# Informe de Laboratorio 12

## Tema: Definicion de Clases de Usuario

| Nota |
|---|
| |

| Estudiante | Escuela | Asignatura |
|---|---|---|
| Jhonatan David Arias Quispe<br>jariasq@unsa.edu.pe | Escuela Profesional de Ingeniería de Sistemas | Fundamentos de Programacion 2<br>Semestre: II<br>Código: 1701213 |

| Laboratorio | Tema | Duración |
|---|---|---|
| 12 | Definicion de Clases de Usuario | 04 horas |

| Semestre académico | Fecha de inicio | Fecha de entrega |
|---|---|---|
| 2023 - B | Del 4 Diciembre 2023 | Al 11 Diciembre 2023 |

## 1. Actividades

- Al ejecutar el videojuego, el programa deberá dar las opciones:
- 1. Juego rápido (tal cual como en el laboratorio 11) Al acabar el juego mostrar las opciones de volver a jugar y de volver al menú principal. También se deberá tener la posibilidad de cancelar el juego actual en cualquier momento, permitiendo escoger entre empezar un juego totalmente nuevo o salir al menú principal.
- 2. Juego personalizado: permite festionar ejércitos. Primero se generan los 2 ejércitos con sus respectivos soldados y se muestran sus datos. Luego se tendrá que escoger cuál de los 2 ejércitos se va a gestionar, después se mostrarán las siguientes opciones:
  - Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Eliminar Soldado (no debe permitir un ejército vacío)
  - Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)
  - Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivel-Defensa, vidaActual)
  - Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)
  - Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)

- Ver soldado (Búsqueda por nombre)
- Ver ejército
- Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército
  - Por ejemplo, si ejército tendría 3 soldados:
  - s=s1.sumar(s2).sumar(s3);
  - s es un objeto Soldado nuevo que contendría las sumatorias de los 4 atributos indicados de los 3 soldados. Ningún soldado cambia sus valores
- Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.
- Volver (muestra el menú principal) Después de escoger alguna de las opciones 1) a 9) se podrá volver a elegir uno de los ejércitos y se mostrarán las opciones 1) a 11)

- 3. Salir

## 2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernell
- NeoVim
- OpenJDK 64-Bit 20.0.1
- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Creacion de programas con CLI
- Bilioteca Graphics (origen propio)

## 3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JhonatanDczel/fp2-23b.git`
- URL para el laboratorio 12 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/fp2-23b/tree/main/fase02/lab12`
- El trabajo de este laboratorio es en su mayor parte lo mismo que en otros laboratorios, por lo que las variaciones seran minimas

# 4. Proyecto lab12

- Creamos un directorio en fase02 que contenga los archivos del laboratorio y copiamos los archivos del anterior laboratorio.
- Para el tablero se usará un array bidimensional simple.
- La estructura del laboratorio presente es:

```
>>> lab12  git:(main) tree
.
├── latex
│   ├── acts
│   │   ├── a1.tex
│   │   ├── a2.tex
│   │   ├── a3.tex
│   │   ├── a4.tex
│   │   ├── a5.tex
│   │   ├── commits.tex
│   │   ├── exec.tex
│   │   └── main.tex
│   ├── foot
│   │   ├── rubricas.aux
│   │   └── rubricas.tex
│   ├── head
│   │   ├── codeFormat.tex
│   │   ├── dataTable.tex
│   │   ├── format.tex
│   │   ├── labData.tex
│   │   └── pkgs.tex
│   ├── img
│   │   ├── exec.png
│   │   ├── logo_abet.png
│   │   ├── logo_episunsa.png
│   │   ├── logo_unsa.jpg
│   │   └── tree.jpg
│   ├── informe-latex.aux
│   ├── informe-latex.log
│   ├── informe-latex.out
│   ├── informe-latex.pdf
│   └── informe-latex.tex
├── Soldado.class
├── Soldado.java
├── VideoJuego.class
└── VideoJuego.java

6 directories, 29 files
```

