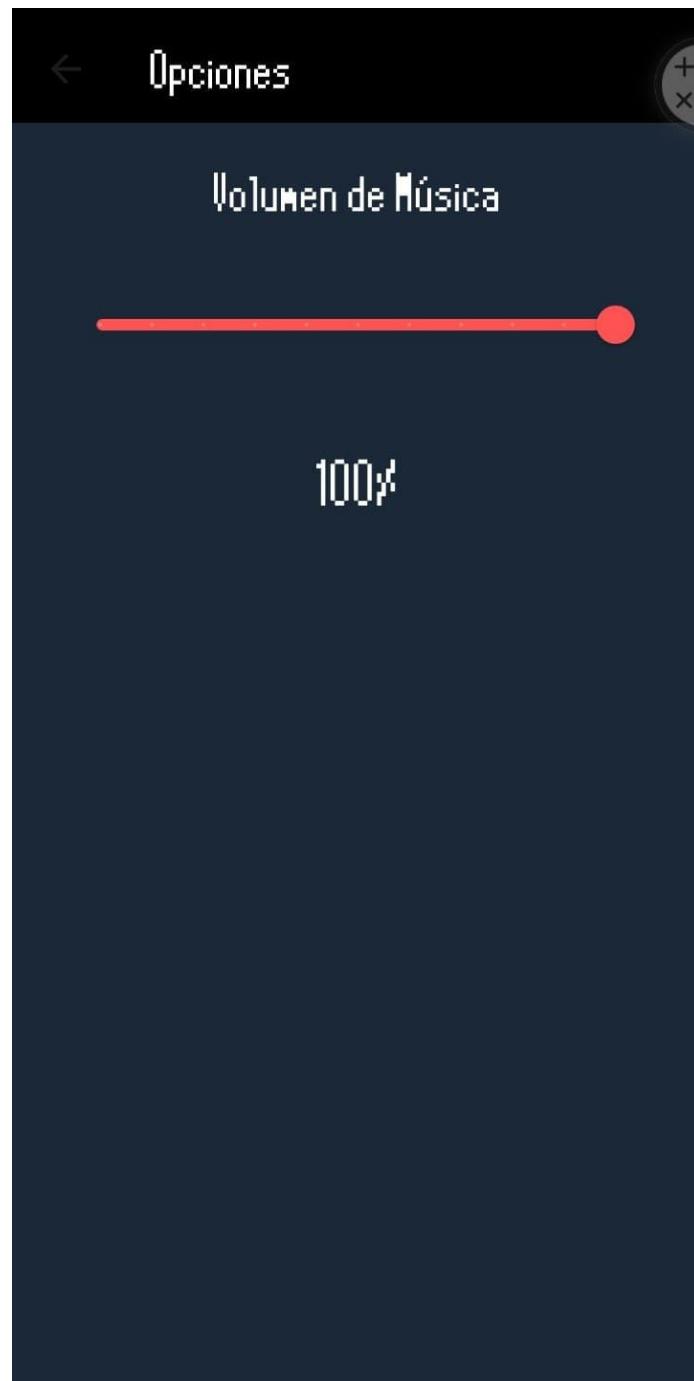


Figura 42: Regulador de volumen

al seleccionar la opción de regular el volumen, se lleva a la siguiente pantalla donde se pueden hacer las modificaciones deseadas

### 9.3. Nueva partida



Figura 43: Iniciando una partida

Al empezar una nueva partida, se mostrará la siguiente pantalla y se dejará al usuario personalizar su entrenador

#### 9.4. Selección de equipo



Figura 44: Eligiendo equipo

Al terminar de personalizar el entrenador, se le darán aleatoriamente 20 pokemones al usuario para que arme un equipo de 6



Figura 45: Equipo formado

Cuando se seleccionen los 6 pokemones, se podrá empezar a combatir

## 9.5. Combates



Figura 46: Combates contra entrenadores

El usuario enfrentará a 6 maestros en total, cuando derrote a uno, continuará con el siguiente

