Universidad Nacional de La Matanza

DEPARTAMENTO DE INGENIERÍA

PARADIGMAS DE PROGRAMACIÓN

TP – Héroes y Villanos

Integrantes:

- Aguilera, Emanuel.
- Arab, Santiago.
- Barcia, Matias Alejandro.
- Sanz, Eliseo Tomas.
- Zamudio, Verónica.

Índice

Introducción

Este informe presenta el segundo trabajo práctico de Paradigmas de Programación, desarrollado para la creación de un videojuego - **"Héroes y Villanos"-**. El objetivo del proyecto es diseñar e implementar un juego que permita el combate entre personajes (héroes o villanos), entre Ligas, y Personaje contra Liga. A continuación, sé detalla la Metodología, las estrategias utilizadas para el proyecto, los archivos, la distribución de tareas y la conclusión del trabajo.

Metodología

Para la elaboración del videojuego se han seguido las siguientes etapas:

1. Diseño:

- Se han definido las mecánicas de juego y los objetivos.

2. Implementación:

- Se ha implementado la lógica del juego, se han incluido los métodos necesarios para la realización del combate, la interacción entre competidores, no habiendo ninguna restricción en los tipos de Competidores a la hora de la Lucha. No hay sistema de puntuación, el programa solo nos indica quién es el vencedor.
- Se ha creado la clase InterfazDelUsuario, que representa el Menú de Juegos. A través de ella se podrá elegir las diferentes opciones que se brinda al Usuario.

3. Pruebas:

- Se ha realizado un proceso de pruebas para detectar y corregir errores en el código.

Estrategias utilizadas

A continuación, se detallan las estrategias de las diferentes secciones relevantes para la creación del código.

1. **Lectura y escritura de archivos**:
   Creación de la clase Archivo para la lectura y escritura de los archivos personajes y ligas, con la información almacenada en mapas.
2. **Composite:**
   Decidimos utilizar el patrón de diseño Composite, ya que este patrón permite trabajar con los personajes y las ligas (composición de personajes) de una manera uniforme, sin la necesidad de distinguir los objetos individuales de las composiciones.
3. **Realización de combates:**
   El usuario podrá optar entre Liga vs Liga, Personaje vs Liga o Personaje vs Personaje.

Descripción de archivos

A continuación, se detallan todos los archivos utilizados para el proyecto.

- **AdministracionDeCombates.java:**
  Esta clase contiene un menú con 3 opciones para poder realizar el combate entre personajes, combate entre ligas y el combate entre personaje y liga.
- **AdministracionDeLigas.java:**
  La clase contiene un menú con 4 opciones para poder realizar: la carga desde el archivo de Ligas, la creación de Ligas, Listar las Ligas y guardar en archivo todas las Ligas.
- **AdministracionDePersonajes.java:**
  Esta clase contiene un menú con 4 opciones para poder realizar: la carga desde el archivo de Personajes, la creación de Personajes, listar los Personajes y guardar en archivo todos los personajes.
- **Archivo.java:**
  Clase dedicada a la gestión de entrada y salida de Ligas y Personajes.
- **Característica.java:**
  Clase Enum que posee las características que contiene un Competidor: VELOCIDAD, FUERZA, RESISTENCIA, DESTREZA.
- **ExceptionMiembroIncluídoEnLiga**
  Clase derivada de Exception encargada del manejo de la excepción cuando un personaje intenta combatir contra una liga que contiene a este personaje.
- **CaracteristicaInexistenteException.java:**
  Clase derivada de Exception encargada del manejo de la excepción por Característica inexistente.
- **CaracteristicaNegativaException.java:**
  Clase derivada de Exception encargada del manejo de la excepción por Característica Negativa.
- **Competidor.java:**
  Clase abstracta del cual se derivan Liga y Personaje. Contiene métodos abstractos que van a ser implementados por las clases hijas. Acá se encuentra el método esGanador(), el cual nos permite saber quién es el ganador según la característica proporcionada.
- **CompetidorComparator.java:**
  Esta clase nos permite la comparación por característica.
- **CompetidoresSingleton.java:**
  En Java, un Singleton es un patrón de diseño que garantiza que una clase tenga solo una instancia y proporciona un punto de acceso global a esa instancia. Esto significa que una vez que se crea la instancia, se reutiliza en lugar de crear una nueva cada vez que se solicita.
  La implementación típica de un Singleton en Java implica lo siguiente:
  - Un constructor privado para evitar la creación de instancias directamente desde fuera de la clase.
  - Un método estático que devuelve la única instancia de la clase.
  - Una variable estática privada que almacena la única instancia de la clase.

