

# Informe de Laboratorio 12

## Tema: Laboratorio 12

Nota

Estudiante	Escuela	Asignatura
Victor Mamani Anahua vmamanian@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20230489

Laboratorio	Tema	Duración
12	Laboratorio 12	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 04 Diciembre 2023	Al 11 Diciembre 2023

### 1. Tarea

- Cree un Proyecto llamado Laboratorio12.
- Puede reutilizar todo el código del laboratorio 11, pero ahora el objetivo es gestionar los ejércitos autogenerados.
- 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 gestionar 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:
    - -1- Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)s.
    - -2- Eliminar Soldado (no debe permitir un ejército vacío)
    - -3- Clonar Soldado (crea una copia exacta del soldado) y se añade al final del ejército (recordar que límite es de 10 soldados por ejército)
    - -4- Modificar Soldado (con submenú para cambiar alguno de los atributos nivelAtaque, nivelDefensa, vidaActual)

- -5- Comparar Soldados (verifica si atributos: nombre, nivelAtaque, nivelDefensa, vidaActual y vive son iguales)
- -6- Intercambiar Soldados (intercambia 2 soldados en sus posiciones en la estructura de datos del ejército)
- -7- Ver soldado (Búsqueda por nombre)
- -8- Ver ejército
- -9- Sumar niveles (usando Method-Call Chaining), calcular las sumatorias de nivelVida, nivelAtaque, nivelDefensa, velocidad de todos los soldados de un ejército 1. Por ejemplo, si ejército tendría 3 soldados: 2. `s=s1.sumar(s2).sumar(s3)`; 3. `s` es un objeto Soldado nuevo que contendría las sumatorias de los 4 atributos indicados de los 3 soldados. Ningún soldado cambia sus valores
- -10- Jugar (se empezará el juego con los cambios realizados) y con las mismas opciones de la opción 1.
- -11- 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-

## 2. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux 23 lunar 64 bits Kernell 6.2.v
- Visual Studio Code.
- VIM 9.0.
- OpenJDK 64-Bits 19.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Actividades del Laboratorio 12.

## 3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/VictorMA18/fp2-23b.git>
- URL para el laboratorio 12 en el Repositorio GitHub.
- <https://github.com/VictorMA18/fp2-23b/tree/main/Fase02/Lab12>

## 4. Actividades del Laboratorio 12

### 4.1. Ejercicio Soldado

- En el primer commit agregando la clase soldado que esta siendo reutilizada para poder reslizar el siguiente ejercicio.
- El codigo y el commit seria el siguiente:

Listing 1: Commit

```
$ git commit -m "Agregando la clase soldado para poder reslizar el siguiente ejercicio "
```