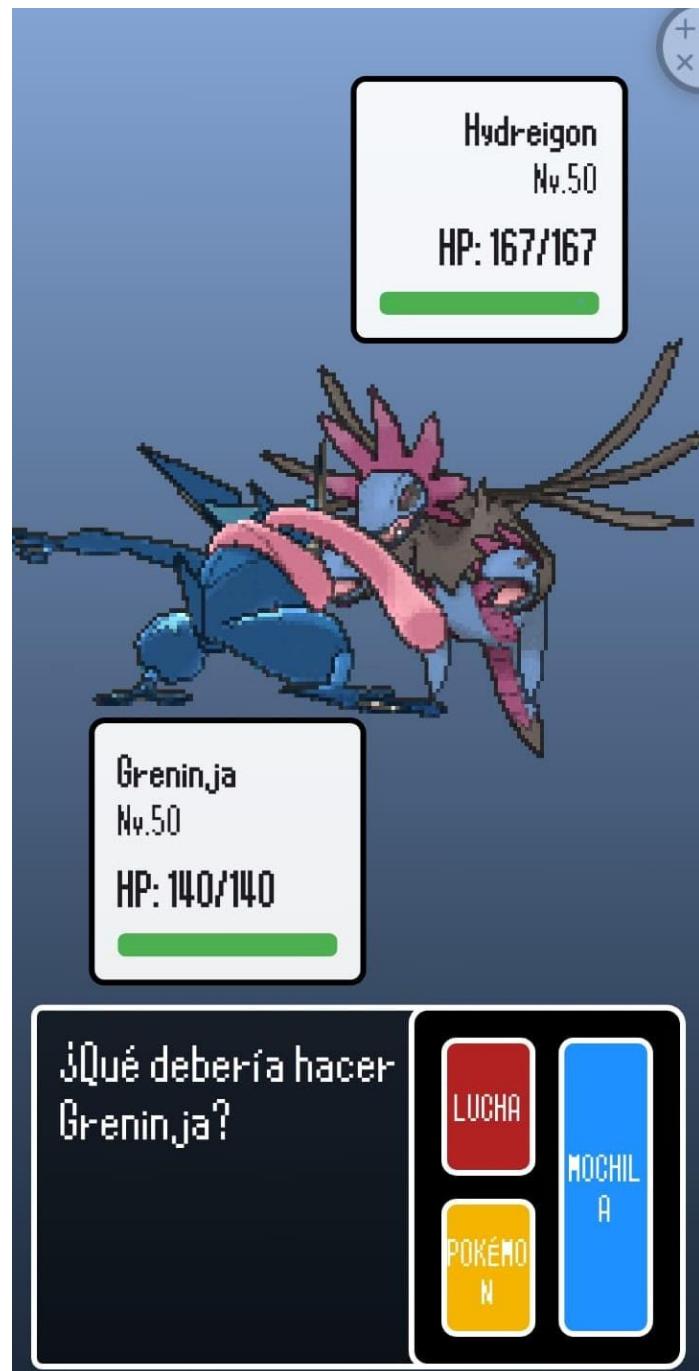


Figura 47: Pokemones combatiendo

Los pokemones empezarán su batalla por turnos



Figura 48: Ataques

El usuario podrá elegir el ataque a realizar

