

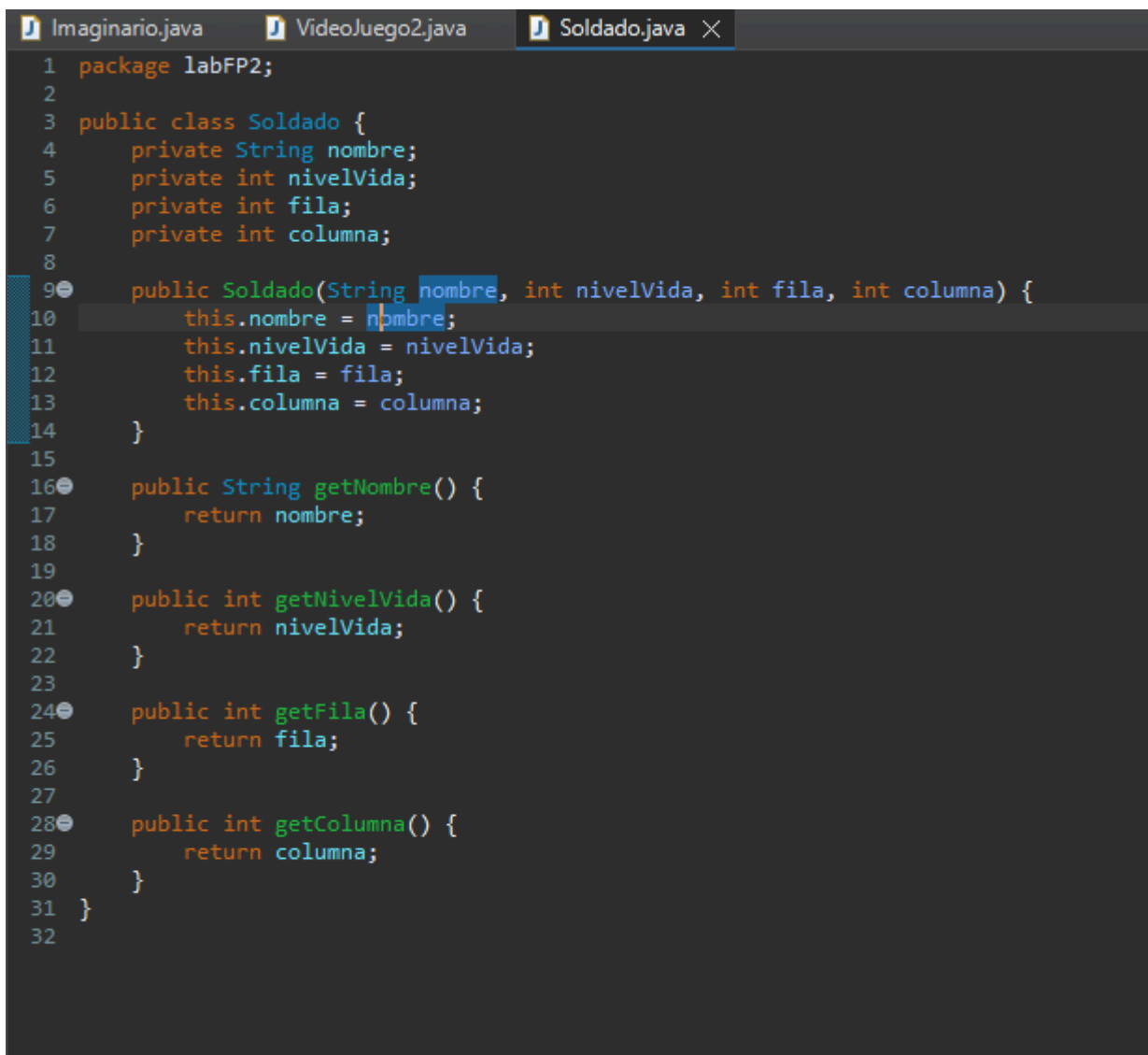
## INFORME DE LABORATORIO

INFORMACIÓN BÁSICA					
ASIGNATURA:	PROGRAMACIÓN WEB 1				
TÍTULO DE LA PRÁCTICA:	Laboratorio 05 : Arreglos Bidimensionales de Objetos				
NÚMERO DE PRÁCTICA:	05	AÑO LECTIVO:	2024 - B	NRO. SEMESTRE:	II
FECHA DE PRESENTACIÓN	22/09/2024	HORA DE PRESENTACIÓN	11:59:00 PM		
INTEGRANTE (s):  Santiago Enrique Palma Apaza				NOTA:	
DOCENTE: Ing. Lino Jose Pinto Oppé					

SOLUCIÓN Y RESULTADOS
<p><b>I. SOLUCIÓN DE EJERCICIOS/PROBLEMAS</b></p> <p>1. Cree un Proyecto llamado Laboratorio5</p> <p>2. Usted deberá crear las dos clases Soldado.java y VideoJuego2.java. Puede reutilizar lo desarrollado en Laboratorio 3 y 4.</p> <p>3. Del Soldado nos importa el nombre, nivel de vida, fila y columna (posición en el tablero).</p> <p>4. El juego se desarrollará en el mismo tablero de los laboratorios anteriores. Pero ahora el tablero debe ser un arreglo bidimensional de objetos. 5. Inicializar el tablero con n soldados aleatorios entre 1 y 10. Cada soldado tendrá un nombre autogenerado: Soldado0, Soldado1, etc., un valor de nivel de vida autogenerado aleatoriamente [1..5], la fila y columna también autogenerados aleatoriamente (verificar que no puede haber 2 soldados en el mismo cuadrado). Se debe mostrar el tablero con todos los soldados creados (usar caracteres como   _ y otros). Además, mostrar los datos del Soldado con mayor nivel de vida, el promedio de nivel de vida de todos los soldados creados, el nivel de vida de todo el ejército, los datos de todos los soldados en el orden que fueron creados y un ranking de poder de todos los soldados creados, del que tiene más nivel de vida al que tiene menos (usar al menos 2 algoritmos de ordenamiento).</p> <p><b>CLASE SOLDADO</b></p> <p><i>clase Soldado representa a un soldado en un juego. Contiene:</i></p>