Listing 2: Las lineas de codigos del metodo creado:

```
// Laboratorio Nro 12 - Ejercicio Soldado
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
import java.util.*;
public class Soldado { //CREAMOS LA CLASE SOLDADO PARA PODER USAR UN ARREGLO
    BIDIMENSIONAL DONDE NECESITAMOS LA VIDA , EL NOMBRE DEL SOLDADO Y TAMBIEN SU
    POSICION COMO LA FILA Y LA COLUMNA

    private String name;
    private int lifeactual;
    private int row;
    private String column;
    private int attacklevel;
    private int defenselevel;
    private int lifelevel;
    private int speed;
    private String attitude;
    private boolean lives;

    Random rdm = new Random();

    //Anadiendo metodo que nos permita que un arreglo tenga datos nulos si este esta
    vacio
    public Soldado(){
        this.name = "";
        this.row = 0;
        this.column = "";
        this.attacklevel = 0;
        this.defenselevel = 0;
        this.lifelevel = 0;
        this.lifeactual = 0;
        this.speed = 0;
        this.attitude = "";
        this.lives = false;
    }

    //Constructor
    public Soldado(String name, int health, int row, String column){
        this.name = name;
        this.lifeactual = health;
```

```
this.lifelevel = health;
this.lifeactual = health;
this.row = row;
this.column = column;
this.lives = true;

//YA QUE ESTOS DATOS SERIAN ALEATORIOS YA QUE SE ESTARIA CREANDO EL SOLDADO
//TENDRIAMOS DATOS QUE SERIAN COMO ATTACKLEVEL DEFENSELEVEL EL CUAL TENDRIAN
//QUE SER ALEATORIOS
this.attacklevel = rdm.nextInt(5) + 1;
this.defenselevel = rdm.nextInt(5) + 1;

}

//Constructor para los diferentes niveles como de vida defensa ataque velocidad
public Soldado(String name, int attacklevel, int defenselevel, int lifelevel, int
    speed, boolean lives, int row, String column) {
    this.name = name;
    this.attacklevel = attacklevel;
    this.defenselevel = defenselevel;
    this.lifelevel = lifelevel;
    this.speed = speed;
    this.lives = lives;
    this.row = row;
    this.column = column;
}

//Metodos necesarios como avanzar defender huir al ser atacado al retroceder
public void advance(){
    this.speed = getSpeed() + 1;
    System.out.println("El soldado " + this.name + "avanzo");
}
public void defense(){
    this.speed = 0;
    this.attitude = "DEFENSIVA";
    System.out.println("El soldado " + this.name + "esta defendiendo");
}
public void flee(){
    this.speed = getSpeed() + 2;
    this.attitude = "HUYE";
    System.out.println("El soldado " + this.name + "esta huyendo");
}
public void back(){
    System.out.println("El soldado " + this.name + "esta retrocediendo");
    if(this.speed == 0){
        this.speed = rdm.nextInt(5) - 5;
    }else{
        if(this.speed > 0){
            this.speed = 0;
            this.attitude = "DEFENSIVA";
        }
    }
}
}

public void attack(Soldado soldier){
    if(this.getLifeActual() > soldier.getLifeActual()){
        int life = this.getLifeActual() - soldier.getLifeActual();
```

```
        this.setLifeActual(life);
        this.setLifeLevel(life);
        soldier.lives = false;
        soldier.morir();
        System.out.println(this.name + " asesino al soldado " + soldier.name);
    }else if(soldier.getLifeActual() > this.getLifeActual()){
        int life = soldier.getLifeActual() - this.getLifeActual();
        this.lives = false;
        this.morir();
        soldier.setLifeActual(life);
        soldier.setLifeLevel(life);
        System.out.println(soldier.name + " asesino al soldado " + this.name);
    }else{
        this.lives = false;
        this.morir();
        soldier.lives = false;
        soldier.morir();
        System.out.println("los 2 soldados se asesinaron");
    }
}

public void morir(){
    this.lives = false;
    this.attitude = "SOLDADO MUERTO";
}

// Metodos mutadores
public void setName(String n){
    name = n;
}

public void setLifeActual(int p){
    lifeactual = p;
}

public void setRow(int b){
    row = b;
}

public void setColumn(String c){
    column = c;
}

public void setAttackLevel(int attacklevel) {
    this.attacklevel = attacklevel;
}

public void setDefenseLevel(int defenselevel) {
    this.defenselevel = defenselevel;
}

public void setLifeLevel(int lifelevel){
    this.lifelevel = lifelevel;
}

public void setSpeed(int speed) {
    this.speed = speed;
}

public void setAttitude(String attitude) {
    this.attitude = attitude;
}

public void setLives(boolean lives) {
    this.lives = lives;
}
```

```
// Metodos accesorios
public String getName(){
    return name;
}
public int getLifeActual(){
    return lifeactual;
}
public int getRow(){
    return row;
}
public String getColumn(){
    return column;
}
public int getAttackLevel() {
    return attacklevel;
}
public int getDefenseLevel() {
    return defenselevel;
}
public int getLifeLevel(){
    return lifelevel;
}
public int getSpeed() {
    return speed;
}
public String getAttitude() {
    return attitude;
}
public boolean getLives() {
    return lives;
}

// Completar con otros metodos necesarios
public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEL OBJETO
    String join = "\nNombre: " + getName() + "\nVida: " + getLifeActual() + "\nFila: " +
        getRow() + "\nColumna: " + getColumn() + "\nNivel de ataque: " +
        getAttackLevel() + "\nNivel de Defensa: " + getDefenseLevel() + "\nNivel de
        vida: " + getLifeLevel() + "\nVelocidad: " + getSpeed() + "\nActitud: " +
        getAttitude() + "\nEstado: " + getLives(); //Agregamos un espaciador para
        poder separar
    return join;
}
}
```