# 5. Clase Soldado

- Los atributos para la clase soldado son:

```
1  private String nombre;
2  private int fila;
3  private int nivelAtaque = random(5);
4  private int nivelDefensa = random(5);
5  private int columna;
6  private int nivelVida;
7  private int vidaActual;
8  private int velocidad;
9  private String actitud;
10 private boolean vive;
11 private String team;
```

- Cada atributo que lo requiere, tiene sus metodos setters y getters para encapsular la información.
- Se usan 3 constructores sobrecargados que son los siguientes:

```java
public Soldado(String t) {
  team = t;
  velocidad = 0;
  vive = true;
  actitud = "ataque";

}
public Soldado(int v, String t) {
  team = t;
  velocidad = v;
  vive = true;
  actitud = "ataque";
}
public Soldado(int v, int nV, String t) {
  team = t;
  vive = true;
  velocidad = v;
  nivelVida = nV;
  actitud = "ataque";
}
```

- Estos constructores nos servirán cuando vayamos a crear soldados con distintos datos base.
- Adicionalmente tenemos los metodos de accion del soldado:

```java
public void atacar() {
  actitud = "ofensiva";
}
public void defender() {
  actitud = "defensiva";
}
public void huir() {
  actitud = "fuga";
  velocidad += 2;
}
public void avanzar() {
  velocidad += 1;
}

public void serAtacado() {
  vidaActual -= 1;
  if(vidaActual == 0) morir();
}
public void morir() {
  vive = false;
}

public void retroceder() {
  if (velocidad > 0) {
    velocidad = 0;
    actitud = "defensiva";
  } else if (velocidad == 0) {
    velocidad = -1;
  }
}
```

- Las acciones cambian los estados de los atributos del soldado, a los que accederemos despues con los metodos accesores:

```java
public String getTeam() {
  return team;
}

public void setNombre(String n) {
  nombre = n;
}

public void setFila(int f) {
  fila = f;
}

public void setColumna(int c) {
  columna = c;
}

public void setNivelVida(int p) {
  nivelVida = p;
}

public String getNombre() {
  return nombre;
}

public int getFila() {
  return fila;
}

public int getColumna() {
  return columna;
}

public int getNivelVida() {
  return nivelVida;
}

public int getNivelAtaque() {
  return nivelAtaque;
}

public int getNivelDefensa() {
  return nivelDefensa;
}

public void setNivelAtaque(int n) {
  nivelAtaque = n;
}

public void setNivelDefensa(int n) {
  nivelDefensa = n;
}

public boolean isLive() {
  return vive;
}
```

- Estos metodos accesores nos servirán para manejar la lógica interna del videojuego
- Adicionalmente tenemos algunos metodos auxiliares que usamos en la misma clase:

```java
public String toString() {
  return "Nombre: " + nombre +
```

```
    " | Ubicacion: " + fila + ", " + columna +
    " | nivelVida: " + nivelVida +
    " | Estado: " + (vive ? "Vivo" : "Muerto") +
    " | Actitud: "+ actitud +"\n" ;
  }
  private int random(int n) {
    return (int) (Math.random() * n + 1);
  }
```

## 6. Clase VideoJuego

La clase VideoJuego contiene la logica principal y es la clase que contiene el metodo main, por lo tanto, el que debería llamarse para ejecutar el juego. La estructura de la clase esta dada de la siguiente manera:

- Metodos:
- juegoRapido
- juegoPersonalizado
- crearSoldado
- eliminarSoldado
- clonarSoldado
- modificarSoldado
- compararSoldado
- intercambiarSoldado
- verSoldado
- verEjercito
- sumarNiveles
- jugar
- volver
- main
- fillTable
- addSoldado
- printTable
- printArr
- random
- printMayorNivel
- printPromedioPuntos
- printPuntosAll

- printSoladosOrdenados
- printRankingPointsBubble
- printRankingPointsSelect
- getWinner
- randomWinner
- mover
- atacar

A continuación veremos la explicación de los metodos principales para el funcionamiento.