- **Atributos:**
  - *nombre:* el nombre del soldado.
  - *nivelVida:* la cantidad de vida del soldado.
  - *fila y columna:* la posición del soldado en un tablero.
- **Constructor:** inicializa un nuevo soldado con un nombre, nivel de vida y su posición.
- **Métodos:**
  - *getNombre():* devuelve el nombre del soldado.
  - *getNivelVida():* devuelve el nivel de vida del soldado.
  - *getFila():* devuelve la fila de su posición.
  - *getColumna():* devuelve la columna de su posición.

*Esta clase te permite crear y gestionar soldados en el contexto de tu aplicación o juego.*



```

1 package labFP2;
2
3 public class Soldado {
4     private String nombre;
5     private int nivelVida;
6     private int fila;
7     private int columna;
8
9     public Soldado(String nombre, int nivelVida, int fila, int columna) {
10         this.nombre = nombre;
11         this.nivelVida = nivelVida;
12         this.fila = fila;
13         this.columna = columna;
14     }
15
16     public String getNombre() {
17         return nombre;
18     }
19
20     public int getNivelVida() {
21         return nivelVida;
22     }
23
24     public int getFila() {
25         return fila;
26     }
27
28     public int getColumna() {
29         return columna;
30     }
31 }
32

```

**Videojuego2. java**

### 1. Clase Principal (VideoJuego2):

- *Esta clase contiene el método main, que es el punto de entrada del programa.*

## **2. Variables y Inicialización:**

- *Se crea un tablero de 10x10 (Soldado[][] tablero).*
- *Se utiliza un ArrayList para almacenar soldados (ArrayList<Soldado> soldados).*
- *Se genera un número aleatorio de soldados entre 0 y 10 (numeroSoldados).*
- *Se inicializa vidaTotal para almacenar la suma de la vida de todos los soldados.*

