Jugadores

- String nombre; //Nombre del jugador.
- int salud; //Salud del jugador.
- int puntos; //Puntos acumulados.

```
//Constructor que recibe nombre, salud y empezar por 0 puntos de un jugador.
+ Jugadores(String nombre, int salud) {
this.nombre = nombre;
this.salud = salud;
this.puntos = 0;
}
+ String getNombre() { //Devolver el nombre del jugador.
return nombre;
}
+ int getSalud() { //Devolver salud del jugador.
return salud;
}
+ void recibirdaño(int daño) { //Restar salud, es decir, aplicando daño al jugador.
salud -= daño;
if (salud < 0) {
salud = 0; //La salud no puede ser negativa.
}
}
//Sumar puntos totales al jugador.
+ void sumarPuntos(int puntosGanados) {
puntos += puntosGanados;
}
//Devolver la cantidad de puntos acumulados por el jugador.
+ int getPuntos() {
return puntos;
}
//Devolver True si el jugador todavia tiene salud.
+ boolean estaVivo() {
return salud > 0;
}
}
```

GestorEquipos

- List<Jugadores> listaJugadores; //Una lista para almacenar todos los jugadores ingresados.

```
//Constructor para iniciar la lista vacía.
+ GestorEquipos() {
listaJugadores = new ArrayList<>();
}
//Metodo para agregar jugadores a la lista.
+ void agregarJugador(Jugadores jugador) {
listaJugadores.add(jugador);
}
//Devolver la lista actual de jugadores.
+ List<Jugadores> getListaJugadores() {
return listaJugadores;
}
+ boolean nombreJugadorRepetido(String nombre) { //Metodo booleano para devolver verdadero o falso
for (Jugadores jugadores : listaJugadores) {
if (jugadores.getNombre().equalsIgnoreCase(nombre)) { //Condicion si hay nombres iguales
return true; //Devolver un valor verdadero
}
}
return false; //Devolver un valor falso
}
//Mostrar todos los jugadores de la lista con sus salud y puntos.
+ void mostrarJugadores() {
if (listaJugadores.isEmpty()) {
//En caso de que no haya jugadores en la lista.
"Error: no hay jugadores en la lista."
return;
}
//Mostrar los datos de cada jugador.
String mensaje = "Lista de jugadores:\n";
for (Jugadores jugador : listaJugadores) {
mensaje += jugador.getNombre() + "\nSalud: " + jugador.getSalud() + "\nPuntos: " + jugador.getPuntos() + "\n";
}
//Mostrar la información de todos los jugadores.
mensaje
}
}
```

Main

```
String[] Opciones = {
"Jugar partida","Salir"
};
int opcion =0; //Variable para guardar la opción seleccionada del menu principal.
do { //Mostrar un menu con una imagen y opciones.
opcion=¿Usted quiere jugar una partida de enfrentamiento?
switch (opcion) {
case 0://Jugar partida
int opcion1 =0; //Variable para guardar la opción.
//Opciones de la partida de enfrentamiento.
String[] Opciones1 = {
"Zombies","Supervivientes","Ver equipos","Empezar la campaña","Salir"
};
//Llamar al gestor de equipos para controlar la lista de jugadores.
GestorEquipos gestorJugador = new GestorEquipos();
do { //Mostrar las instrucciones.
opcion1= "Primero deberá ingresar el nombre de usuario de los 4 jugadores supervivientes y de los 4
jugadores \njugadores zombies, para después empezar la campaña/partida."
switch (opcion1) {
case 0: //Zombies
//Solicitar el nombre del jugador zombie.
String nombreZombie = "Ingrese el nombre del Jugador Zombie:"
if (gestorJugador.nombreJugadorRepetido(nombreZombie)) {
"Error: ya existe un jugador en la lista de supervivientes con un nombre igual al ingresado, ingrese otro."
break;
}
Jugadores jugadorZombie = new Jugadores(nombreZombie, 100); gestorJugador.agregarJugador(jugadorZombie);
break;
case 1://Supervivientes
//Solicitar el nombre del jugador superviviente.
String nombreSuperviviente = "Ingrese el nombre del Jugador Superviviente:"
if (gestorJugador.nombreJugadorRepetido(nombreSuperviviente)) {
"Error: ya existe un jugador en la lista de zombies con un nombre igual al ingresado, ingrese otro."
break;
}
Jugadores jugadorSuperviviente = new Jugadores(nombreSuperviviente, 100);
gestorJugador.agregarJugador(jugadorSuperviviente);
break;
case 2://Ver equipos
gestorJugador.mostrarJugadores(); //Mostrar a los jugadores ingresados.
break;
case 3://Empezar la campaña
//Verificar si hay 8 jugadores para iniciar el enfrentamiento.
if (gestorJugador.getListaJugadores().size() > 8) {
"Error: hay más de 8 jugadores cargados."
break;
}else if (gestorJugador.getListaJugadores().size() < 8) {
"Error: hay menos de 8 jugadores cargados."
break;
}else {
//Crear la campaña con los jugadores que esten en la lista.
Equipo campaña = new Equipo(gestorJugador);
//Ingresar el número de capitulos de la campaña.
String Capitulos = "Ingrese el número de capítulos de la campaña (máximo 5)"
int numeroCapitulos = Integer.parseInt(Capitulos);
//¡¡¡QUE EMPIECE EL ENFRENTAMIENTO!!!
campaña.empezar(numeroCapitulos);
}
break;
case 4://Salir de la partida
"Saliendo de la partida."
break;
}
}while (opcion1!=4); //Hacer un bucle hasta que se elija salir del menu.
case 1://Salir del menu principal
"Saliendo del menu principal."
break;
}
} while (opcion!=1); //Hacer un bucle hasta que se elija salir del menu principal.
}
}
```