# 7. Juego Rapido

El método `juegoRapido()` representa la implementación de un juego estratégico entre dos ejércitos de soldados dispuestos en una matriz bidimensional. A continuación, se proporciona una explicación detallada del código:

## 7.1. Configuración Inicial

La función comienza solicitando al usuario ingresar 'S' si desea salir de la partida. Si se ingresa 'S', la función termina inmediatamente. En caso contrario, se procede a la configuración inicial del juego.

```
Scanner sc = new Scanner(System.in);
System.out.println("Para salir de la partida ingresa 'S'");
String salir = sc.next();
if (salir.toLowerCase().charAt(0) == 's') return;
```

## 7.2. Inicialización de Variables

Se crea una matriz table de dimensiones 10x10 para representar el campo de batalla, y dos listas (ejercito1 y ejercito2) que contendrán instancias de la clase Soldado. La función fillTable() se llama para inicializar la matriz y las listas.

```
Soldado[][] table = new Soldado[10][10];
ArrayList<Soldado> ejercito1 = new ArrayList<Soldado>();
ArrayList<Soldado> ejercito2 = new ArrayList<Soldado>();
fillTable(table, ejercito1, ejercito2);
```

## 7.3. Impresión de Información Inicial

Se imprime en consola el soldado con mayor nivel en cada ejército y, posteriormente, se muestra el promedio de puntos de ambos ejércitos.

```
printMayorNivel(table, ejercito1, ejercito2);
System.out.println("_____________________________________");
System.out.println("IMPRIMIR PROMEDIO DE PUNTOS");
printPromedioPuntos(table, ejercito1, ejercito2);
System.out.println("_____________________________________");
```

### 7.4. Ordenamiento y Ranking

Se imprime la lista de soldados ordenada por algún criterio no especificado y se muestra el ranking de puntos de ambos ejércitos, utilizando algoritmos de ordenamiento Bubble Sort y Select Sort, respectivamente.

```
System.out.println("___________________________________");
System.out.println("SOLDADOS ORDENADOS");
printSoladosOrdenados(table, ejercito1, ejercito2);
System.out.println("___________________________________");
System.out.println("RANKING DE PUNTOS EJERCITO1 POR BUBBLE SORT");
printRankingPointsBubble(table, ejercito1);
System.out.println("___________________________________");
System.out.println("RANKING DE PUNTOS EJERCITO2 POR SELECT SORT");
printRankingPointsSelect(table, ejercito2);
```

### 7.5. Desarrollo del Juego

Se inicia un bucle que representa los turnos del juego. En cada turno, se muestra el estado actual del campo de batalla (printTable(table)) y se realiza un movimiento de los soldados. Se alterna entre los dos ejércitos en cada turno.

```
int turno = 1;
while (true) {
  printTable(table);
  mover(table, ejercito1, ejercito2, turno);

  turno = turno == 1 ? 2 : 1;

  String winner = getWinner(ejercito1, ejercito2);

  if (winner != null) {
    System.out.println("############################");
    System.out.println("GANA EL EQUIPO " + winner);
    System.out.println("############################");
    break;
  }
}
```

### 7.6. Conclusión del Juego

Cuando se determina un ganador (mediante la función getWinner()), se imprime el equipo ganador y se finaliza el juego.

Entonces, el método `juegoRapido()` encapsula la lógica de un juego estratégico entre dos ejércitos de soldados, proporcionando información inicial, estadísticas y una interfaz para que los usuarios participen en turnos alternativos hasta que se determine un ganador.