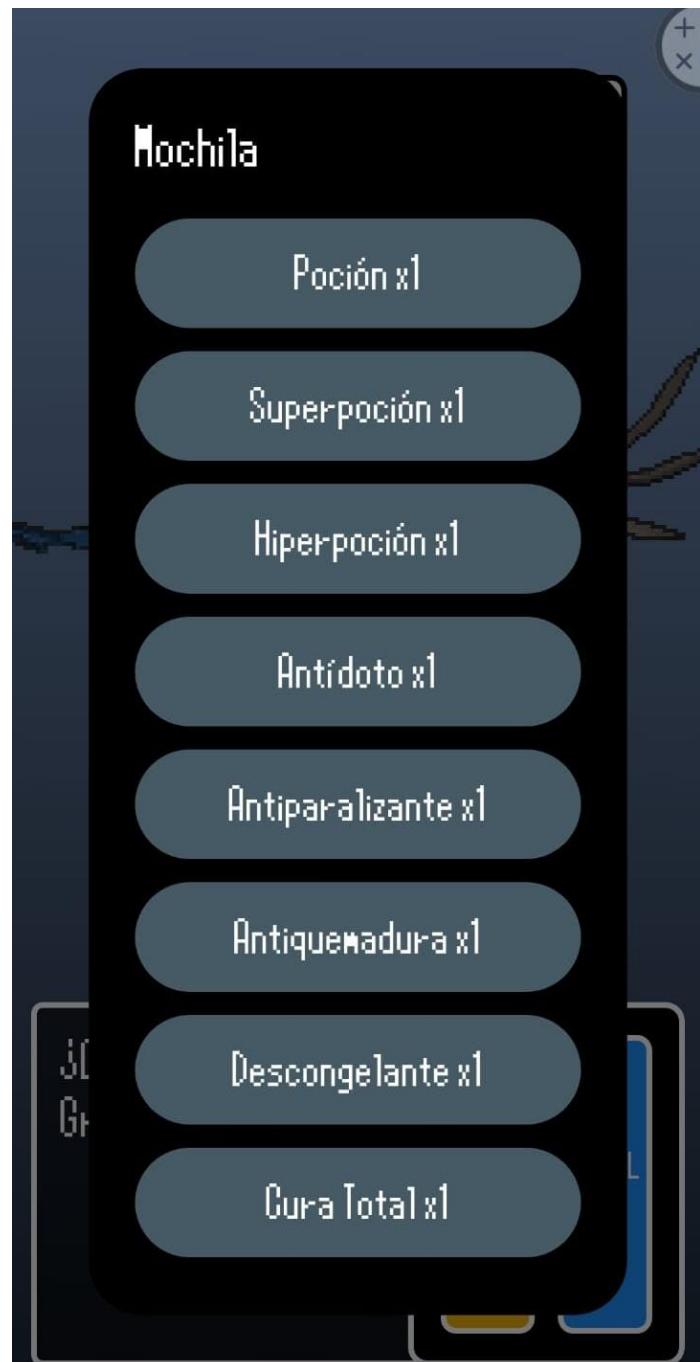


Figura 49: Mochila

Dependiendo de la dificultad seleccionada, el usuario contará con cierta cantidad de ítems consumibles en su mochila

