MANUAL DE USUARIO

Proyecto Final Laboratorio 3

(Valentin Cueva Buono - Juan I. Hussen - Juan C. Martel)

1. Al **iniciar el juego** nos encontraremos con las siguientes opciones:

```
1. Crear Jugador
2. Cargar Partida
```

```
switch (opcion) {
    case 1:
        jugador = crearJugadors();
        zonaActual = Zona.SELVA; // Zona inicial
        enemigosPorZona = inicializarEnemigos(jugador.getNivel());
        break;
    case 2:
        System.out.println("Ingrese el nombre de su personaje");
        String nombrePersonaje = scanner.nextLine();
        jugador = cargarPersonaje(nombrePersonaje);
        enemigosPorZona= inicializarEnemigos(jugador.getNivel());
        zonaActual = Zona.SELVA;
        if (jugador == null) {
            System.out.println("No se pudo cargar el personaje. Creando un nuevo personaje.");
            jugador = crearJugadors();
        }
        break;
    default:
        System.out.println("Opción no válida. Creando un nuevo jugador por defecto.");
        jugador = crearJugadors();
        zonaActual = Zona.SELVA; // Zona inicial
        enemigosPorZona = inicializarEnemigos(jugador.getNivel());
        break;
}
```

El funcionamiento de estas opciones es el siguiente:

1. Crear Jugador

- Permite crear un nuevo personaje para iniciar una partida desde el inicio..
- Seleccionando la opción "1" en el menú principal.
- Se ingresa el nombre del personaje con el que se jugará :

- Así como su raza luego (dentro de las opciones dadas) :

- Se creará un nuevo personaje, el cual comenzará su aventura en la zona inicial (Selva). Se inicializan los enemigos correspondientes al nivel del jugador. En este caso, siempre un nuevo jugador será nivel 1.

2. Cargar Partida

- Permite cargar una partida guardada anteriormente.
- Seleccionando la opción "2" en el menú principal y luego ingresa el nombre del personaje cuando se lo solicite. El nombre del personaje será el nombre con el cual se guardará un archivo con el progreso del mismo.
- Si el personaje existe, se cargará la partida correspondiente con los enemigos correspondientes al nivel del jugador. Si el personaje no se encuentra, se mostrará un mensaje de error y se creará un nuevo personaje por defecto.

Opción No Válida

- Maneja la selección de una opción no válida en el menú principal.
- Si se ingresa una opción que no es "1" ni "2".
- Se mostrará un mensaje indicando que la opción no es válida y se creará un nuevo personaje por defecto, comenzando la aventura en la zona inicial (Selva) con los enemigos correspondientes al nivel del jugador.

2. Inicialización de Enemigos según el Nivel del Jugador

La función ‘inicializarEnemigos’ se encarga de generar enemigos en distintas zonas del juego basándose en el nivel actual del jugador. A continuación se explica cómo funciona:

- Inicializar y asignar enemigos a diferentes zonas del juego de acuerdo con el nivel del jugador.
- Recibe por parámetro el nivel del jugador (int).
- Retorna un mapa (HashMap) que asocia cada zona (Zona) con una lista de enemigos (ArrayList<Personaje>).

Se crea una lista de zonas (zonasACargar) que determinará en qué zonas deberá completar el jugador eliminando enemigos, según su nivel:

- Nivel menor que 3: Se incluyen las zonas Selva, Desierto y Ciudad.
- Nivel entre 3 y 5: Se incluyen las zonas Desierto y Ciudad.
- Nivel 6 o mayor: Solo se incluye la zona Ciudad.

Para cada zona en la lista ‘zonasACargar’, se asignan enemigos específicos.

Esta función se invoca al inicio del juego cuando se crea un nuevo personaje o se carga una partida guardada.