Equipo

- GestorEquipos gestorJugador; //Variable para acceder a la lista de jugadores cargados previamente.

```
+ Equipo(GestorEquipos gestorJugador) { //Constructor que recibe el gestor de jugadores.
this.gestorJugador = gestorJugador;
}
+ void empezar(int numeroCapitulos) { //Metodo para iniciar la partida con la cantidad de capitulos
ingresada.
if (numeroCapitulos <1 || numeroCapitulos>5) {//Verificar que el numero de capitulos esté entre 1 y 5.
"Error: el número de capítulos debe ser entre 1 y 5."
return;
}
if (gestorJugador.getListaJugadores().size() >= 8 && gestorJugador.getListaJugadores().size() <= 8) {//Verificar
que haya 8 jugadores.
//Obtener los 8 jugadores.
Jugadores jugador1 = gestorJugador.getListaJugadores().get(0);
Jugadores jugador2 = gestorJugador.getListaJugadores().get(1);
Jugadores jugador3 = gestorJugador.getListaJugadores().get(2);
Jugadores jugador4 = gestorJugador.getListaJugadores().get(3);
Jugadores jugador5 = gestorJugador.getListaJugadores().get(4);
Jugadores jugador6 = gestorJugador.getListaJugadores().get(5);
Jugadores jugador7 = gestorJugador.getListaJugadores().get(6);
Jugadores jugador8 = gestorJugador.getListaJugadores().get(7);
for (int i = 1; i <= numeroCapitulos; i++) { //Por cada capitulo, se crean los enfrentamientos y se juegan
las rondas.
List<Jugadores[]> enfrentamiento = new ArrayList<>(); //Lista de pares de jugadores para este capitulo.
//Agregar 4 enfrentamientos por capítulo.
enfrentamiento.add(new Jugadores[]{jugador1, jugador3}); //SUPERVIVIENTE1 vs ZOMBIE1
enfrentamiento.add(new Jugadores[]{jugador2, jugador4}); //SUPERVIVIENTE2 vs ZOMBIE2
enfrentamiento.add(new Jugadores[]{jugador6, jugador5}); //SUPERVIVIENTE3 vs ZOMBIE3
enfrentamiento.add(new Jugadores[]{jugador8, jugador7}); //SUPERVIVIENTE4 vs ZOMBIE4
Capitulos capitulo = new Capitulos(i, enfrentamiento); //Crear un nuevo capítulo con los enfrentamientos.
//Ejecutar los enfrentamientos normales (supervivientes y zombies) e inversas (zombies [supervivientes] y
supervivientes [zombies]) del capítulo.
capitulo.jugarRonda(); //Primera ronda.
capitulo.jugarRondaInversa(); //Intercambiar los roles.
capitulo.mostrarResultados(); //Mostrar los resultados.
}
} else {
//En caso de que no haya jugadores en las listas.
"Error: no hay suficientes jugadores en la lista."
}
//Mensaje de enfrentamiento finalizado.
"Fin de la campaña."
gestorJugador.mostrarJugadores(); //Mostrar los puntos de los jugadores.
//Variables para acumular los puntos para ambos equipos.
int puntosZombie = 0;
int puntosSuperviviente = 0;
//Mostrar los puntos por cada jugador.
for (Jugadores jugador : gestorJugador.getListaJugadores()) {
jugador.getNombre() + " - Puntos: " + jugador.getPuntos()
}
//Sumar puntos según el rol que ocuparon en los enfrentamientos.
List<Jugadores> jugadores = gestorJugador.getListaJugadores();
for (int i = 0; i < jugadores.size(); i++) {
Jugadores jugador = jugadores.get(i);
if (i < jugadores.size() / 2) {
puntosZombie += jugador.getPuntos();
} else {
puntosSuperviviente += jugador.getPuntos();
}
}
//Mostrar los resultados.
if (puntosZombie > puntosSuperviviente) {          "¡Ganaron los Zombies!"
} else if (puntosSuperviviente > puntosZombie) {
"¡Ganaron los Supervivientes!"
} else {
"¡Empate!"
}
//Mostrar los puntos de cada equipo.          "Puntos totales: " + "\nEquipo Zombie: " + puntosZombie +
"\nEquipo Superviviente: " + puntosSuperviviente
}
}
```