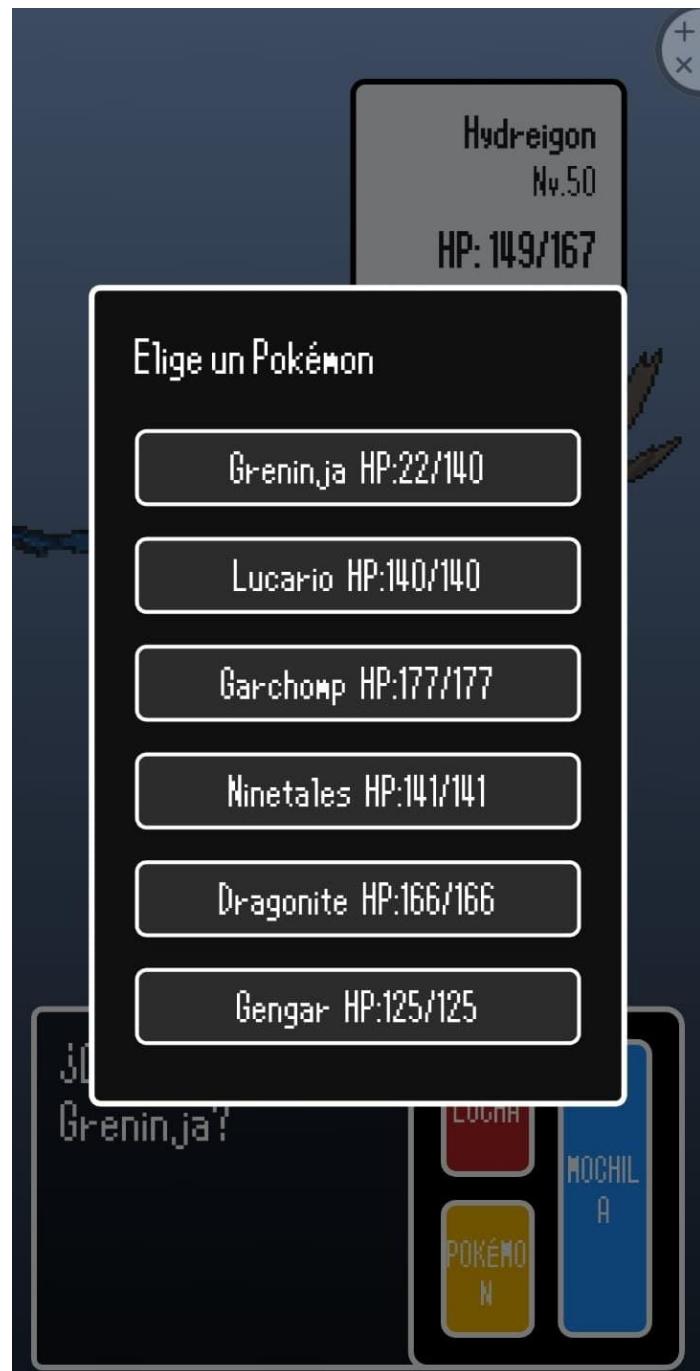


Figura 50: Cambiar de pokémon

Manualmente el usuario podrá cambiar de entre sus pokemones elegidos si así lo desea