#### 4.2. Ejercicio Videojuego2

- En el segundo commit ponemos las cosas necesarias que se requirio para el lab12 que era que tenemos que reutilizar el lab11 y en el lab11 se dice que las batallas entre 2 soldados sse define por su nivel de probabilidad asi imprimendolo para mostrar el resultado de esa batalla
- El codigo y el commit seria el siguiente:

Listing 3: Commit

```
$ git commit -m "Poniendo las funciones del laboratorio 11 las cuales se pedian para
hacer este lab por ejemplo las batallas entre 2 soldados y el resultado de estas
mediante porcentajes de probabilidad"
```

Listing 4: Las lineas de codigos del metodo creado:

```
if(health2 > health1){
    double sumhealth = (health2 + health1) * 1.0;
    System.out.println("\n \t Resultado de la Batalla");
    System.out.println("El soldado del bando X es el ganador ya que su probabilidad de
    ganar la batalla es --> " + String.format( "%.1f" , ((health2/sumhealth) *
    1000 ) / 10) + "% y la probabilidad del soldado del bando Y es ---> " +
    String.format( "%.1f" , ((health1/sumhealth) * 1000) / 10) + "%" );
}
else if(health1 > health2){
    double sumhealth = (health2 + health1) * 1.0;
    System.out.println("\n \t Resultado de la Batalla");
    System.out.println("El soldado del bando Y es el ganador ya que su probabilidad de
    ganar la batalla es --> " + String.format( "%.1f" , ((health1/sumhealth) * 1000
    ) / 10) + "% y la probabilidad del soldado del bando X es ---> " +
    String.format( "%.1f" , ((health2/sumhealth) * 1000) / 10) + "%" );
}
```

Mostrando tabla de posicion ... --  
Leyenda: Ejercito1 --> X | Ejercito2 --> Y

	A	B	C	D	E	F	G	H
1								
2	X							
3								
4								
5								
6	X							
7							X	X
8								
9	Y							
10								

\*\*\*\*\*  
EMPIEZA EL JUGADOR CON EL BANDO --X--  
TIENE QUE SELECCIONAR UNA OPCION  
1 : MOVER SOLDADO  
2 : SALTAR TURNO  
1

-Seleccione el soldado:  
Fila: 6  
Columna: A

-Ingresa la casilla a mover  
Fila: 9  
Columna: A

Resultado de la Batalla  
El soldado del bando X es el ganador ya que su probabilidad de ganar la batalla es --> 55.6% y la probabilidad del soldado del bando Y es ----> 44.4%

\* Puede reutilizar todo el código del laboratorio 11, pero ahora el objetivo es gestionar los ejércitos autogenerados.

\* Al ejecutar el videojuego, el programa deberá dar las opciones:

1. Jugar juego (tal cual como en el laboratorio 11)
2. Jugar personalizado: permite gestionar 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:

- 1) Crear Soldado: permitirá crear un nuevo soldado personalizado y añadir al final del ejército (recordar que límite es de 10 soldados por ejército)
- 2) Eliminar Soldado (no debe permitir un ejército vacío)
- 3) Clonar Soldado (crea una copia exacta del soldado) y se añade al ejército





```
Mostrando tabla de posicion ... --
Leyenda: Ejercito1 --> X | Ejercito2 --> Y
E A B C D E F G H I J
1
2
3
4
5
6
7
8
9
10

*****
\subsection{Estructura de laboratorio 12}
\begin{itemize}
-- OPCIONES DE BATALLA --
-- El contenido que se entrega en este laboratorio12 es el siguiente:
\end{itemize}
SELECCIONE UN NUMERO PARA PODER EMPEZAR O TERMINAR
1 : JUEGO RAPIDO
2 : JUEGO PERSONALIZADO
1
-----
\begin{code}
-- MENU PRINCIPAL
\end{code}
SELECCIONE UN NUMERO PARA PODER EMPEZAR O TERMINAR
1 : JUGAR
2 : NO JUGAR
1
Mostrando tabla de posicion ... --
Leyenda: Ejercito1 --> X | Ejercito2 --> Y
```