### **Creación de Soldados**

## **3. Bucle para Crear Soldados:**

- *Un bucle for que itera hasta el número de soldados a crear.*
- *Dentro del bucle, se generan aleatoriamente la vida, fila y columna de cada soldado.*
- *Se utiliza un do...while para asegurarse de que un soldado no se coloque en una posición ocupada en el tablero (verificar(tablero, soldado)).*
- *Se añaden los soldados al tablero y a la lista de soldados, y se suma su vida a vidaTotal.*

### **Métodos para Mostrar Información**

## **4. Mostrar el Tablero:**

- *mostrar(Soldado[][] tablero): imprime el estado del tablero, mostrando los nombres de los soldados o un marcador de vacío si no hay soldado en esa posición.*

## **5. Soldado con Mayor Vida:**

- *soldadoMayorVida(ArrayList<Soldado> soldados): encuentra y muestra el soldado con la mayor vida.*

## **6. Mostrar Datos del Ejército:**

- *mostrarDatosEjercito(ArrayList<Soldado> soldados, int vidaTotal): calcula y muestra la vida total y el promedio de vida de los soldados, además de listar todos los soldados y sus vidas.*

### **Métodos de Clasificación**

## **7. Clasificación de Soldados:**

- *Burbuja (rankingSoldadosBurbuja(ArrayList<Soldado> soldados)): implementa el algoritmo de ordenamiento burbuja para clasificar soldados por vida y mostrar el ranking.*
- *Selección (rankingSoldadosSeleccion(ArrayList<Soldado> soldados)): utiliza el algoritmo de selección para ordenar soldados por vida y mostrar el ranking.*

### **Otros Métodos**

## **8. Buscar Soldado:**

- *buscarSoldado(ArrayList<Soldado> soldados, String nombre): busca un soldado por nombre en la lista y muestra su información si es encontrado.*

```

1 package labFP2;
2
3 import java.util.ArrayList;
4
5 public class VideoJuego2 {
6     public static void main(String[] args) {
7         Soldado[][] tablero = new Soldado[10][10];
8         ArrayList<Soldado> soldados = new ArrayList<>();
9         int numeroSoldados = (int) (Math.random() * 11);
10        int vidaTotal = 0;
11
12        for (int i = 0; i < numeroSoldados; i++) {
13            Soldado soldado;
14            int vida, fila, columna;
15            do {
16                vida = (int) (Math.random() * 6) + 1;
17                fila = (int) (Math.random() * 10);
18                columna = (int) (Math.random() * 10);
19                soldado = new Soldado("Soldado" + i, vida, fila, columna);
20            } while (verificar(tablero, soldado));
21            tablero[fila][columna] = soldado;
22            soldados.add(soldado);
23            vidaTotal += vida;
24        }
25
26        mostrar(tablero);
27        soldadoMayorVida(soldados);
28        mostrarDatosEjercito(soldados, vidaTotal);
29        rankingSoldadosBurbuja(soldados);
30        rankingSoldadosSeleccion(soldados);
31    }
32
33    public static boolean verificar(Soldado[][] tablero, Soldado soldado) {
34        return tablero[soldado.getFila()][soldado.getColumna()] != null;
35    }
36
37    public static void mostrar(Soldado[][] tablero) {
38        for (int i = 0; i < tablero.length; i++) {
39            for (int j = 0; j < tablero[i].length; j++) {
40                if (tablero[i][j] == null) {
41                    System.out.print("_____| ");
42                } else {
43                    System.out.print(tablero[i][j].getNombre() + " | ");
44                }
45            }
46            System.out.println();
47        }
48    }
49
50    public static void soldadoMayorVida(ArrayList<Soldado> soldados) {
51        Soldado soldadoMayor = soldados.get(0);
52        for (Soldado soldado : soldados) {
53            if (soldado.getNivelVida() > soldadoMayor.getNivelVida()) {
54                soldadoMayor = soldado;
55            }
56        }
57        System.out.println("Soldado con mayor nivel de vida: " + soldadoMayor.getNombre() +
58            " (Vida: " + soldadoMayor.getNivelVida() + ")");
59    }