## 8. Juego Personalizado

El método `juegoPersonalizado` proporciona una interfaz interactiva para que el usuario personalice y gestione los soldados de dos equipos en un juego de estrategia. Cada opción del menú está asociada a una función específica que realiza acciones como crear, eliminar, clonar o modificar soldados, así como visualizar información sobre los mismos y el estado general de los ejércitos. el método también incluye opciones para avanzar al juego o volver al menú principal. A continuación se detalla su estructura interna:

## 8.1. Inicialización de Variables

Se utiliza un objeto `Scanner` para la entrada de datos desde la consola. Se crea una matriz llamada `table` de dimensiones $10 \times 10$ para representar el campo de juego. Se declaran dos `ArrayList` llamados `ejercito1` y `ejercito2` para almacenar los soldados de cada equipo.

```
Scanner sc = new Scanner(System.in);
Soldado[][] table = new Soldado[10][10];
ArrayList<Soldado> ejercito1 = new ArrayList<Soldado>();
ArrayList<Soldado> ejercito2 = new ArrayList<Soldado>();
fillTable(table, ejercito1, ejercito2);
```

## 8.2. Rellenar la Matriz y Listas de Soldados

Se invoca el método `fillTable` para inicializar la matriz y las listas de soldados de ambos equipos.

```
fillTable(table, ejercito1, ejercito2);
```

## 8.3. Impresión de Soldados Ordenados

Se imprime en la consola la información de los soldados ordenados en el campo de juego.

```
System.out.println("SOLDADOS ORDENADOS");
printSoladosOrdenados(table, ejercito1, ejercito2);
System.out.println("###################################");
```

## 8.4. Selección del Equipo a Personalizar

Se solicita al usuario que ingrese el símbolo del equipo (`"*"` o `"#"`) que desea personalizar. Se determina el `ArrayList` del equipo seleccionado mediante un operador ternario.

```
System.out.print("Escoge al equipo a personalizar: ");
String team = sc.next();
ArrayList<Soldado> currentE = team.equals("*") ? ejercito1 : ejercito2;
```

## 8.5. Menú de Opciones

Se presenta un menú con varias opciones numeradas que permiten al usuario realizar acciones específicas en relación con los soldados y equipos.

```
System.out.println("Escoge una opcion: ");
// ... (Listado de opciones)
int input = sc.nextInt();
```

## 8.6. Ejecución de Acciones Según la Opción Seleccionada

Se utiliza una serie de condicionales para determinar la acción a realizar según la opción ingresada por el usuario.

```
if (input == 1) crearSoldado(table, currentE, team);
if (input == 2) eliminarSoldado(table, currentE, team);
// ... (Resto de opciones)
if (input == 10) jugar();
```

```
5 if (input == 11) volver();
```

# 9. Métodos de gestión de Soldados

Estas funciones proporcionan al usuario la capacidad de crear, eliminar, clonar y modificar soldados en su ejército, contribuyendo así a la personalización y estrategia dentro del juego. Cada función interactúa directamente con la matriz que representa el campo de juego y la lista de soldados del equipo correspondiente.

## 9.1. Función `crearSoldado`

**Propósito:** Permite al usuario crear un nuevo soldado para su ejército.
**Parámetros:**

- `Soldado[][] t`: La matriz que representa el campo de juego.
- `ArrayList<Soldado>e`: Lista de soldados del equipo actual.
- `String team`: Símbolo del equipo actual.

**Descripción:**

```
1  public static void crearSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
2      Scanner sc = new Scanner(System.in);
3      Soldado s = new Soldado(team);
4
5      if(e.size() >= 10) {
6        System.out.println("El ejercito tiene el maximo de soladados: ");
7        return;
8      }
9
10     System.out.print("Ingresa el nivel del soldado: ");
11     s.setNivelVida(sc.nextInt());
12     System.out.print("Ingresa el nivel de ataque: ");
13     s.setNivelAtaque(sc.nextInt());
14     System.out.print("Ingresa el nivel de defensa: ");
15     s.setNivelDefensa(sc.nextInt());
16     System.out.print("Ingresa la columna: ");
17     s.setColumna(sc.nextInt());
18     System.out.print("Ingresa la fila: ");
19     s.setFila(sc.nextInt());
20     s.setNombre("Soldado " + (e.size() + 1));
21
22     e.add(s);
23     t[s.getColumna()][s.getFila()] = s;
24
25 }
```

## 9.2. Función `eliminarSoldado`

**Propósito:** Permite al usuario eliminar un soldado de su ejército.
**Parámetros:**

- `Soldado[][] t`: La matriz que representa el campo de juego.
- `ArrayList<Soldado>e`: Lista de soldados del equipo actual.
- `String team`: Símbolo del equipo actual.