Determina los enemigos que el jugador encontrará en las distintas zonas, adaptando la dificultad y variedad de enemigos según el progreso del jugador.

```java
private HashMap<Zona, ArrayList<Personaje>> inicializarEnemigos(int nivel) {  4 usages
    HashMap<Zona, ArrayList<Personaje>> localEnemigosPorZona = new HashMap<>();

    ArrayList<Zona> zonasACargar = new ArrayList<>();

    if (nivel < 3) {
        zonasACargar.add(Zona.SELVA);
        zonasACargar.add(Zona.DESIERTO);
        zonasACargar.add(Zona.CIUDAD);
    } else if (nivel < 6) {
        zonasACargar.add(Zona.DESIERTO);
        zonasACargar.add(Zona.CIUDAD);
    } else {
        zonasACargar.add(Zona.CIUDAD);
    }

    if (zonasACargar.contains(Zona.SELVA)) {
        localEnemigosPorZona.put(Zona.SELVA, new ArrayList<>(Arrays.asList(
                new Goblin( nombre: "Goblin errante",  vida: 50,  nivel: 1),
                new NoMuerto( nombre: "Doctor Pocovivo",  vida: 30,  nivel: 1),
                new Pandaren( nombre: "Panda bimbo",  vida: 70,  nivel: 2)
        )));
    }
```

```java
    if (zonasACargar.contains(Zona.DESIERTO)) {
        localEnemigosPorZona.put(Zona.DESIERTO, new ArrayList<>(Arrays.asList(
                new ElfoMago( nombre: "El mago cocina del desierto",  vida: 40,  nivel: 3),
                new Enano( nombre: "Pequeño Ladron del Desierto",  vida: 60,  nivel: 4),
                new Goblin( nombre: "Goblin de las Arenas",  vida: 50,  nivel: 4)
        )));
    }

    if (zonasACargar.contains(Zona.CIUDAD)) {
        localEnemigosPorZona.put(Zona.CIUDAD, new ArrayList<>(Arrays.asList(
                new Humano( nombre: "Pandillero",  vida: 45,  nivel: 5),
                new Humano( nombre: "Ladron",  vida: 35,  nivel: 6),
                new Humano( nombre: "Policia Corrupto",  vida: 70,  nivel: 7)
        )));
    }

    return localEnemigosPorZona;
}
```

3. Menu dentro del juego

```java
/**
 * Muestra el menu una vez ya iniciado sesion llamando todas las funciones anteriores*/
public void mostrarMenu() {  1 usage
    //crearArchivos();
    boolean menu = true;
    while (menu) {
        limpiarConsola();
        System.out.println("Menú principal:");
        System.out.println("1. Explorar");
        System.out.println("2. Ver estadisticas");
        System.out.println("3. Inventario");
        System.out.println("4. Ir a la tienda");
        System.out.println("5. Cambiar nombre de jugador");
        System.out.println("6. Guardar Partida");
        System.out.println("7. Salir");
        System.out.print("Elige una opción: ");
        int opcion = scanner.nextInt();
        scanner.nextLine(); // Limpiar el buffer

        switch (opcion) {
            case 1:
                explorar();
                limpiarConsola();
                break;
```

```java
            case 2:
                jugador.mostrarEstadisticas();
                presionarParaContinuar();
                limpiarConsola();
                break;
            case 3:
                jugador.verInventario();
                presionarParaContinuar();
                limpiarConsola();
                break;
            case 4:
                irTienda();
                limpiarConsola();
                break;
            case 5:
                    cambiarNombreAnciano();
                break;
            case 6:
                System.out.println("Guardado");
                guardarPersonaje(jugador);
                break;
            case 7:
                System.out.println("Saliendo del juego....");
                menu=false;
                break;
```

- Ofrece al usuario un menú con varias opciones para explorar el juego, gestionar su personaje y guardar el progreso.
- A partir del número que elija el usuario se selecciona una opción.
- El menú se repetirá hasta que selecciones la opción "Salir" (7), lo que cerrará el juego.

```
Menú principal:
1. Explorar
2. Ver estadísticas
3. Inventario
4. Ir a la tienda
5. Cambiar nombre de jugador
6. Guardar Partida
7. Salir
Elige una opción:
```

