ICC713 - Taller 2: Javascript

En este taller se espera que usted demuestre sus habilidades en el lenguaje Javascript nivel básico. Para esto, se le presenta el siguiente problema para que usted lo solucione con dicho lenguaje:

Existe un juego de RPG por turnos, en que necesitan testear su lógica de batalla y logs, por ende se le solicita realizar la demo de la batalla de este juego.

En este juego existe un stack de ataques posibles por tipo de personaje. Este stack viene adjunto al taller, en el archivo **attacks.json**. Al verlo usted ve que el ataque dispone de los siguientes atributos:

- **name**: Nombre del ataque
- **type**: Tipo del ataque, pueden ser: MAGIC, PHYSICAL
- **damage**: Cantidad de daño
- **accuracy**: Precisión o porcentaje de efectividad del ataque (golpea o falla)

Además, existen tipos de personajes o clases, cada clase puede utilizar solo un tipo de ataque, estas son:

- **Clase:** MAGICIAN **| Tipo de ataque:** MAGIC
- **Clase:** KNIGHT **| Tipo de ataque:** PHYSICAL
- **Clase:** WARRIOR **| Tipo de ataque:** PHYSICAL
- **Clase:** FAIRY **| Tipo de ataque:** MAGIC

Con estos datos, es importante saber que cada personaje lo componen los siguientes atributos:

- **name**: Nombre del personaje (cualquier nombre)
- **class**: Clase o tipo de personaje
- **firstAttack**: Primer ataque, del tipo aceptado para la clase
- **secondAttack**: Segundo ataque, del tipo aceptado para la clase
- **health**: Vida del personaje. (máximo 200, mínimo 100)
- **speed**: Velocidad del personaje. (Máximo: 10, mínimo 1)

En cuanto a la lógica de la batalla, se sabe que:

- Se necesitan 2 personajes para una batalla, pueden ser o no del mismo tipo
- Cada personaje realiza un ataque por turno
- El personaje más rápido (**mayor speed**) siempre pelea primero, si no hay diferencia entonces pelea cualquiera (al azar 50/50)
- El ataque puede fallar, ya que depende del **accuracy**
- Al tener 0 de vida (**health**) el personaje ya no puede continuar

Para la demo se tiene por necesidad de la empresa que usted automatice la batalla con los siguientes datos.

- Generar 2 personajes al azar, con nombre cualquiera pero distinto, con una clase al azar (entre las 4 disponibles) y por ende con 2 ataques de esa clase. Con velocidad (**speed**) al azar entre 1 y 10 y que la vida (**health**) se genere al azar entre 100 y 200.
- Que cada turno el personaje use al azar (50/50) uno de los 2 ataques.
- Que al finalizar la batalla (uno de los 2 no puede continuar) se guarde en un archivo **.txt** el log de toda la batalla. Este log debe tener esta estructura:

—----------

INICIO

Personaje_1 **|** *CLASS_1* | *HEALTH_1* **de vida vs** *Personaje_2* **|** *CLASS_2* | *HEALTH_2* **de vida**

BATALLA

Turno *N*
Personaje_1 **ataca con** *ATACK_NAME***… Da en el blanco!. La vida del** *Personaje_2* **queda en HEALTH_2.**
Personaje_2 **ataca con** *ATACK_NAME***… Da en el blanco!. La vida del** *Personaje_1* **queda en** *HEALTH_1***.**

Turno *N*
Personaje_1 **ataca con** *ATACK_NAME_1***…Falla!. La vida del** *Personaje_2* **se mantiene en** *HEALTH_2***.**
Personaje_2 **ataca con** *ATACK_NAME_2***…Falla!. La vida del** *Personaje_1* **se mantiene en** *HEALTH_1***.**

Turno *N*
Personaje_1 **ataca con** *ATACK_NAME_1***…Falla!. La vida del** *Personaje_2* **se mantiene en** *HEALTH_2***.**
Personaje_2 **ataca con** *ATACK_NAME_2***…Da en el blanco!. El** *Personaje_1* **no puede continuar.**