**Descripción:**

```java
public static void eliminarSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
    Scanner sc = new Scanner(System.in);

    if(e.size() <= 1) {
      System.out.println("El ejercito nesecita al menos 1 soldado");
      return;
    }

    System.out.print("Ingresa la columna: ");
    int col = sc.nextInt();
    System.out.print("Ingresa la fila: ");
    int fila = sc.nextInt();

    for(int i = 0; i < e.size(); i += 1) {
      if(e.get(i).getColumna() == col && e.get(i).getFila() == fila){
        e.remove(i);
        break;
      }
    }

    t[col][fila] = null;

}
```

## 9.3. Función `clonarSoldado`

**Propósito:** Permite al usuario clonar un soldado existente en una nueva posición.
**Parámetros:**

- `Soldado[][] t`: La matriz que representa el campo de juego.
- `ArrayList<Soldado>e`: Lista de soldados del equipo actual.
- `String team`: Símbolo del equipo actual.

**Descripción:**

```java
public static void clonarSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Ingresa la columna: ");
    int col = sc.nextInt();
    System.out.print("Ingresa la fila: ");
    int fila = sc.nextInt();

    Soldado soldadoOriginal = t[col][fila];

    Soldado soldadoNuevo = new Soldado(soldadoOriginal.getTeam());
    soldadoNuevo.setNivelAtaque(soldadoOriginal.getNivelAtaque());
    soldadoNuevo.setNivelDefensa(soldadoOriginal.getNivelDefensa());
    soldadoNuevo.setNivelVida(soldadoOriginal.getNivelVida());
    soldadoNuevo.setNombre("Soldado " + (e.size() + 1));

    System.out.print("Ingresa la columna nueva: ");
    int newCol = sc.nextInt();
    System.out.print("Ingresa la fila nueva: ");
    int newFila = sc.nextInt();

    e.add(soldadoNuevo);
    t[newCol][newFila] = soldadoNuevo;
}
```

### 9.4. Función `modificarSoldado`

**Propósito:** Permite al usuario modificar las características y posición de un soldado existente.
**Parámetros:**

- `Soldado[][] t:` La matriz que representa el campo de juego.
- `ArrayList<Soldado>e:` Lista de soldados del equipo actual.
- `String team:` Símbolo del equipo actual.

**Descripción:**

```java
public static void modificarSoldado(Soldado[][] t, ArrayList<Soldado> e, String team) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Ingresa la columna: ");
    int col = sc.nextInt();
    System.out.print("Ingresa la fila: ");
    int fila = sc.nextInt();

    Soldado s = t[col][fila];
    System.out.println("Ingresa nivel de vida nuevo: ");
    s.setNivelVida(sc.nextInt());
    System.out.println("Ingresa nivel de ataque nuevo: ");
    s.setNivelAtaque(sc.nextInt());
    System.out.println("Ingresa nivel de defensa nuevo: ");
    s.setNivelDefensa(sc.nextInt());
    System.out.println("Ingresa la columna nueva: ");
    s.setColumna(sc.nextInt());
    System.out.println("Ingresa la fila nueva: ");
    s.setFila(sc.nextInt());
}
```