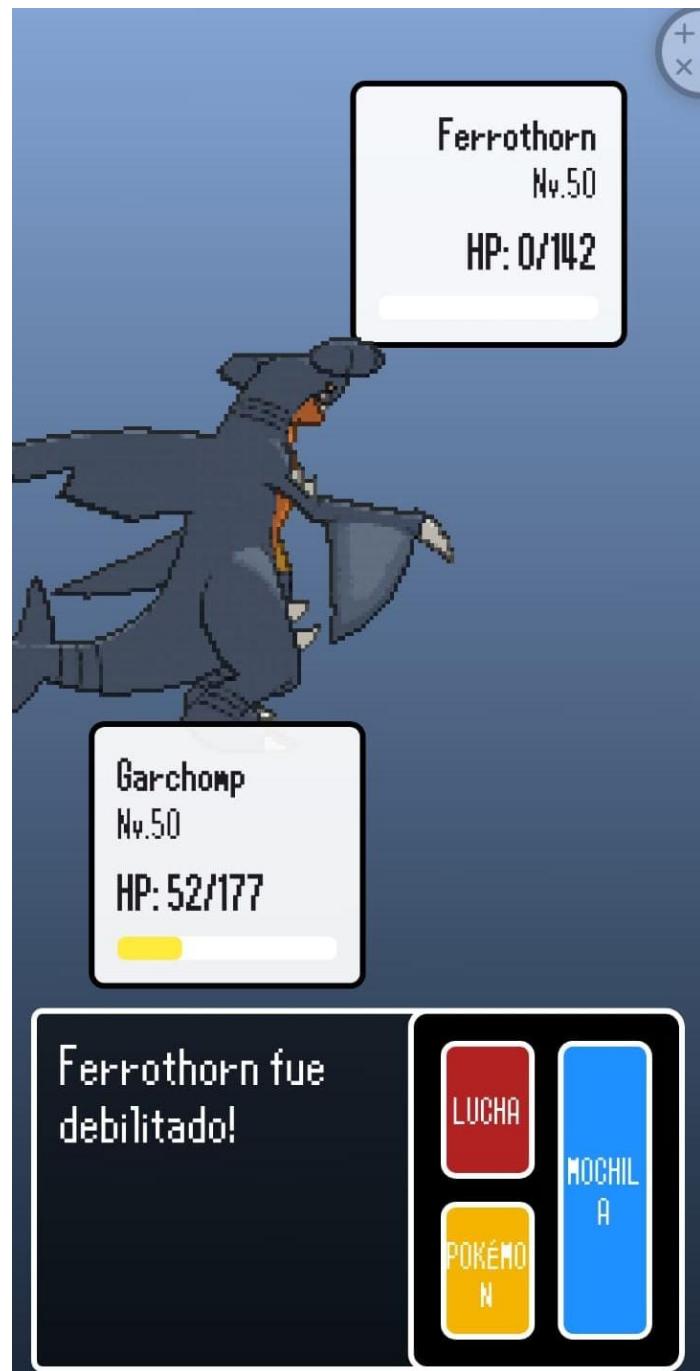


Figura 51: Pokemon derrotado

Cuando un pokémon es derrotado y aún quedan pokemones en pie, automáticamente cambiará al siguiente pokémon