RESUMEN

Personaje_2 **gana la batalla!**
El *Personaje_1* **falló** *N_FALLOS_1* **veces su ataque**
El *Personaje_2* **falló** *N_FALLOS_2* **veces su ataque**

—----------

Notar que el texto en negrita es fijo, solo el texto en cursiva es el dinámico que debe cambiar según:

- **Personaje_1**: Nombre del personaje 1
- **Personaje_2**: Nombre del personaje 1
- **ATACK_NAME_1**: Nombre del ataque usado por el personaje 1 en el turno
- **ATACK_NAME_2**: Nombre del ataque usado por el personaje 2 en el turno
- **N**: Número del turno
- **HEALTH_1**: Vida actual del personaje 1 en el turno posterior al ataque del personaje 2
- **HEALTH_2**: Vida actual del personaje 2 en el turno posterior al ataque del personaje 1
- **N_FALLOS_1**: Cantidad de ataques fallados en la batalla por parte del personaje 1
- **N_FALLOS_2**: Cantidad de ataques fallados en la batalla por parte del personaje 2

En consola no debe ver nada, solo exportar el log al finalizar la batalla con el archivo: **log_batalla.txt.** Para realizar esta exportación a un archivo se adjunta esta funcion que recibe como parametros un string con todo el log a guardar y el nombre del archivo:

```
function generateFileLog(logs, filename) {
    const fs = require("fs");

    fs.writeFile(filename, logs, (err) => {
        if (err) throw err;
    });
}
```

Suponiendo que tiene todo su string de logs en la variable **fightLogs,** entonces la función se utilizaría al finalizar la batalla así:

```
generateFileLog(fightLogs, "logs_batalla.txt");
```

Con esto se genera un archivo llamado **logs_batalla.txt** en la carpeta del proyecto.

Para cargar el archivo attacks.json en una variable de su programa, debe utilizar la siguiente línea:

```
const attacks = require("./attacks.json");
```

Entonces el array queda en **attacks.**

Evaluación

Al revisar se le **quitarán 5 décimas de nota** por cada funcionalidad, condición o paso saltado, no programado o que no funcione. Todas las entregas comienzan con un 7. A grandes rasgos estas son las funcionalidades padres o claves:

- Creación de personajes:
- Lógica de batalla:
- Generación de logs:

Aproximadamente, la cantidad de funcionalidades detalladas es de 12, por lo tanto, si fallan en todas (ej: no genera aleatoriamente la clase, no genera la vida, no decide quien ataca primero, etc) tendrá un 1.0.

Utilizar al menos una vez el método **filter** de un array. Si no se ve en el código se le bajará 1 punto de nota.

Consejos

Se aconseja dejar la lógica de los logs al finalizar toda la batalla correctamente, ya que si utiliza gran parte de su tiempo en eso puede obtener una nota reprobatoria. Siempre enfocarse en lo importante, en este caso la batalla.

Dividir el trabajo en módulos o funciones que puedan conectarse fácilmente, ya que si lo hacen todo desordenado o sin funciones les va a costar trabajar en equipo. Recuerde: Divide y vencerás y DRY (Don’t repeat yourself).

Aproveche Github.

AVISO IMPORTANTE

Si utiliza **ChatGPT** para la solución, le recomiendo que estudie lo que está entregando, ya que aleatoriamente escogeré a uno del grupo para explicarme su trabajo, esto si encuentro dudas o semejanzas en la posible respuesta que de esta herramienta y su trabajo entregado. Recuerde que como Software Engineer Senior en estos años he realizado múltiples entrevistas a ingenieros de software y en alguna oportunidad me ha tocado decidir quién ingresa a importantes empresas, además, recuerde que mi trabajo full time es trabajar con los lenguajes JS, TS, revisar código de software engineers juniors y semi seniors, se a la perfección cuando alguien sabe y domina lo que creó. Si utiliza la herramienta espero que al menos estudie y así aprenda a explicar.