3.1)Explorar

```java
 * Simula una accion dentro del menu y crea el escenario de pelea con el enemigo nuevo*/
private void explorar() {  1 usage
    Personaje enemigo = generarEnemigoAleatorio(zonaActual);
    if (enemigo == null) {
        cambiarZona();
    } else {
        System.out.println("¡Has encontrado un " + enemigo.getNombre() + " en la " + zonaAc
        iniciarCombate(enemigo);
    }
}
```

Genera un enemigo aleatorio de los 3 que están cargados dentro del array de cada zona (enum ‘Zona’) . En caso de estar la zona libre de enemigos, significa que ya fue completada y te envía a la siguiente zona de combate.

```java
private void cambiarZona() {  1 usage
    switch (zonaActual) {
        case SELVA:
            zonaActual = Zona.DESIERTO;
            System.out.println("1er CHECKPOINT- SELVA COMPLETADA!");
            System.out.println("Guarda partida si lo crees necesario");
            presionarParaContinuar();

            break;
        case DESIERTO:
            zonaActual = Zona.CIUDAD;
            System.out.println("2do CHECKPOINT DESIERTO COMPLETADO!");
            System.out.println("Guarda partida si lo crees necesario");
            presionarParaContinuar();

            break;
        case CIUDAD:
            System.out.println("¡Completaste todas las zonas!");
            System.out.println("Gracias por jugar!!");
            presionarParaContinuar();
            return;
    }
    System.out.println("¡Has llegado a la " + zonaActual + "!");
}
```

```java
public void iniciarCombate(Personaje enemigo) {  1 usage
    Random random = new Random();
    boolean jugadorVivo = true;
    boolean enemigoVivo = true;

    while (jugadorVivo && enemigoVivo) {
        System.out.println("\nVida del jugador: " + jugador.getVida());
        System.out.println("Vida del enemigo (" + enemigo.getNombre() + "): " + enemigo.
        System.out.println("\nTurno del jugador:");
        System.out.println("1. Atacar");
        System.out.print("Elige una acción: ");
        int eleccion = scanner.nextInt();

        switch (eleccion) {
            case 1:
                limpiarConsola();
                jugador.atacar(enemigo);
                presionarParaContinuar();
                break;
            case 2:
                limpiarConsola();
                jugador.subirNivel();
                break;
            default:
```

En el caso de *'atacar'*, abrirá con jugador.atacar(enemigo objetivo) un menú mostrando los 3 ataques personalizados de cada clase, cada uno con ventajas y desventajas personalizadas según el rival que se encuentre.
El case 2 es únicamente para dinamizar la demostración del funcionamiento del juego en caso de una presentación.

EJEMPLO ENANO.ATACAR:

3.1) Ejemplo de explorar con personaje Enano :

Genera un escenario de combate contra un enemigo aleatorio dentro de las posibilidades de la zona.

```
¡Has encontrado un Goblin errante en la SELVA!

Vida del jugador: 130.0
Vida del enemigo (Goblin errante): 50.0

Turno del jugador:
1. Atacar
Elige una acción:
```

3.1.1) Atacar

Abre un menú para poder “Atacar” al enemigo generado anteriormente.
Este menú de ataques, dependerá de la clase que se eligió cuando se crea el personaje.
Por ejemplo :

```
Tus ataques son:
1. Placaje
2. Ataque con espada
3. Cabezazo
```

Hay ataques predefinidos como “placaje” que viene en todas las clases, y otras como Ataque con espada, que viene con el arma del personaje. En este caso el arma tiene su daño predefinido, el cual afectará el daño del ataque.