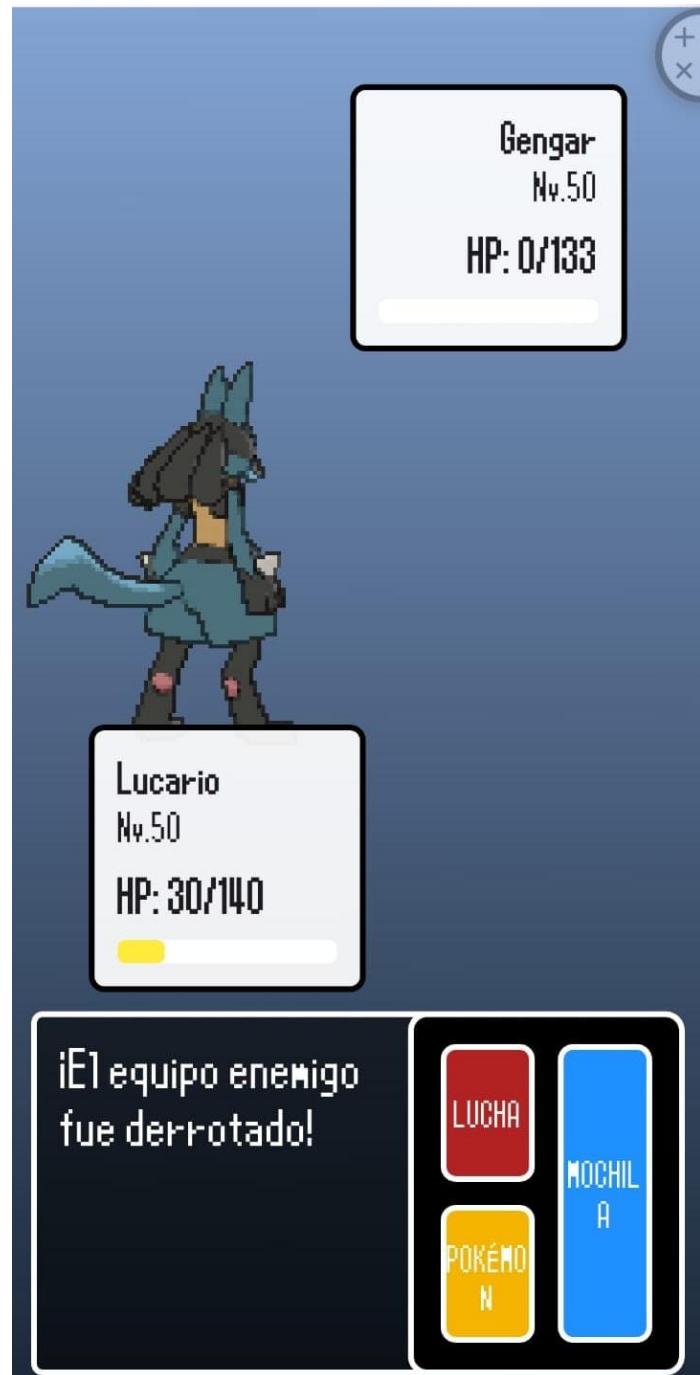


Figura 52: Enemigo derrotado

Cuando el rival se queda sin pokemones en pie, el rival es vencido y se avanza contra el siguiente oponente

## 9.6. Pantallas finales

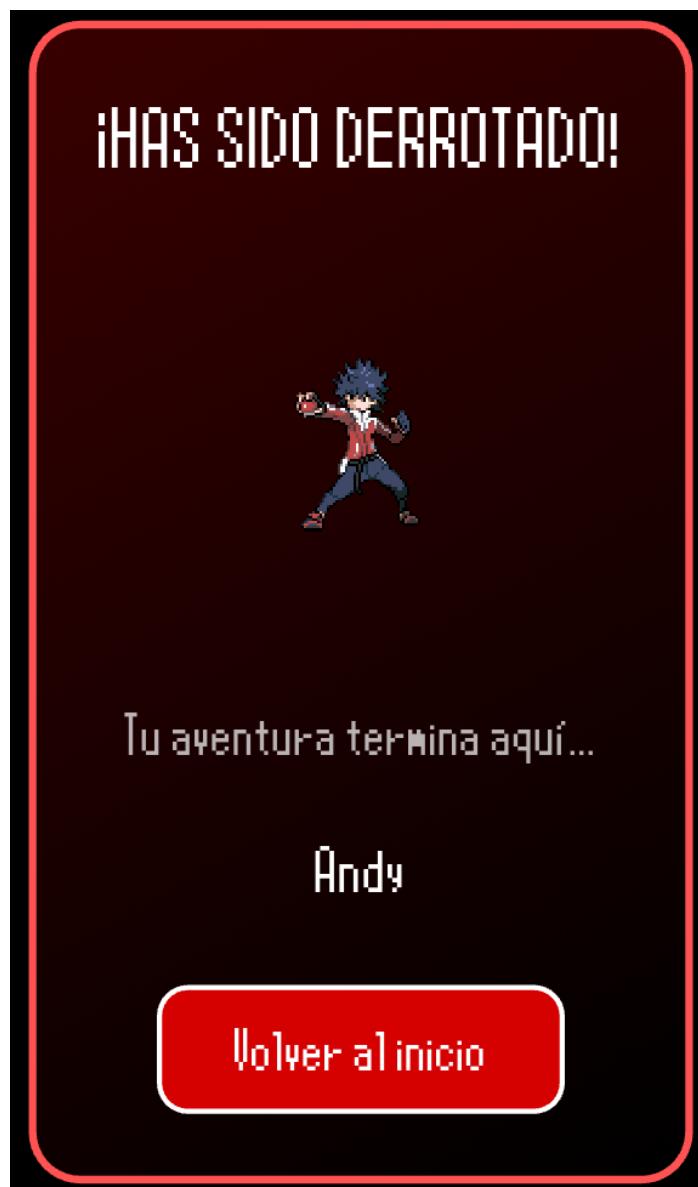


Figura 53: Derrota

Cuando el equipo del usuario es derrotado, se muestra la pantalla final de derrota



Figura 54: Victoria

Cuando el usuario derrota a los 6 entrenadores rivales, se muestra la imagen de victoria, en ambos casos (derrota y victoria) se da la opción de volver a la pantalla de inicio.

