Trabajo Práctico n.ͦ 3

Algoritmos y Programación III

PokeBatalla

| Penedo Nicolas …………………………….. 109362    Juan Cruz Robledo Puch…………………….. 106164    Franco Singh.………………………… 108327 |
| --- |

Fecha de entrega: 13 / 12 /2023

### 

### Introducción:

Como lo aclara el enunciado del TP, El juego es una partida entre 2 jugadores que se enfrentan por turnos a turno hasta que uno alcance la victoria derrotando los pokemones del rival o que el mismo se rinda. Cada uno tiene un máximo de 6 Pokemones y hasta 4 Items. Cada Pokémon tiene un estado, tipo y demás estadísticas. Son duelos 1 a 1 en los que pueden atacarse mediante habilidades, utilizar ítems, posicionar otro pokémon sobre el campo o rendirse.

### Hipótesis y Supuestos:

Como se mencionó en clase, los pokemones e ítems iniciales por el momento, son totalmente elegidos arbitrariamente mediante “Generador”. Sin embargo, especulamos y preparamos el programa para poder recibir la información de pokémons e ítems de cada uno de los jugadores mediante JSON, CSV, etc. Al ser un juego con tantas características y partes, tuvimos que tomar distintos supuestos:

* Todos los pokemones empiezan sin estado.
* Si el jugador utiliza una poción que no se puede usar, este pierde el turno.
* el juego al iniciar se le asigna un clima ‘sin clima’, el cual no afecta a los pokemones

Diseño:

Durante el diseño procuramos respetar los pilares de la programación orientada a objetos y programación funcional. “Juego” da inicio y “Generador” se encarga de crear Pokemones e items a usar por el Jugador. El propio juego determina y maneja los turnos durante toda la partida. Nos pareció interesante que Jugador sea el dueño de los Pokemones e Items y así tener control absoluto sobre ellos. Pokemon conoce sus propios atributos y su tipo. Esta es una clase abstracta que implementa un método que utilizan sus clases hijas para saber la efectividad contra otros tipos. Tuvimos un debate sobre qué hacer con las habilidades y finalmente decidimos que cada Pokémon conozca las suyas ya que cada uno tiene diferentes habilidades. Tambien cada Pokemon es representado por un Sprite que lo caracteriza

Priorizamos respetar los principios SOLID, básicamente el principio de abstracción y responsabilidades únicas son el tronco del diseño del juego.

Para los Ítem se decidió crear una clase abstracta Items para hacer uso de herencia, ya que los diferentes objetos que plantea el enunciado comparten un comportamiento con el que afectan al estado de un pokémon, pudiendo implementar polimorfismo con el método aplicarItem(). También se planteó lo mismo para el caso de los ítems Modificadores

Para las habilidades también se utilizó Herencia con Polimorfismo en el método usarEnPokemon(). Cada nueva habilidad sera un nuevo objeto que extienda de Habilidad

Para los tipos se implementó algo parecido al patron Strategy con un diccionario (HashMap) para cada uno de los tipos, en donde se guarda cada uno de los multiplicadores de la tabla y sus respectivas relaciones. Cada subclase de Tipo proporciona su propia implementación de “efectividad” para el método getEfectividad(String tipo), lo que permite obtener la efectividad de un tipo específico contra el tipo de estrategia representado por esa subclase. No llega a ser un patrón Strategy del todo debido a la falta de una interfaz común para todas las estrategias y la ausencia de un mecanismo para intercambiar estrategias dinámicamente.

Los estados ahora son clases concretas que implementan la interfaz EstadosPokemon. cada objeto se encargará de implementar el funcionamiento de cada método especificado en EstadoPokemon para su posterior uso en forma Polimórfica, siendo el mas importante AplicarEstado. Estos estados se visualizan en forma de iconos en la UI

Los climas se integraron al programa mediante la clase Clima, la cual se ocupa de mejorar el daño del ataque de un pokemon y, en el caso de que sea un clima que reste vida, se ejecute este comportamiento. Dicho clima vive dentro del campo, en donde además de manejar los ataques entre pokémons, se añade la mejora de ataque y el daño añadido según el clima. Los climas estan representados graficamente por imagenes en la UI.

Tambien se tomo la decicion de deserializar los climas desde un archivo JSON para asi agregar cualquier clima que se desee

Para la parte de las Persistencia se crearon objetos Parcer que se encargan de recopilar los datos de los archivos .JSON e integrarlos en el juego. Para el caso de las habilidades se creó una Clase RepositorioHabilidades que administra las Ids de cada habilidad

La UI/UX esta integrada mediante el patron MVC con el cual decidimos hacer un controlador por cada pantalla que aparece en el juego, siendo estas la de Batalla, Elegir Pokemon, Inicio, Fin de Partida y Mochila. Cada una posee un archivo .fxml que contiene el estilo de los elementos que se muestran en la visualización

En PantallaBatallaController se controlan las animaciones, habilidades, mensajes y datos de la batalla en la pantalla principal. Tambien se visualizan los pokemones en el campo, sus estados y la vida de los mismos. El jugador puede elegir mediante una botonera entre atacar, cambiar de pokemon, huir y usar un item. cada una de estas opciones abre una nueva pestaña controlada por su respectivo controlador

En PantallaCambiarPokemon, como el nombre indica, se encarga de controlar el cambio de los pokemon mediante una ventana emergente. Esta ventana mostrara la vida actual de cada pokemon, su nombre y los estados que tiene

PantallaDerrotaController hace que en una ventana nueva se muestre, una vez finalizado el combate, quien fue el perdedor del encuentro. Tambien te permitirá volver a jugar la partida

PantallaItemController se encarga de controlar el manejo de los items y la aplicacion de los mismos. Al igual que los otros controllers, abrira una nueva ventana con la interfaz para cambiar de item, mostrando los nombres y la cantidad de cada uno. Tambien al elegir un item, se abrira un menu oara seleccionar a cual pokemon se le quiere aplicar deicho item

Patrones de Diseño:

Para poder usar la lógica de las habilidades con los pokemon, utilizamos el patron Visitor para usarlo desde un Campo, que conoce a los pokemones e interactúa con una habilidad.

También se implementó el uso de FactoryMethod para la creación de los distintos tipos leídos en el parser. Se usó MVC para gestionar y administrar los eventos y las propias comunicaciones. Esto ayudó a la organización y delegar mejor las responsabilidades a otras clases.

### Conclusión:

En conclusión, utilizar programación orientada a objetos y programación funcional nos ayuda a construir una solución informática consistente, escalable y fuerte. Poder pensar en delegar y ordenar las responsabilidades de cada clase ayuda a un código más ordenado, limpio y menos repetitivo al usar polimorfismo por ejemplo para escalar un producto a algo mucho más grande de lo esperado inicialmente. También destacar que el uso de la serialización es sumamente importante ya que permite mantener un registro de los objetos que viven en el programa y también permite configurar un estado inicial para algunos elementos. Otro aspecto importante es el buen manejo de la UI/UX que permite una relación menos estricta con el programa, facilitando su uso y manejo. Y por último los Test, siendo estos de suma importancia para poner a prueba la implementación de nuestro codigo, eliminar errores y mantener un funcionamiento estable del programa, además de servir como documentación.