

TP1

Los mensajes de Herta



El contexto:

Hoyoverse, empresa de videojuegos en auge por sus títulos anteriores, está contratando programadores para su nuevo proyecto: **Honkai Star Rail**. Es un juego de combate por turnos, basado en el universo creado en las entregas anteriores.

Ustedes tuvieron la suerte (o desdicha) de ser contratados para el desarrollo de la versión de salida 1.0, por lo que su tarea a lo largo del desarrollo será programar sistemas claves para diferentes áreas del juego.

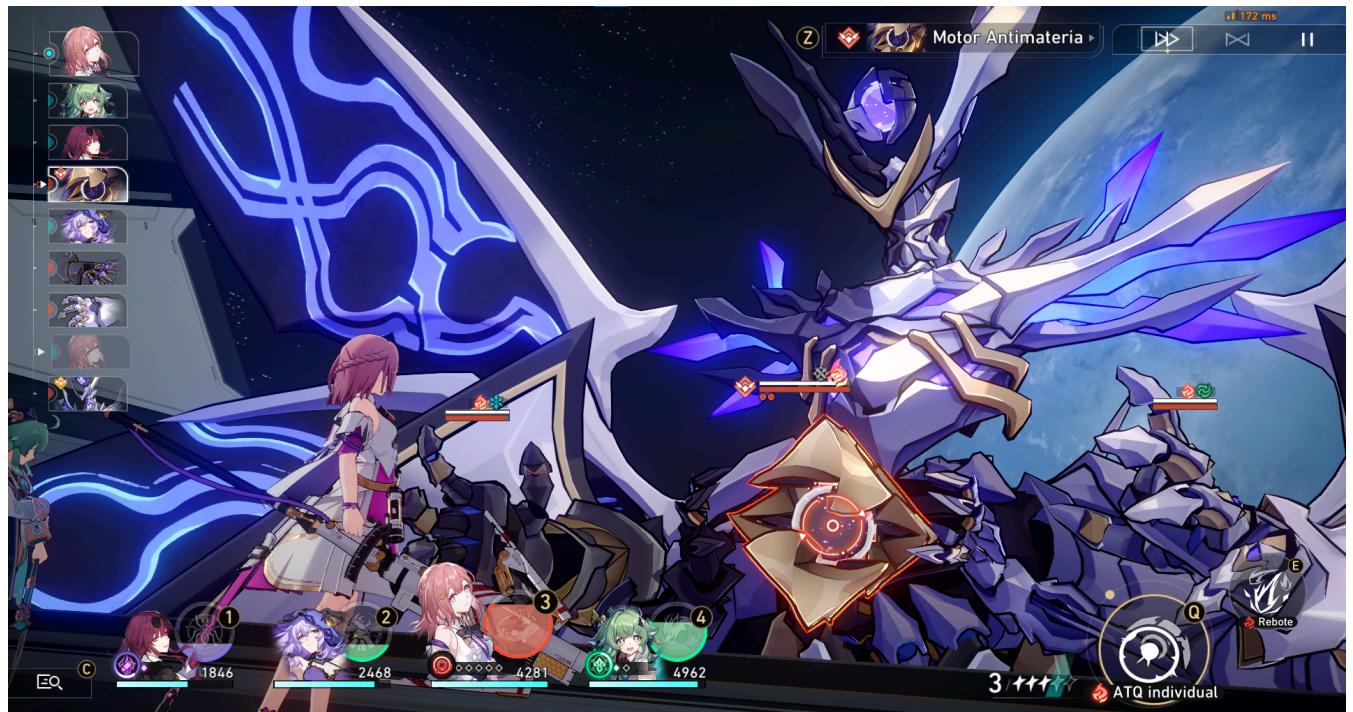


Imagen 1: pelea contra [Doomsday Beast](#), uno de los primeros jefes del juego.

La primera tarea:

El director creativo del juego consideró que sería una buena funcionalidad para incorporar un sistema de chats (ficticios) que el jugador enviará y recibirá de los diferentes personajes que se cruzará a lo largo de la historia. Esto permitirá ampliar las historias principal y secundarias del universo.

Uno de los personajes es **Herta**, Miembro nº 83 del Círculo de Genios y dueña de la Estación Espacial. Es de los primeros aliados que el jugador se encuentra en su viaje. Su personalidad es acorde, como se imaginaran, a alguien “miembro del Círculo de Genios”, por lo que sus chats reflejarán su arrogancia y desinterés.