322114461

## 10. Conclusiones

Al crear este simulador de batallas Pokémon, comprobamos que los conceptos que aprendimos a lo largo de todo el curso de programación orientada a objetos son la base para desarrollar aplicaciones estructuradas y extensibles, como un juego, en este caso. A través de clases como Pokémon, Movimiento, Item, y clases de alto nivel como GameManager y ControlBatalla, se pudo modelar un sistema complejo manteniendo una clara separación de responsabilidades. La modularidad y la cohesión son esenciales para mantener el orden en un proyecto con múltiples archivos y funciones, como se demostró al separar los modelos, los controladores, los datos, la interfaz gráfica y el audio.

El proyecto también aplicó el control de estado interno de los objetos y el encapsulamiento. Que los Pokémon llevaran su propia vida, estados alterados y estadísticas y que sólo el motor de batalla los modificara, ayudó a la coherencia lógica en los combates. Además, gracias a las enumeraciones para definir clases de movimiento, tipos de objetos y estados modificados, se pudo aplicar polimorfismo de manera simple pero poderosa, cambiando la forma en que el sistema se comportaba según el tipo de acción sin necesitar herencias profundas.

En cuanto al diseño, los diagramas UML estático y dinámico proporcionaron una vista completa del sistema que ayudó en la planificación anticipada y en la documentación final. El diagrama de clases facilitó la representación gráfica de las conexiones entre entidades, la composición entre Pokémon y movimientos, y también la dependencia que existe entre las pantallas y los controladores. Los diagramas de secuencia, en cambio, ilustraron el flujo de mensajes a través de un turno de combate, el uso de objetos y la navegación general entre pantallas. Esto confirmó que UML es una buena herramienta para razonar sobre el comportamiento de la aplicación antes y después de codificar.

Se demostró que desacoplar la lógica de negocio de la presentación tiene beneficios cuando se emplea un modelo basado en eventos para comunicar el ControlBatalla con la interfaz gráfica. La interfaz no necesita conocer los detalles de cómo se calculan los daños o las prioridades, solo recibe eventos como ataques, cambios de estado, mensajes de texto o actualizaciones en las barras de vida. Esta resolución de diseño sentó las bases para que en el futuro se expandiera el sistema, a la vez que permitió añadir transiciones, animaciones y efectos de sonido sin modificar el motor de batalla.

La gestión estatal mundial y la permanencia fueron otros factores. El GameManager, que se codificó como Singleton, agrupó la información de la partida: jugador, equipo, dificultad, torre de entrenadores y la creación dinámica de la lista de jugadores. El uso de almacenamiento con shared preferences permitió guardar información entre ejecuciones, haciendo del proyecto algo muy cercano a lo que necesitaría una aplicación real para usuarios finales. También el AudioManager, otro Singleton, unificó la gestión de efectos y música, lo que permitió implementarlo en todas las pantallas.

Las pruebas en distintos dispositivos y la exportación a Android mostraron que el diseño es lo suficientemente flexible como para adaptarse a diferentes situaciones. Cosas como la legibilidad del HUD, el comportamiento de las animaciones, la respuesta del audio o la escala de la interfaz dejaron claro lo importante que es pensar desde el principio en que el software puede correr en múltiples plataformas. Las fases de testeo y corrección permitieron mejorar la experiencia de usuario, estabilizar el comportamiento de la lógica de batalla y refinar las fórmulas internas.

Finalmente, el proyecto logró demostrar que es posible y es necesario integrar diferentes temas del curso (POO, UML, manejo de errores, patrones de diseño, estructuras de control, entrada/salida de datos) para resolver problemas más complejos. Para que un simulador de batallas como el desarrollado funcione, se requiere de una implementación que siga los principios aprendidos, un diseño orientado a objetos y una documentación coherente. En definitiva, el trabajo realizado demuestra que la aplicación disciplinada de los conceptos teóricos permite construir sistemas robustos, comprensibles y preparados para crecer más allá de las necesidades iniciales.

