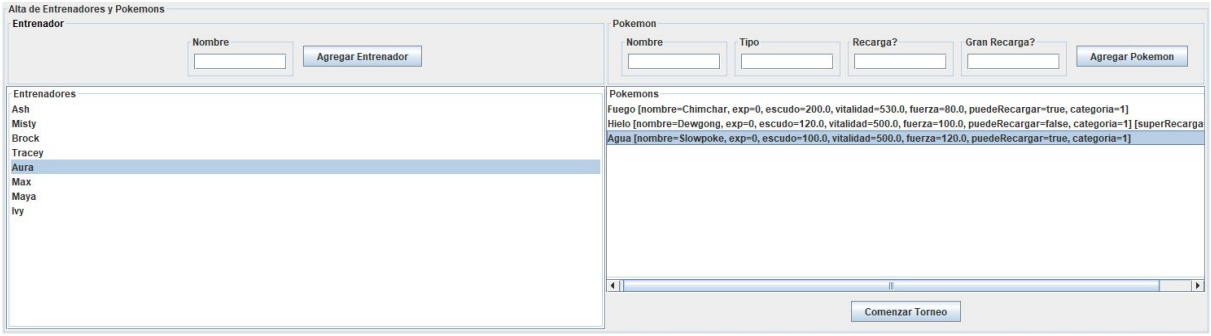
INFORME GIMNASIO POKEMON PARTE 2 - GRUPO 11

Integrantes: Coyos, Joel; Marinucci, Lorenzo; Palomeque, Lucía Lourdes

---

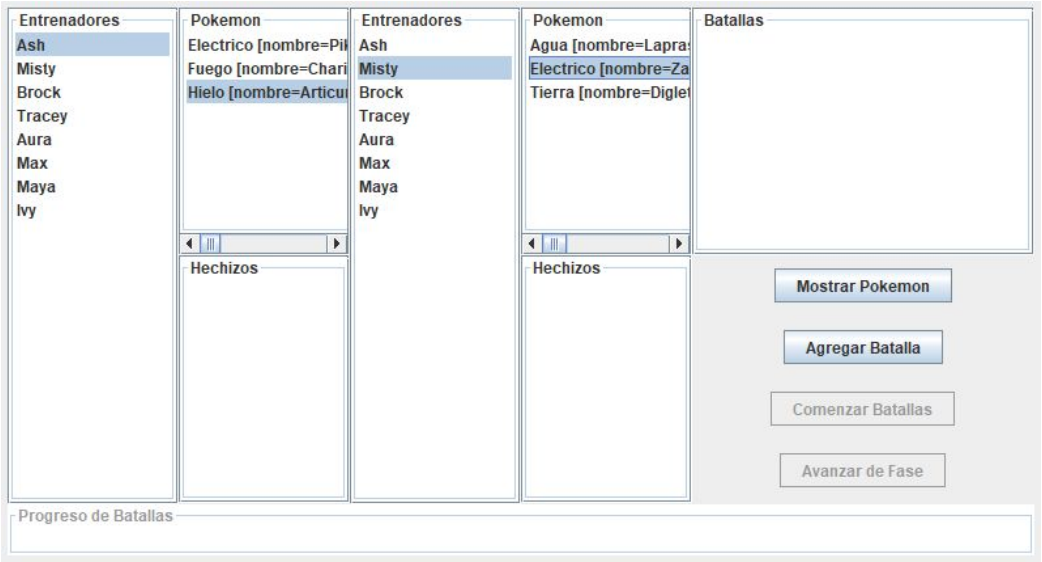
FUNCIONAMIENTO DEL TORNEO

Al ejecutar el programa por primera vez, se abrirá la ventana de altas, en la cual agregamos entrenadores y sus pokemones.