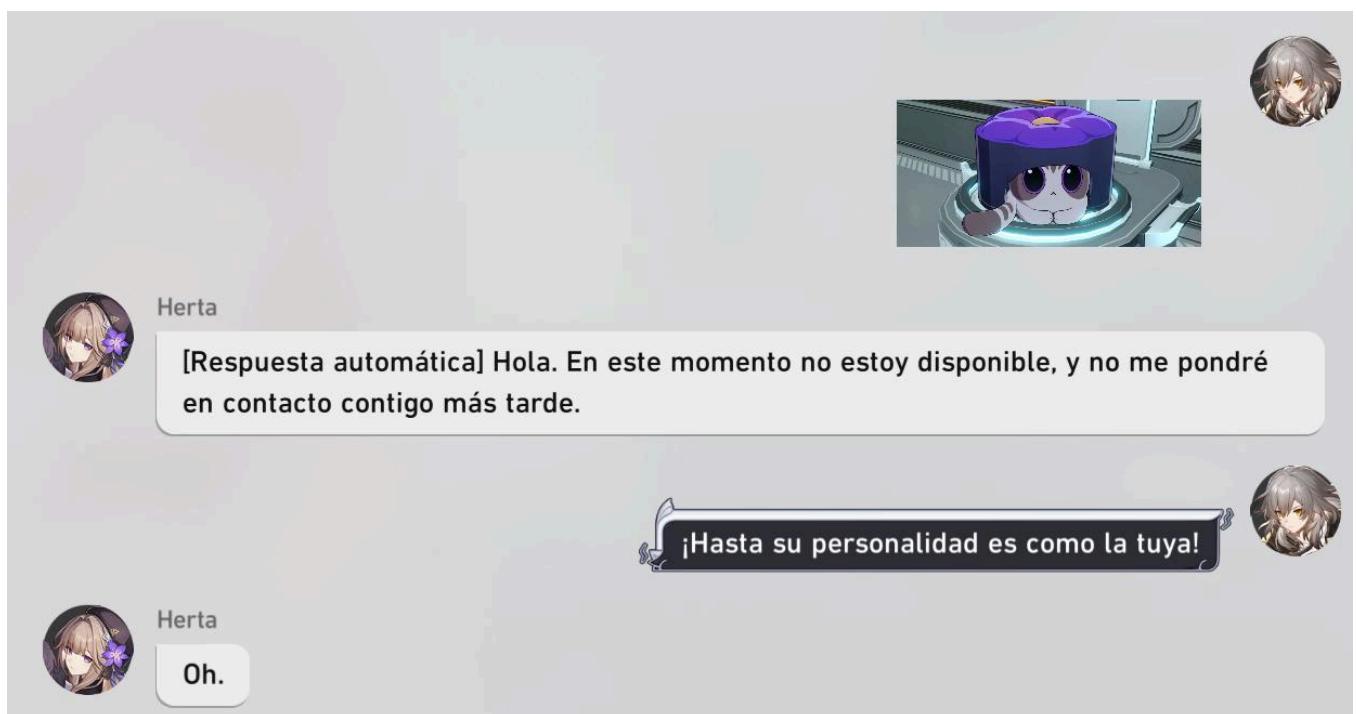


Imagen 2: Ejemplo de una interacción con Herta en el juego.

Funcionalidades a implementar:

Concretamente, nuestra primera tarea será implementar la clase **Herta**, que luego se instanciará y responderá a los mensajes que le enviemos. La implementación del sistema deberá aplicar el paradigma orientado a objetos.

Los métodos a implementar son:

1. `void responder(std::string mensaje):`

Este método recibirá el mensaje que el jugador le quiera enviar a Herta e imprimirá por pantalla su respuesta. Las posibles respuestas serán desarrolladas más adelante.

Pueden agregar métodos y atributos **privados** y constructores **públicos** a la clase **Herta**. El único método público (además de los constructores) será *responder*. También pueden crear más clases.

Adicionalmente, se deberá escribir un *main.cpp* sencillo que lea los mensajes del usuario por teclado y se los envíe a un objeto **Herta**. El objetivo de este código será probar manualmente el comportamiento correcto de la clase y su implementación queda a criterio del estudiante.

Comportamiento de responder.