Cuando se elija el ataque , este se aplicará sobre el enemigo y reducirá la vida del oponente dependiendo del daño el cual se aplique. Además, hay que tener en cuenta que el daño cambia por tipos de razas, unas tienen ventajas ante otras. Este proceso se repetirá hasta que la vida de alguno de los dos llegue a 0.

```java
public void atacar(Personaje objetivo){  1 usage
    Scanner scanner;
    System.out.println("Tus ataques son:");
    System.out.println("1. Placaje");
    System.out.println("2. Ataque con espada");
    System.out.println("3. Cabezazo");
    double danio=0;
    scanner = new Scanner(System.in);
    int opcion = scanner.nextInt();
    switch (opcion){
        case 1: danio = this.placaje(objetivo);
        break;
        case 2: danio = this.atacarConArma();
        break;
        case 3 : danio = this.cabezazo(objetivo);
        break;
        default:
            System.out.println("no elegiste bien la opcion, perdiste tu turno 🙂 ");
    }
    objetivo.recibirDanio(danio);
}
```

En el caso de resultar vencedores en el combate podemos acceder al inventario del enemigo y retirar una pertenencia según el índice que le pasemos. La recolección de estos ítems es necesaria para completar el juego.

Cuando hayas ganado el combate, la pantalla se vera algo asi :

```
Vida del jugador: 90.0
Vida del enemigo (Goblin errante): -10.0
Goblin errante ha sido derrotado.
juani ha subido al nivel 2!
0) Fragmento de Vinilo de Help!
¿Cuál desea agarrar?
```

El enemigo derrotado dejará un objeto ,el cual podremos agarrar del suelo o no. El mismo servirá para poder intercambiarlo en la tienda, cuando juntemos los necesarios para poder hacerlo.

Seleccionamos el índice del objeto que figura a la izquierda del objeto en caso de querer agarrarlo.

```
    if (enemigo.getVida() <= 0) {
        System.out.println(enemigo.getNombre() + " ha sido derrotado.");
        jugador.subirNivel();
        enemigo.getInventario().agregarElemento(viniloSorpresa);
        jugador.agarrarObjeto(enemigo);
        enemigoVivo = false;
        break;
    }
```

```
Doctor Pocovivo ha sido derrotado.
torrico ha subido al nivel 2!
0) Fragmento de Vinilo de Help!
¿Cuál desea agarrar?
0
Has agarrado: Fragmento de Vinilo de Help!
```

El manejo del inventario funciona a partir de una clase genérica creada llamada ‘ListaGenerica’.

3.2) Mostrar Estadísticas

```
    //stats
    public void mostrarEstadisticas(){  1 usage
        System.out.println(
                "Nombre=" + nombre + "\n" +
                "Vida=" + vida + "\n" +
                "Nivel=" + nivel + "\n" +
                "Arma=" + arma.getNombre()+ "\n" +
                "Daño Arma=" + arma.getDano() + "\n");
    }
}
```

Llama al método dentro de la clase personaje que muestra sus atributos + el arma que equipa con su respectivo daño.

3.3) Ver Inventario

(En la clase del personaje)

```
public void verInventario() { inventario.listarElementos(); }
```

(Método de la lista genérica)

```
@Override  2 usages
public void listarElementos() {
    int i =0;
    for(T objetos:listaGenerica)
    {
        System.out.println(i+") "+objetos.toString());
        i++;
    }

}
```

La función ‘listarElementos’ se utiliza para mostrar en pantalla todos los elementos de una lista genérica de objetos.

3.4) Ir Tienda

La función ‘irTienda’ te permite interactuar con un personaje en el juego que te asignará una misión y te recompensará por completarla. Dependiendo de la cantidad de fragmentos de vinilo que tenga el jugador en su inventario, recibirá diferentes recompensas.

```
else if (jugador.inventario.contarElementos()>= 6) {
  Arma poderosaElectrica = new Arma( nombre: "Bajo Iconico Hofner de 'Macca'",  peso: 2,  dano: 100,
  System.out.println("Toma este iconico Bajo Hofner electrico que me regalo un amigo, te servi
  jugador.setArma(poderosaElectrica);
  System.out.println("Has recibido "+ poderosaElectrica.getNombre());
  presionarParaContinuar();
```