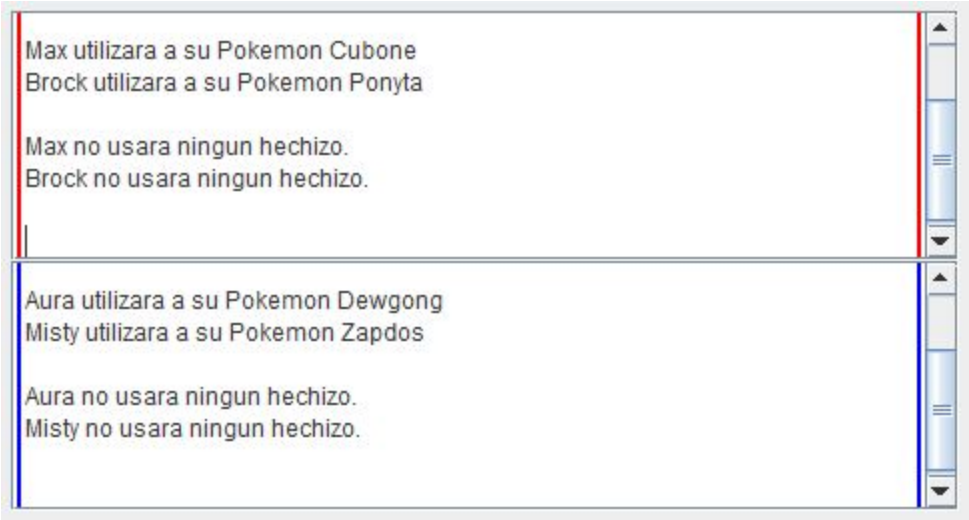


Para agregar un entrenador sólo necesitamos un nombre. Cuando queremos agregar un pokémon a un entrenador, seleccionamos uno, un nombre, un tipo válido (Agua,Tierra, Eléctrico, Fuego o Hielo), si tiene una recarga (sí o no), y en el caso de un Pokemon tipo Hielo, si tiene gran recarga (sí o no).

Cuando tengamos 8 entrenadores podremos presionar el botón Comenzar Torneo, y se abrirá la ventana del torneo. A su vez, el resto de los botones permanecerán deshabilitados.



Aquí podemos agregar batallas. Después de seleccionar dos entrenadores, presionaremos "mostrar pokemon" para ver los pokemon de cada uno. Además, si el entrenador todavía posee hechizos disponibles, se mostrarán los mismos. Es posible seleccionar un hechizo (Tormenta, Viento, Niebla) o no seleccionar ninguno. Después de realizar las selecciones, presionamos "Agregar Batalla". Luego de crear todas las batallas, podremos comenzar a ejecutarlas presionando "Comenzar Batallas". Esto abrirá la ventana de arena, que mostrará la evolución de las mismas.



Cuando terminen todas las batallas, podremos presionar el botón “Avanzar de Fase”, el cual abrirá una ventana de torneo con los entrenadores que ganaron en la fase anterior. De esta forma se repetirá hasta llegar a la final. Al finalizar el torneo, saldrá un cartel indicando el ganador. Además, será posible realizar una nueva iteración del programa.

Después de cada fase se guardará el estado del torneo, por lo que si lo cerramos en medio de la ejecución, podremos recuperar el último estado.

CONCURRENCIA

Arena: contiene una batalla e implementa la interfaz IStateArena. Contiene un método ejecutarFase() que dependiendo de su estado (Preliminar, Batalla, Finalizacion o Limpieza; en ese orden) tendrá un comportamiento distinto. Las arenas se encuentran guardadas en el Torneo y la asignación de las mismas se realiza con un método synchronized en el Torneo.

Batalla: se extiende de la clase Thread, contiene todas las variables necesarias para realizar una batalla: ambos entrenadores, los pokemones que usarán y sus hechizos.

Al realizar el método start, la batalla tratará de adquirir una arena. En caso de que haya una disponible, ésta será asignada a la batalla. De darse el hecho de que no haya una arena disponible, el Thread quedará en espera. Tras conseguir la arena, se procede la ejecución de los métodos correspondientes a la misma. Al finalizar el último método (Limpieza), se libera el recurso compartido y se notifica al resto de Threads.

Torneo: tendrá los entrenadores que se inscribieron al torneo, los participantes que siguen en juego, los enfrentamientos que han ocurrido de momento, una lista de arenas y un iEtapa, que puede ser Alta (creación de entrenadores y pokemones) o Desarrollo (selección y transcurso de las batallas).

El método asignarArenas() es el encargado de otorgar una arena a aquella batalla que la solicite. Al revisar si la lista de arenas está vacía, se tienen dos posibles situaciones: en el caso que lo esté, se espera a que se notifique la liberación de una arena, y en caso contrario se toma una arena del ArrayList y se la asigna a la batalla que la solicitó.