```

```

61 public static void mostrarDatosEjercito(ArrayList<Soldado> soldados, int vidaTotal) {
62     double promedioVida = (double) vidaTotal / soldados.size();
63     System.out.println("Nivel de vida total del ejército: " + vidaTotal);
64     System.out.println("Promedio de nivel de vida: " + promedioVida);
65
66     System.out.println("Datos de todos los soldados:");
67     for (Soldado soldado : soldados) {
68         System.out.println(soldado.getNombre() + " - Vida: " + soldado.getNivelVida());
69     }
70 }
71
72 public static void rankingSoldadosBurbuja(ArrayList<Soldado> soldados) {
73     Soldado[] soldadosArray = soldados.toArray(new Soldado[0]);
74     for (int i = 0; i < soldadosArray.length - 1; i++) {
75         for (int j = 0; j < soldadosArray.length - i - 1; j++) {
76             if (soldadosArray[j].getNivelVida() < soldadosArray[j + 1].getNivelVida()) {
77                 Soldado temp = soldadosArray[j];
78                 soldadosArray[j] = soldadosArray[j + 1];
79                 soldadosArray[j + 1] = temp;
80             }
81         }
82     }
83
84     System.out.println("Ranking de poder de los soldados (Burbuja):");
85     for (Soldado soldado : soldadosArray) {
86         System.out.println(soldado.getNombre() + " - Vida: " + soldado.getNivelVida());
87     }
88 }
89
90 public static void rankingSoldadosSeleccion(ArrayList<Soldado> soldados) {
91     Soldado[] soldadosArray = soldados.toArray(new Soldado[0]);
92
93     for (int i = 0; i < soldadosArray.length - 1; i++) {
94         int maxIdx = i;
95         for (int j = i + 1; j < soldadosArray.length; j++) {
96             if (soldadosArray[j].getNivelVida() > soldadosArray[maxIdx].getNivelVida()) {
97                 maxIdx = j;
98             }
99         }
100         Soldado temp = soldadosArray[maxIdx];
101         soldadosArray[maxIdx] = soldadosArray[i];
102         soldadosArray[i] = temp;
103     }
104
105     System.out.println("Ranking de poder de los soldados (Selección):");
106     for (Soldado soldado : soldadosArray) {
107         System.out.println(soldado.getNombre() + " - Vida: " + soldado.getNivelVida());
108     }
109 }
110
111 public static void buscarSoldado(ArrayList<Soldado> soldados, String nombre) {
112     for (Soldado soldado : soldados) {
113         if (soldado.getNombre().equals(nombre)) {
114             System.out.println("Encontrado: " + soldado.getNombre() + " - Vida: " + soldado.getNivelVida());
115             return;
116         }
117     }
118     System.out.println("Soldado no encontrado: " + nombre);
119 }

```

## **II. PRUEBAS**

*¿Con qué valores comprobaste que tu práctica estuvo correcta?*

*¿Qué resultado esperas obtener para cada valor de entrada?*

*¿Qué valor o comportamiento obtuviste para cada valor de entrada?*

### **1. Valores de Entrada y Comprobaciones**

**Entrada:** La cantidad de soldados (número aleatorio entre 0 y 10).

**Comprobaciones:**

- *Verificar que se crean entre 0 y 10 soldados.*
- *Asegurarse de que cada soldado tiene una vida entre 1 y 6.*
- *Comprobar que no hay soldados en posiciones duplicadas en el tablero.*
- *Validar que los métodos de clasificación devuelven la lista ordenada correctamente.*