Capitulos

- int numeroCapitulo; //Número del capitulo actual.
- List<Jugadores[]>enfrentamiento; //Lista de enfrentamientos con pares de jugadores: supervivientes y zombies.
- Random random; //Generador de números aleatorios.

```
//Constructor para recibir el número del capitulo y la lista de enfrentamientos.
+ Capitulos(int numeroCapitulo, List<Jugadores[]>enfrentamiento) {
this.numeroCapitulo = numeroCapitulo;
this.enfrentamiento=enfrentamiento;
this.random = new Random();
}
//Metodo para ejecutar la ronda normal donde el zombie ataca al superviviente.
+ void jugarRonda() {
for (Jugadores[] jugador : enfrentamiento) {
Jugadores jugadorSuperviviente = jugador[0]; //Jugador superviviente.
Jugadores jugadorZombie = jugador[1]; //Jugadore zombie.
int daño = (int)(Math.random()*100); //Generar aleatoriamente un daño entre 0 y 99.
jugadorSuperviviente.recibirdaño(daño); //Daño al superviviente.
jugadorZombie.sumarPuntos(daño); //Sumar puntos al jugador zombie del daño hecho
al superviviente.
//Mostrar los resultados del primer enfrentamiento entre jugadores zombies y jugadores
supervivientes.
"Capítulo " + numeroCapitulo + ": Enfrentamiento entre " +
jugadorSuperviviente.getNombre() + " (Superviviente) y " + jugadorZombie.getNombre() + "
(Zombie)"
jugadorSuperviviente.getNombre() + " ha recibido " + daño + " de daño."
jugadorZombie.getNombre() + " ha ganado " + daño + " puntos por el daño."
}
}
//Metodo para ejecutar el segundo enfrentamiento (invertir los roles de los jugadores)
donde el superviviente ataca al zombie (sin modificar la lista de jugadores).
+ void jugarRondaInversa() {
for (Jugadores[] jugador : enfrentamiento) {
//Invertir los roles de los jugadores.
Jugadores invertirrol = jugador[0];
jugador[0] = jugador[1]; //Jugadore zombie.
jugador[1] = invertirrol; //Jugador superviviente.
int daño = (int)(Math.random()*100); //Generar aleatoriamente un daño entre 0 y 99.
jugador[0].recibirdaño(daño); //El jugador superviviente recibe daño.
jugador[1].sumarPuntos(daño); //El jugador zombie se le suma puntos.
//Mostrar el resultado del segundo enfrentamiento (invertir los roles de los jugadores).
"Ronda Inversa - " + jugador[0].getNombre() + " ha recibido " + daño + " de daño.\n" +
jugador[1].getNombre() + " gana " + daño + " puntos."
}
}
//Mostrar los resultados del enfrentamiento en general del capitulo.
+ void mostrarResultados() {
String resultado = "Los resultados del capítulo " + numeroCapitulo + " son:\n";
for (Jugadores[] jugador : enfrentamiento) {
resultado += "Superviviente: " + jugador[0] + "\nZombie: " + jugador[1] + "\n\n";
}
resultado
}
}
```