3.5) Cambiar tu Nombre hablando con un Anciano

La función ‘cambiarNombreAnciano’ permite al jugador modificar el nombre de su personaje mediante un proceso interactivo que incluye un diálogo con un anciano del juego. En el caso de realizarse un cambio de nombre, recibirá una penalización en su vida al sellar su “pacto de sangre”.

```
try{
    System.out.println("Cual va a ser su nuevo nombre ? : ");
    String nombreNuevo = scanner.nextLine();
    jugador.setNombre(nombreNuevo);
}
catch (Exception e){throw new RuntimeException("inserta un nombre valido por favor");}
jugador.setVida(jugador.getVida() - 15);
System.out.println(jugador.getNombre() + "has perdido 15 de vida por sellar el pacto, felicit
presionarParaContinuar();
```

3.6) Guardar Partida

La función ‘guardarPersonaje’ permite guardar el estado actual del personaje en un archivo, de manera que se pueda recuperar más tarde. Esto es esencial para conservar el progreso del jugador entre sus sesiones de juego.

```
public static void guardarPersonaje(Personaje personaje) {  1 usage
    try (FileOutputStream fileOut = new FileOutputStream(personaje.getNombre());
         ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
        out.writeObject(personaje);
        System.out.println("Guardado Finalizado " + personaje.getNombre());
    } catch (IOException e) {
        System.err.println("Error al guardar el personaje en el archivo " + personaje.getNombre
        e.printStackTrace();
    }
}
```

3.7) Salir del juego

Finaliza la ejecución del bucle del menú.

```
case 7:
    System.out.println("Saliendo del juego....");
    menu=false;
    break;
```

Implementación de API

La clase ‘elfoMago’ se conecta con la api, la instancia y guarda todos sus datos en la clase ‘ClimaApi’. Una vez inicializada la api del clima pide los datos de la húmedad, y la temperatura de un día en específico. Decidimos que trabaje con los datos de la ciudad de Mar del Plata. Una vez obtenidos los datos, compara las variables recibidas para afectar el rendimiento de los ataques de la clase ‘elfoMago’.

```
public static void actualizarDatosClima(String fecha) {  2 usages
    try {
        JSONObject json_datos = new JSONObject(ClimaAPI.getInfo(fecha));
        datosClima.fromJSON(json_datos);
    } catch (JSONException e) {
        /// llamada recursiva a funcion
        actualizarDatosClima(fecha);
    }
}
```

(en clase ClimaApi)

EJEMPLOS DE CÓMO AFECTA EL CLIMA A LAS HABILIDADES DEL ELFO MAGO:

```
public double bolaDeFuego(){  1 usage
    double danio =0;
    if (datos.getHumedad()>80)
    {
        danio=20;
        System.out.println("Por la alta la humedad tu bola de fuego hace menos daño, hizo"+danio);
    }else {
        danio=35;
        System.out.println("Gracias a la baja humedad tu bola de fuego hace mas daño, hizo"+danio);
    }
    return danio;
}
```

```
public double congelar(){  1 usage
    double danio=0;
    if(datos.getTemperaturaMaxima()<20)
    {
        danio=35;
        System.out.println("Gracias a la baja temperatura tu hielo hoy hace mas daño, hizo"+danio);

    }else{
        danio=15;
        System.out.println("Por la alta temperatura tu hielo se fue derritiendo e hizo menos daño "+danio);
        }
    return danio;
}
```

```
public double hidroBomba(){  1 usage
    double danio=0;
    if(datos.getHumedad()>80)
    {
        danio=35;
        System.out.println("Aprovechas el dia humedo para hacer mas daño, hiciste "+danio);

    }else{danio=15;
        System.out.println("Por culpa de la baja humedad y poco aprovechamiento del agua, se hizo menos daño :"+dani
        }
    return danio;
}
```