### **2. Resultados Esperados**

- **Número de Soldados:** *Debería ser un número entero entre 0 y 10.*
- **Vida de Soldados:** *Cada soldado debería tener un valor de vida que varíe entre 1 y 6.*
- **Posiciones en el Tablero:** *No debería haber más de un soldado en la misma posición (validar con el método verificar).*
- **Mayor Vida:** *El método que busca el soldado con mayor vida debería devolver el soldado correcto.*
- **Promedio de Vida:** *El promedio de vida debería ser calculado como la suma de las vidas de todos los soldados dividida por el número de soldados.*
- **Ranking:** *Tanto el ranking por burbuja como el de selección deberían mostrar la lista de soldados ordenada de mayor a menor vida.*

### **3. Resultados Obtenidos**

- **Número de Soldados:** *Si ejecutas varias veces, deberías observar diferentes números entre 0 y 10.*
- **Vida de Soldados:** *Cada soldado debería mostrar una vida entre 1 y 6, sin excepciones.*
- **Posiciones en el Tablero:** *Al imprimir el tablero, deberías ver soldados colocados en diferentes posiciones, sin duplicados.*
- **Mayor Vida:** *El soldado con mayor vida debería ser correcto y coincidir con el que realmente tiene más vida.*
- **Promedio de Vida:** *El promedio calculado debe ser coherente con la suma total de vida de los soldados.*
- **Ranking:** *Los soldados deben estar listados correctamente en orden descendente según su vida.*

## COMMITTS

### VIDEOJUEGO2.JAVA

#### Resumen de Commits

##### 1. Commit 1:

- Descripción: Este commit establece la estructura básica del juego y la creación de soldados aleatorios. Aquí se inicializa el tablero y se generan soldados con atributos aleatorios, como vida y posición en el tablero.

```
santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git add VideoJuego2.java

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git commit -m "Inicialización del tablero y creación de soldados aleatorios."
[master 1fac64e] Inicialización del tablero y creación de soldados aleatorios.
1 file changed, 30 insertions(+)
 create mode 100644 VideoJuego2.java

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git push origin master
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 716 bytes | 716.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_0
5
```

##### 2. Commit 2:

- Descripción: En este commit se añade la funcionalidad para mostrar el estado del tablero y los datos del ejército. Se implementa un método que visualiza el tablero y otro que muestra información sobre cada soldado en el ejército.

```

$ git add VideoJuego2.java

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git commit -m "Añadido método para mostrar el tablero y datos del ejército."
[master 3f186e2] Añadido método para mostrar el tablero y datos del ejército.
1 file changed, 27 insertions(+)

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 990 bytes | 990.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_05
1fac64e..3f186e2 master -> master

```

### 3. Commit 3:

- Descripción: Este commit implementa la lógica para determinar qué soldado tiene la mayor vida y también clasifica a los soldados según su nivel de vida. Se añaden métodos que calculan y muestran estas estadísticas.

```


santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git add VideoJuego2.java

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git commit -m "Implementados métodos de ranking de soldados y mayor vida."
[master eafb98b] Implementados métodos de ranking de soldados y mayor vida.
1 file changed, 48 insertions(+), 6 deletions(-)

santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 12 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 856 bytes | 856.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/santiagopalma12/PALMA_APAZA_SANTIAGO_ENRIQUE_LABORATORIO_05
3f186e2..eafb98b master -> master


santi@5lofiu MINGW64 /c/users/santi/onedrive/escritorio/sas (master)
$ !