- **CompetidorTest.java:**
  Se realizan los test de prueba para las características del Competidor, a través del método esGanador().
- **Heroe.java:**
  Clase derivada de Personajes.
- **InterfazDelUsuario.java:**
  Clase principal para el manejo del juego.
  Método menu(): Muestra un menú principal con varias opciones para administrar personajes, ligas, realizar combates, generar reportes y salir del juego. Según la opción seleccionada por el usuario, se ejecuta el correspondiente método.

  Método obtenerOpcion(): Solicita al usuario que ingrese una opción válida dentro de un rango específico. Esta función asegura que la entrada del usuario sea un número dentro del rango especificado.

  Método mostrarCompetidor() y _mostrarCompetidor(): Muestran información detallada sobre un competidor, incluyendo sus características como nombre, tipo (Héroe/Villano), nombre real y atributos como velocidad, fuerza, destreza y resistencia. Si el competidor es una liga, también muestra los competidores que forman parte de ella.

  Métodos mostrarPersonajes() y mostrarLigas(): Muestran una lista de todos los personajes y ligas registrados en el juego, respectivamente. Utilizan los métodos de mostrar competidor para mostrar la información detallada de cada uno.

  Método guardarEnArchivo(): Guarda la información de los personajes o ligas en un archivo de texto. El nombre del archivo se determina según el tipo de información a guardar (personajes o ligas). Utiliza un BufferedWriter para escribir en el archivo.

- **Liga.java:**
  Clase derivada de Competidor encargada del manejo de Ligas.
- **LigaInexistenteException.java:**
  Clase derivada de Exception encargada del manejo de la excepción por la existencia o no de una Liga.
- **Personajes.java:**
  Clase derivada de Competidor encargada del manejo de personajes individuales.
- **Reportes.java:**
  Clase dedicada al guardado de archivos de reportes.
- **Villano.java:**
  Clase derivada de Personajes.

Distribución del trabajo

Desde un inicio se ha definido el rol y responsabilidad de cada integrante. A medida que se completaron las tareas asignadas, cada integrante subía las partes del código a la plataforma GitHub con el propósito de poder modificar y actualizar el proyecto. Al mismo tiempo se han realizado pruebas y al completarse todo el trabajo se realizó una validación general de todo el proyecto.

Tareas:

- Composite: Eliseo y Matias
- Reportes: Emanuel
- UML: Santiago y Veronica
- Archivos Entrada/Salida: Eliseo y Matias
- Tests y casos de pruebas: Veronica, Eliseo y Matias
- Ligas: Eliseo y Matias
- Main: Veronica, Eliseo y Matias
- Combates: Eliseo y Matias
- Informe: Veronica y Emanuel

Conclusión

El objetivo principal del trabajo práctico fue afianzar los conceptos de la programación orientada a objetos, uso de recursos, aplicación de conceptos, pruebas unitarias, organización y distribución de trabajo en distintas áreas. Se logró desarrollar satisfactoriamente un sistema funcional que cumple con los requisitos planteados, demostrando las habilidades y compresión del equipo en el desarrollo de software. Consideramos que para estos casos, el uso de un lenguaje orientado a objetos facilita mucho las cosas, ya que hacer el mismo proyecto en un lenguaje como, por ejemplo, C, hubiese sido muy complicado.