NUEVAS INTERFACES

IEtapas

Alta: el método faseCompletada() retorna true si hay ocho entrenadores y cada uno de ellos tiene más de un pokemon. Al avanzar de fase se copian los entrenadores del torneo a los participantes actuales para no perder los entrenadores originales, y se cambia el estado del torneo a Desarrollo.

Desarrollo: el método comenzarBatallas() toma todas las batallas que fueron asignadas en la ventana del Torneo y comienza a ejecutar los threads. En esa etapa, el método faseCompletada() retorna true si la cantidad de participantes actuales es la mitad de la ronda anterior, es decir, si se terminaron de ejecutar las batallas.

IStateArena

ArenaPreliminar: se presentan los entrenadores, sus pokemon y sus hechizos.

ArenaBatalla: se realizan las acciones correspondientes a la batalla. De haber hechizos, se aplican, y luego se procede a realizar los ataques.

ArenaFinalizacion: se calcula el puntaje de cada Pokemon y se determina el ganador, este se agregara a la lista de participantes actuales.

ArenaLimpieza: se realizan tareas de limpieza para la siguiente batalla y la arena es liberada.

PATRONES

Patron MVC

Controlador

El controlador funciona como un intermediario entre la vista y el modelo del programa. La vista se comunica con el controlador mediante eventos, los cuales son procesados en el método actionPerformed. En base al evento pasado como parámetro, se realizan acciones que tendrán impacto en el torneo.

El modelo interactúa con la vista mediante la notificación a su observador, el controlador. Mediante el envío de mensajes, el controlador dictamina que

acciones debe realizar la vista correspondiente. Esto puede incluir el cierre de ventanas, la apertura de otras, la actualización de logs, etc.

Modelo

El modelo encapsula el estado de la aplicación y realiza la funcionalidad de la misma. Incluye las clases previamente utilizadas en la primer parte, además de la inclusión de otra nuevas como las Batallas, las Arenas, entre otras.

Vista

Representa la parte visual de la aplicación. Es la que recibe los datos ingresados por el usuario y muestra los progresos que va realizando el Torneo. Se compone de 3 posibles ventanas.

VentanaAlta: es aquella ligada a la etapa de Altas del Torneo. Permite el agregado de entrenadores hasta un determinado límite (8), además del añadido de sus respectivos Pokemones. El botón "Agregar Entrenador" verifica que el campo nombre no esté vacío. En caso de no estarlo, se procede a enviar el evento de agregación al controlador. Lo mismo ocurre con el botón "Agregar Pokémon", sólo que aquí se revisan los campos correspondientes a su creación.

Una vez cargados los 8 entrenadores y que éstos posean mínimo un Pokemon cada uno, se puede dar paso a la ventana de Torneo, presionando el botón "Comenzar Torneo". Este botón, mediante su respectivo evento asociado, hará que el Controlador dé inicio a la fase de desarrollo del Torneo.

VentanaTorneo: en esta ventana se permite la creación de batallas. El usuario debe seleccionar dos entrenadores distintos de las listas y luego pulsar el botón

“Mostrar Pokemon”. Esto llevará a que los Pokemones que tenga disponibles el entrenador sean mostrados en la ventana. A su vez, de tener hechizos disponibles, se podrá seleccionar aquel que el entrenador desee utilizar (puede no utilizar ninguno). Una vez definidas todas las batallas correspondientes a la ronda, el usuario debe pulsar el botón “Comenzar Batallas”, el cual transmitirá un evento al Controlador, que se encargará de invocar al método correspondiente dentro del Torneo. Al dar inicio a las batallas, se abrirá una VentanaArena. Cuando finalicen todas las batallas, el usuario podrá avanzar de fase, lo cual mostrará una nueva VentanaTorneo (si el Torneo no ha finalizado) o un mensaje que anuncia al ganador.

VentanaArena: esta ventana mostrará las acciones que suceden en cada Arena. La misma se divide en 2, y cada Arena mostrará en su correspondiente sección los hechos que van sucediendo en la batalla. Esto incluye el ingreso de los participantes, los ataques y hechizos realizados, los resultados finales, la premiación al entrenador ganador, etc.

Patron Observer

El patrón se aplica para que determinados objetos puedan notificar al controlador de los cambios que ocurren en el modelo, y que estos puedan ser mostrados en la interfaz gráfica. Contamos con una clase observer, el Controlador, y dos clases Observables, el Torneo y las Arenas.