## 10. Ejecucion grafica y por consola

- A continuacion se vera la Ejecucion tanto grafica como por consola

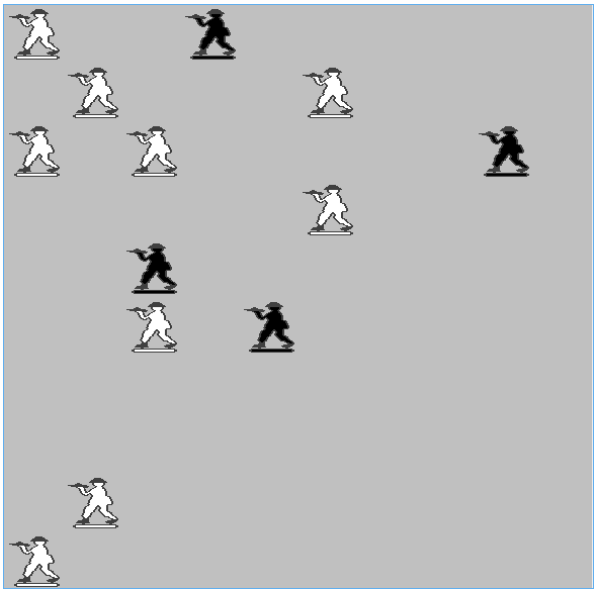
### 10.1. Ejecucion por consola

```
===== Ejercito 1 =====
 Soldado 0x8:
  Nivel de vida: 3
  Fila: 9
  Columna: 2

 Soldado 0x6:
  Nivel de vida: 3
  Fila: 3
  Columna: 1

 Soldado 0x7:
  Nivel de vida: 5
  Fila: 1
  Columna: 1

 Soldado 0x4:
  Nivel de vida: 4
  Fila: 2
  Columna: 2
```

```
21
22  Soldado 0x5:
23   Nivel de vida: 1
24   Fila: 6
25   Columna: 3
26
27  Soldado 0x2:
28   Nivel de vida: 4
29   Fila: 4
30   Columna: 6
31
32  Soldado 0x3:
33   Nivel de vida: 3
34   Fila: 10
35   Columna: 1
36
37  Soldado 0x0:
38   Nivel de vida: 4
39   Fila: 2
40   Columna: 6
41
42  Soldado 0x1:
43   Nivel de vida: 2
44   Fila: 3
45   Columna: 3
46
47
48 ===== Ejercito 2 =====
49  Soldado 1x3:
50   Nivel de vida: 1
51   Fila: 1
52   Columna: 4
53
54  Soldado 1x1:
55   Nivel de vida: 3
56   Fila: 5
57   Columna: 3
58
59  Soldado 1x2:
60   Nivel de vida: 4
61   Fila: 3
62   Columna: 9
63
64  Soldado 1x0:
65   Nivel de vida: 1
66   Fila: 6
67   Columna: 5
68
69 Soldado con maxima vida:
70  Soldado 0x7:
71   Nivel de vida: 5
72   Fila: 1
73   Columna: 1
74
75
76 ===== Ranking de soldados: =====
77  Soldado 0x7:
78   Nivel de vida: 5
79   Fila: 1
80   Columna: 1
81
82  Soldado 0x2:
83   Nivel de vida: 4
84   Fila: 4
85   Columna: 6
```

```
Soldado 1x2:
 Nivel de vida: 4
 Fila: 3
 Columna: 9

Soldado 0x0:
 Nivel de vida: 4
 Fila: 2
 Columna: 6

Soldado 0x4:
 Nivel de vida: 4
 Fila: 2
 Columna: 2

Soldado 1x1:
 Nivel de vida: 3
 Fila: 5
 Columna: 3

Soldado 0x8:
 Nivel de vida: 3
 Fila: 9
 Columna: 2

Soldado 0x6:
 Nivel de vida: 3
 Fila: 3
 Columna: 1

Soldado 0x3:
 Nivel de vida: 3
 Fila: 10
 Columna: 1

Soldado 0x1:
 Nivel de vida: 2
 Fila: 3
 Columna: 3

Soldado 1x3:
 Nivel de vida: 1
 Fila: 1
 Columna: 4

Soldado 0x5:
 Nivel de vida: 1
 Fila: 6
 Columna: 3

Soldado 1x0:
 Nivel de vida: 1
 Fila: 6
 Columna: 5
```

- Como vemos, la estructura del metodo main esta representada en la salida por consola
- A continuacion se vera la salida grafica:



# 11. Commits mas importantes

- A continuacion se muestran los commits mas importantes
- Los commits se hicieron siguiendo la convencion para commits de git, y siguiendo las recomendaciones practicas para hacer mensajes de commits (Mensajes de forma imperativa)

### Commit 7f2adda566d664a4c7c38300c68c287c27e220d6

**Autor:** JhonatanDczel
**Descripción:** Implementa el método "getWinner"que determina el ganador en el juego.

### Commit 22d7b9aac5a07bcc41552fbe65401617f11164db

**Autor:** JhonatanDczel
**Descripción:** Agrega el método para imprimir el ranking de soldados por puntos, proporcionando una visión general del rendimiento de los soldados.