```


**PALMA\_APAZA\_SANTIAGO\_ENRIQUE\_LABORATORIO\_05**
Public
Pin
Unwatch

master
1 Branch
Tags

Add file
Code


**santiagopalma12**
Implementados métodos de ranking de soldados y mayor vida.
eafb98b · 6 minutes ago
7 Commits

Soldado.java



## 1. Commit del 17 de octubre de 2024: "Se cambió la clase Soldado, añadiendo los atributos de fila y columna"

- **Descripción:** Este commit indica que se modificó la clase Soldado para incluir dos nuevos atributos: fila y columna.
- **Significado:** Estos atributos son importantes porque permiten ubicar a cada soldado en el tablero de juego, lo que es fundamental para la lógica del programa. Esto sugiere que el juego necesita conocer la posición de cada soldado para poder representarlos correctamente en el tablero.

```
santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio
$ cd sas

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas (master)
$ git init
Reinitialized existing Git repository in C:/Users/santi/OneDrive/Escritorio/sas/.git/

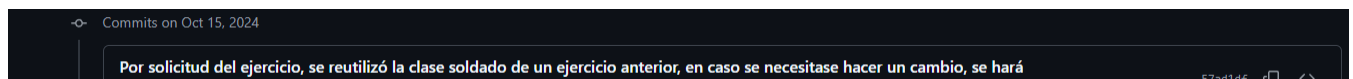
santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas (master)
$ git add Soldado.java

santi@5lofiu MINGW64 /c/users/santi/OneDrive/Escritorio/sas (master)
$ git commit -m "Se cambió la clase Soldado, añadiendo los atributos de fila y columna"
On branch master
```



## 2. Commit del 15 de octubre de 2024: "Por solicitud del ejercicio, se reutilizó la clase soldado de un ejercicio anterior, en caso se necesitase hacer un cambio, se hará"

- **Descripción:** Este commit menciona que se ha reutilizado la clase Soldado de un ejercicio previo, lo cual indica que se está aprovechando código existente en lugar de crear una nueva implementación desde cero.
- **Significado:** Reutilizar código puede ser una buena práctica para evitar duplicación y para mantener la coherencia en la lógica del programa. Sin embargo, también sugiere que si es necesario realizar cambios para adaptarla a las necesidades actuales del ejercicio, se está dispuesto a hacerlo.



## RÚBRICA DE EVALUACIÓN

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	x	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	x	3	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	x	2	

4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	x	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	x	1	
6. Fechas	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	x	1	
7. Ortografía	El documento no muestra errores ortográficos.	2	x	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	x	3	

	TOTAL	20		16	

### Conclusiones

1. La implementación del juego en fases permite un desarrollo organizado y facilita la identificación de errores. Cada commit representa una funcionalidad específica, lo que ayuda en el seguimiento de los cambios realizados.
2. La creación de soldados con atributos aleatorios añade dinamismo al juego, permitiendo que cada ejecución sea única y mantenga el interés del usuario.
3. Mostrar el estado del tablero y los datos del ejército es crucial para la experiencia del usuario. Esto permite que los jugadores entiendan la situación actual del juego y tomen decisiones estratégicas.
4. La clasificación de soldados y la identificación del soldado con mayor vida no solo añaden un componente competitivo, sino que también permiten realizar análisis de rendimiento, lo que puede ser útil para futuras mejoras.

### Metodología de Trabajo

1. Antes de comenzar la codificación, se definieron los objetivos del juego y se esbozaron las funcionalidades necesarias, como la creación de soldados, la visualización del tablero y el ranking de soldados.
2. El código se implementó en fases, comenzando con la creación del tablero y soldados, seguido de la visualización del estado del juego y, finalmente, la lógica de análisis. Esto permite realizar pruebas y ajustes en cada etapa.

3. Se utilizó Git para el control de versiones, lo que permitió registrar cambios en el código, colaborar de manera efectiva y mantener un historial de modificaciones. Cada fase del desarrollo se documentó mediante commits descriptivos.
4. Tras cada implementación, se realizaron pruebas para verificar que las funcionalidades funcionaran como se esperaba. Los errores se corrigieron en el momento, asegurando que el código estuviera siempre en un estado funcional.
5. Se mantuvo una documentación clara de los métodos y funciones implementadas, lo que facilita la comprensión del código tanto para el autor como para futuros colaboradores.

#### REFERENCIAS Y BIBLIOGRAFÍA

*Aedo López, M. W. (2019). Fundamentos de programación I: Java Básico (1. ed.). Universidad Nacional de San Agustín. ISBN 978-612-4337-55-0.*