Las Arenas notifican los hechos que ocurren en cada batalla ejecutada en las mismas. Mediante el uso del método notifyAll, se envían las salidas por pantalla que se requieren.

El Torneo también es observado por el controlador, y el mismo realizará notificaciones cuando su estado cambie. Estas notificaciones son realizadas para ligar las ventanas correspondientes con su respectiva fase. Por ejemplo, al cambiar de una fase de Alta a Desarrollo, se instruye al Controlador para que cierre la ventana de altas y abra la ventana correspondiente al Torneo.

SERIALIZACIÓN

TorneoSerializable: debido a que el Torneo aplica el patrón Singleton, se decidió por la realización de una clase intermedia que almacene el estado actual del Torneo. Esta clase, mediante el uso de getters, adquiere los atributos que posee el Torneo al momento de realizarse la serialización. Dentro de sus atributos contiene a los Entrenadores, los participantes actuales, las arenas, entre otros. Este es el objeto que es serializado en un archivo XML, y el que también es recuperado al realizar el Deserialize. Para el pasaje de atributos de la clase leída al Torneo, se utilizan los setters asociados a cada uno.

SerializeToXML: esta clase posee un método estático que realiza la serialización de un TorneoSerializable. Previamente, el método realiza el correspondiente copiado de los atributos.

DeserializeFromXML: la clase realiza la deserialización del TorneoSerializable guardado en el archivo XML. Tras realizar la lectura, se procede a copiar los atributos al Torneo mediante sus setters.

COMPLICACIONES

Cómo realizar la serialización de una clase Singleton: la clase Torneo utiliza el patrón Singleton. Esto fue realizado de esta manera, ya que se accede al Torneo

desde numerosas partes del programa y esto hubiera implicado el traspaso de la referencia al Torneo en numerosos llamados a métodos. La complicación que trajo la utilización de este patrón es que para realizar la serialización se requería un método de nombre setInstance. La existencia de este método, puede llevar a la existencia de varias instancias de Torneo en el programa. Si bien era posible establecer ciertos mecanismos de seguridad para que esto no suceda, finalmente se decidió por la creación de una clase intermedia (TorneoSerializable) para ayudar con la serialización.

Threads que escriban en logs: debido a que los Threads de batalla se ejecutan simultáneamente, el programa mostraba un comportamiento errático a la hora de realizar una escritura simultánea sobre los logs de las ventanas. Para la solución de este problema, se intentó hacer que los métodos de escritura sean synchronized. Sin embargo, esto no solucionó los inconvenientes. Tras una investigación sobre el problema, se llegó a que el Swing y la concurrencia en conjunto, traían problemas. Al parecer, más allá de establecer el método como synchronized, el textArea no termina de actualizarse correctamente al finalizar el método, haciendo que el próximo Thread que busque escribir lo haga en una posición incorrecta de la misma. Esto lleva a que el programa se congele, siendo imposible la realización de las acciones subsiguientes.

Una solución barajada fue la creación de ventanas particulares para cada Batalla y además, quitar el log de la ventana del Torneo. Debido a que visualmente era preferible una ventana conjunta, se trató de buscar otra solución para esta situación. Se llegó a que la escritura se realizaba en forma correcta utilizando la siguiente porción de código:

```
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
                textArea.append(string);
        }
})
```

Esto es necesario para garantizar la correcta asociación de la concurrencia con las estructuras de datos que posee Swing, ya que estas no funcionan correctamente de forma concurrente. Este método provee una solución que garantiza el correcto funcionamiento de los Threads.

Selección de log en el cual escribir: se buscaba que en un pequeño log en la ventana del Torneo se anuncien los comienzos de las diferentes batallas. Esto era responsabilidad de las Arenas, las cuales notifican al Controlador de los hechos asociados a cada batalla. Debido a que también se realiza la escritura en la ventana de Arenas, debíamos encontrar una forma para que el Controlador pueda diferenciar en qué ventana escribir. Para ello, las Arenas pueden comunicarse de dos formas distintas. Ambas envían un array de object, donde el segundo parámetro será el mensaje a enviar por el log. El primer parámetro, puede ser una Batalla o nulo. Esto se realizó ya que para mostrar por el log de la ventana del Torneo, se necesitaba la Batalla. De esta forma, se analiza la primera posición del vector, si es una referencia null se escribe en la ventana de Arenas, y si es distinta de null (es una Batalla) en la ventana del Torneo.