### Commit 883c7b61657b46f8add303a013decb7c71167e3f

**Autor:** JhonatanDczel
**Descripción:** Introduce métodos para imprimir promedios y puntos, contribuyendo a la visualización estadística de los soldados.

### Commit b7aecfc012fda83b168a8a0b318f4e80f528d68f

**Autor:** JhonatanDczel
**Descripción:** Implementa el método principal "main,"que sirve como punto de entrada y coordinación de la aplicación.

### Commit 8c95e9be0340d97b729310b1ca0d1ca1a372968e

**Autor:** JhonatanDczel
**Descripción:** Desarrolla la lógica para modificar un soldado existente en el juego.

### Commit 70e2bad4bbd4bddd3d02ce881d90c947d2a302fb

**Autor:** JhonatanDczel
**Descripción:** Implementa la funcionalidad para eliminar un soldado del ejército.

### Commit 496c05baf98c4b3233cbcc33c5b20323e2dba4f9

**Autor:** JhonatanDczel
**Descripción:** Agrega la funcionalidad para crear un nuevo soldado y añadirlo al ejército.

### Commit 89bcc0118d1dd84f6edb69549684e1767a0e77f1

**Autor:** JhonatanDczel
**Descripción:** Introduce el modo de juego personalizado, proporcionando opciones para personalizar y gestionar los ejércitos.

### Commit 274cf853e8a6b713a3ee1a2157103a431cf30009

**Autor:** JhonatanDczel
**Descripción:** Agrega la funcionalidad inicial, marcando el punto de partida del desarrollo del juego.

### Commit 1740eabf3958fed46ecd4be9987d88a37a93d8a2

**Autor:** JhonatanDczel
**Descripción:** Configura los getters y setters para la clase Soldado, estableciendo la estructura básica de la entidad.

### Commit 0c4a358709095cfe75a19e87c1eb852ca20152e1

**Autor:** JhonatanDczel
**Descripción:** Crea métodos básicos de acciones, sentando las bases para las operaciones relacionadas con los soldados en el juego.

### Commit 5be9489569976460f460a45809a7f17eedf1416a

**Autor:** JhonatanDczel
**Descripción:** Agrega métodos para el constructor sobrecargado, permitiendo la creación de soldados con parámetros específicos.

### Commit a7fb4575d45eaadc5f8a62a5e5baaa9c956c53a4

**Autor:** JhonatanDczel
**Descripción:** Introduce atributos para la clase Soldado, estableciendo las propiedades básicas de un soldado en el juego.

## Commit 41f67e32852f4721ae938bbf9d0187685ff05141

**Autor:** JhonatanDczel
**Descripción:** Crea los archivos para el laboratorio, marcando el inicio del proyecto y la organización de la estructura inicial del código.

# 12. Rúbricas

## 12.1. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: Niveles de desempeño

| | Nivel | | | |
|---|---|---|---|---|
| **Puntos** | Insatisfactorio 25 % | En Proceso 50 % | Satisfactorio 75 % | Sobresaliente 100 % |
| **2.0** | 0.5 | 1.0 | 1.5 | 2.0 |
| **4.0** | 1.0 | 2.0 | 3.0 | 4.0 |

Tabla 2: Rúbrica para contenido del Informe y demostración

| 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 | 4 | |
| **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 | 1.5 | |
| **5. Pregunta** | Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación). | 2 | X | 2 | |
| **6. Fechas** | Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos. | 2 | X | 2 | |
| **7. Ortografía** | El documento no muestra errores ortográficos. | 2 | X | 1.5 | |
| **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 | 4 | |
| **Total** | | 20 | | 19 | |