425093384

## Referencias

- [1] Universidad de Valladolid Departamento de Informática. *Criterios de calidad de diseño modular: cohesión y acoplamiento*. Inf. téc. Tema 9: criterios de calidad para diseño modular. Universidad de Valladolid, 2004. URL: <https://www.infor.uva.es/~jvalvarez/teaching/ingenieria%20software/p tema9is1.pdf>.
- [2] UPM – ETSISI. “Programación Orientada a Objetos con Java”. En: (2013). Apuntes del curso de POO. URL: [https://www.etsisi.upm.es/sites/default/files/curso\\_2013\\_14/MASTER/MIW.JEE.POOJ.pdf](https://www.etsisi.upm.es/sites/default/files/curso_2013_14/MASTER/MIW.JEE.POOJ.pdf).
- [3] UNAM Facultad de Ingeniería. *Guía práctica de estudio 05: Abstracción y Encapsulamiento*. Documento educativo en línea. s.f. URL: [https://profesores.fi-b.unam.mx/annkym/LAB/poo\\_p5.pdf](https://profesores.fi-b.unam.mx/annkym/LAB/poo_p5.pdf).
- [4] Xavier Ferré Grau y María Isabel Sánchez Segura. *Desarrollo Orientado a Objetos con UML*. Manual sobre UML. Universidad del Valle de México, 2011. URL: <https://www.uv.mx/personal/maymendez/files/2011/05/umltotal.pdf>.
- [5] Universidad de Guadalajara – Pregrado. *Programación orientada a objetos*. Universidad didáctica en PDF. s.f. URL: [https://www.pregrado.udg.mx/sites/default/files/unidadesAprendizaje/programacion\\_orientada\\_a\\_objetos.pdf](https://www.pregrado.udg.mx/sites/default/files/unidadesAprendizaje/programacion_orientada_a_objetos.pdf).
- [6] M. González Harbour y M. Aldea. *Modularidad y diseño modular en programación orientada a objetos*. Capítulo “Modularidad y módulos predefinidos”. Universidad de Cantabria, 2016. URL: <https://ocw.unican.es/pluginfile.php/2330/course/section/2281/cap7-modularidad.pdf>.
- [7] BUAP – Unidad Académica de Informática. “Principios Básicos de la POO: Abstracción, Encapsulamiento, Herencia y Polimorfismo”. En: Material docente. s.f. URL: [https://ecosistema.buap.mx/forms/files/dspace-23/1\\_principios\\_bsicos\\_de\\_la\\_poo.html](https://ecosistema.buap.mx/forms/files/dspace-23/1_principios_bsicos_de_la_poo.html).
- [8] NetMentor. “Encapsulamiento en programación orientada a objetos”. En: (2019). Artículo en línea. URL: <https://www.netmentor.es/entrada/encapsulamiento-poo>.
- [9] Y. R. Ortiz. “La Programación Orientada a Objetos como metodología de desarrollo de software”. En: *Revista de Ingeniería y Computación* (2021). Artículo en español sobre fundamentos de POO. URL: <https://www.redalyc.org/journal/5315/531566552006/531566552006.pdf>.
- [10] M. F. Barcell / UNED. *Programación orientada a objetos – Tema 7*. Capítulo sobre diseño orientado a objetos, cohesión, acoplamiento y buenas prácticas. 2018. URL: [https://www.mfbarcell.es/docencia\\_uned/poo/tema07/capitulo\\_08.pdf](https://www.mfbarcell.es/docencia_uned/poo/tema07/capitulo_08.pdf).

- [11] CONALEP Veracruz. *Programación Orientada a Objetos – Módulo Profesional*. Documento educativo en PDF. 2021. URL: <https://www.conalepveracruz.edu.mx/iniciobackup/wp-content/uploads/2021/03/Programaci%C3%B3n-orientada-a-objetos-M%C3%93DULO-PROFESIONAL.pdf>.