Proyecto 03

Maria Jose Rodríguez Alvarez

Diciembre 2025

1 Marco Teórico

1.1 Dart

Dart es un lenguaje de programación moderno, desarrollado por Google, que combina el poder de la programación orientada a objetos con la facilidad y eficiencia de los lenguajes de programación basados en scripts [1].

1.2 Clases y Objetos

Una clase es una plantilla que define atributos y comportamientos, mientras que un objeto es una instancia concreta de dicha clase. En Dart, las clases permiten representar entidades del sistema como Pokémon, ataques o ítems, y facilitan la modularidad del código [2].

1.3 Herencia y Clases Abstractas

Son los mecanismos que permiten crear jerarquías donde una clase base define comportamientos comunes que son extendidos y concretados por subclases específicas. Esto facilita la reutilización de código y el polimorfismo [3].

1.4 Polimorfismo

El polimorfismo permite que un mismo método pueda tener comportamientos diferentes dependiendo del objeto que lo invoque. En el sistema de batalla, ataques de distintos tipos pueden sobrescribir métodos para calcular daño de forma específica.

1.5 Encapsulamiento

Consiste en ocultar los detalles internos de una clase y exponer solo lo necesario mediante métodos públicos. Esto permite gestionar de forma segura atributos como puntos de salud (HP) o velocidad, evitando modificaciones no controladas.

1.6 Constructores y Parámetros Nombrados

Los constructores permiten inicializar objetos y, con los parámetros nombrados, Dart facilita la legibilidad y claridad al crear entidades complejas como ataques o Pokémon con múltiples atributos.

1.7 Colecciones y Listas

Dart proporciona listas dinámicas que permiten almacenar colecciones de objetos, como los movimientos de un Pokémon o los ítems disponibles del jugador.

1.8 Manejo de Estados y Lógica de Negocio

La lógica de turnos, aplicación de daño, cambios de estado y verificación de condiciones de victoria se implementan mediante métodos dentro de las clases del sistema. El uso de estructuras condicionales y funciones permite controlar cada etapa del combate.

1.9 Uso de la Biblioteca `dart:math`

Se emplea para generar valores aleatorios en ataques, efectos secundarios, o variaciones en el daño, mediante la clase `Random`.

1.10 Flutter

Flutter es un *framework* basado en widgets, componentes visuales inmutables que describen la estructura de la interfaz. Cada pantalla o elemento gráfico (selección de ataques, barra de vida, menús) se construye con widgets [4].

Stateful Widgets. Se utilizan cuando la interfaz debe reaccionar a cambios en el estado interno, como la actualización del HP, la activación de efectos de estado, o el avance del combate por turnos.

Gestión de Estado. El método `setState()` permite actualizar variables y reflejar cambios en la interfaz. Esto es fundamental para mostrar en tiempo real la vida restante, los mensajes de combate o la disponibilidad de acciones.

1.11 Mecánicas de Combate Pokémon

HP (Health Points): Es la condición física del Pokémon, representada con un valor numérico. Estos son reducidos normalmente mediante los ataques del oponente en combate, los efectos del veneno, las quemaduras, o varios climas entre otros medios. [5]

Velocidad: La velocidad es la propiedad del Pokémon de atacar, antes o después, que el oponente. A la hora de atacar el Pokémon con un mayor valor de velocidad, por lo general, siempre atacará primero.

Efectividad por Tipos. Cada tipo de ataque puede ser más o menos efectivo dependiendo del tipo del oponente. Esta efectividad se expresa como

multiplicadores: neutral (1), supereficaz (2), poco eficaz (0.5) o sin efecto (0) [6].

Estados Alterados. Efectos como envenenamiento, quemadura, parálisis o congelación influyen directamente en el combate, ya sea reduciendo vida por turno o afectando la capacidad del Pokémon para atacar.

1.12 Patrones de Diseño

Modelo–Vista–Controlador (MVC). El patrón MVC separa la lógica de negocio (modelo), la interfaz de usuario (vista) y el flujo de datos (controlador). En Flutter, esta separación se logra mediante la distribución de clases y widgets, permitiendo un proyecto organizado, escalable y mantenible [7].

References

[1] Google. **A Tour of the Dart Language**. https://dart.dev/language

[2] Dart Language Documentation. **Classes**. Nov. 2025. URL: https://dart.dev/language/classes

[3] Dart Language Documentation. **Object-Oriented Programming in Dart**. Nov. 2025. URL: https://dart.dev/language/extend

[4] Flutter Documentation. **Architectural Overview**. Dic. 2025. URL: https://docs.flutter.dev/resources/architectural-overview

[5] Wikidex. **Mecánica - Características**. Nov. 2025. URL: https://www.wikidex.net/wiki/Caracter

[6] Wikidex. **Mecánica - Tipos**. Nov. 2025. URL: https://www.wikidex.net/wiki/TipoEfectividades$_d e_l os_t ipos$

[7] Rhm, F. (2023). **Understanding MVC Architecture in Flutter: A Comprehensive Guide with Examples**. Medium. URL: https://medium.com/@Faiz$_R hm/understanding-mvc-architecture-in-flutter-a-comprehensive-guide-with-examples-5d1a372c7eaf$