Herta responderá a nuestros mensajes de formas diferentes, dependiendo de las siguientes condiciones:

1. Herta siempre responde de la misma forma al primer mensaje que le envíemos, independientemente de cual sea. El mensaje para este caso es:
[Respuesta automática] Hola. En este momento no estoy disponible, y no me pondre en contacto contigo mas tarde."
2. Herta tiene una paciencia limitada, por lo que solo nos responderá cinco veces de forma manual. Luego, volverá al mensaje de respuesta automática.
NOTA: El primer mensaje de Herta (inciso 1) **no** cuenta como un mensaje manual.

En los casos de respuesta manual, Herta responderá de la siguiente forma:

1. El mensaje enviado contiene la palabra "kuru" o "kururin":
Herta responderá con uno de sus mensajes icónicos: "KURU", o "KURU KURU", o "KURURIN". La respuesta es al azar.
2. El mensaje enviado contiene la palabra "hola":
Herta responderá: "Hola."
3. El mensaje enviado contiene la palabra "problema":
Herta responderá: "No te preocupes. Ya esta solucionado."
4. El mensaje enviado contiene la palabra "preocupado" o "preocupada":
Herta responderá: "De que te preocupas si yo estoy aqui?"
5. El mensaje enviado contiene la palabra "simulado":
Herta responderá: "La actualizacion del Universo Simulado ya esta lista, ven a probarla."
6. El mensaje enviado es "Eres Herta?":
Herta responderá: "Quieres una selfie para demostrarlo o que?"
7. El mensaje enviado es "Y tu marioneta?":
Herta responderá: "Vaya, parece que la perdi. No me extraña que no la encuentre."
8. Si el mensaje no cumple con ninguna de las condiciones anteriores:
Herta responderá: "Oh." o "..." al azar.

Algunas observaciones:

1. En caso de que un mensaje cumpla varias de las condiciones anteriores, se priorizará **solamente** la primera respuesta en base al *orden* del listado.
2. Deben considerarse las palabras y los mensajes sin importar si los caracteres están en mayúsculas o minúsculas. Esto también aplica a los incisos 6 y 7.

Pruebas:

La clase **Herta** estará probada por una *serie de pruebas* que proporcionará la cátedra en el [repositorio base del Trabajo Práctico](#). La nota de funcionalidad estará dada por la cantidad de pruebas que pase su código. Si una prueba falla, podrán ver un mensaje de error que les explicará con más detalle el problema.

Aclaraciones y criterios de corrección:

Para que el trabajo tenga buena recepción por nuestros superiores en Hoyoverse, se deberán cumplir las siguientes pautas:

1. Debe implementar las clases de manera *totalmente original y propia*. La detección de copias alevosas resultará en la **pérdida de la regularidad de los involucrados**.
2. El trabajo debe compilar con las flags **-Wall -Werror -Wconversion**. Un trabajo que no compila es un trabajo que *no funciona* (y por lo tanto no pasa ninguna prueba).
3. El código **debe pasar** los tests de la cátedra y también respetar sus firmas (es decir, no se puede *modificar* las pruebas).
4. Los nombres de las clases y sus archivos asociados deben estar en camelCase y empezar con mayúscula. Todo lo demás, en snake_case.

Adicionalmente:

1. Se puede utilizar **std::string** y las funciones y métodos asociados.
2. Se pueden utilizar bibliotecas como **<random>**.
3. No es necesaria ninguna estructura de datos para resolver este TP (como, por ejemplo, **std::vector** o **std::list**). Ante la duda, consultar por Slack con algún ayudante.

Los criterios de evaluación y corrección por parte de la cátedra son:

1. **Funcionalidad** (50% de la nota):

Compilación: sin warnings ni errores.

Funcionalidad: que el código pase las pruebas.

2. **Estilo de código** (50% de la nota):

Uso del paradigma.

Eficiencia espacial.

Eficiencia temporal.

Modularización.

Precondiciones y postcondiciones.

Buenas prácticas de programación propuestas por la cátedra.

Formato de entrega:

Se deberá subir al campus un único archivo comprimido **.zip** en la sección TPs.

En la carpeta comprimida, se deberá entregar el repositorio de git. Esto se puede hacer directamente descargando el repositorio de forma manual en la página de GitHub o generando una Release.

El nombre del archivo debe tener el siguiente formato:

tp1-1c2024-USERNAME-PADRON.zip

El plazo de entrega vence el día **VIERNES 29/3 23:59 hrs.** NO se aceptarán entregas fuera de término.

Puntaje: **20** puntos. **10** puntos de funcionalidad y **10** de estilo de código.