#### 4.4. Ejercicio Videojuego2

- En el cuarto commit agregamos opciones para el juego personalizado estas serian como la primera opcion la cual nos ayuda a crear un soldado para el ejercito para esto hacemos un do while el cual nos ayudara con las opciones que quiera escoger el jugador y en esta opcion de crear el jugador ponemos la condicion de que no sobrepase el maximo de soldados el cual seria 10 soldados para este soldado creado le ponemos un nombre una vida y su ubicacion en el mapa.
- El codigo y el commit seria el siguiente:

Listing 7: Commit

```
$ git commit -m "Creando la primera opcion del Juego personalizado"
```

Listing 8: Las lineas de codigos del metodo creado:

```
public static void battlePersonalized(ArrayList<ArrayList<Soldado>> army1 ,
    ArrayList<ArrayList<Soldado>> army2){
    Scanner sc = new Scanner(System.in);
    System.out.println("Gestionar Ejercito \n[1] Ejercito 1\n[2] Ejercito 2");
    int optarmy = sc.nextInt();
    do {
        System.out.println("Escoja una de estas opciones para el ejercito 1");
        System.out.println("[1] Crear Soldado"+
            "\n[2] Eliminar Soldado" +
            "\n[3] Clonar Soldado"+
            "\n[4] Modificar Soldado"+
            "\n[5] Comparar Soldados"+
            "\n[6] Intercambiar Soldados"+
            "\n[7] Ver soldado"+
```

```
        "\n[8] Ver ejercito"+
        "\n[9] Sumar Niveles"+
        "\n[10] Jugar" +
        "\n[11] Volver");
int optPersonalized = sc.nextInt();
switch (optPersonalized) {
    case 1:
        int numbersoldiers = 0;
        for(int i = 0; i < 10; i++){ //ITERACION CREADA PARA PODER SABER QUE SI ESTE
            BANDO DEL EJERCITO TIENE SOLDADOS PARA PODER JUGAR SI TIENE 10 ESTA
            OPCION ESTA CANCELADA
            for(int j = 0; j < 10; j++){
                if(army1.get(i).get(j) != null){
                    numbersoldiers++;
                }
            }
        }
        if(numbersoldiers == 10){
            System.out.println("USTED NO PUEDE CREAR MAS SOLDADOS EL MAXIMO ES 10
            SOLDADOS POR EJERCITO");
        }else{
            String name = sc.next();
            int health = sc.nextInt();
            int row = sc.nextInt() - 1;
            String column = sc.next();
            Soldado soldier = new Soldado(name, health, numbersoldiers, column);
            army1.get(row).set((int)column.charAt(0) - 65, soldier);
        }
        viewBoard(army1, army2);
        break;
    default:
        break;
}

} while (optarmy == 1);
}
```

\*\*\*\*\*

OPCIONES DE BATALLA

SELECCIONE UN NUMERO PARA PODER EMPEZAR O TERMINAR

1 : JUEGO RAPIDO

2 : JUEGO PERSONALIZADO

Gestionar Ejercito

[1] Ejercito 1

[2] Ejercito 2

1

Escoja una de estas opciones para el ejercito 1

[1] Crear Soldado

[2] Eliminar Soldado

[3] Clonar Soldado

[4] Modificar Soldado

[5] Comparar Soldados

[6] Intercambiar Soldados

[7] Ver soldado

[8] Ver ejercito

[9] Sumar Niveles

[10] Jugar

[11] Volver

1

TR

2

2

2

B

ESQUEMA

LÍNEA DE TIEMPO

```
Mostrando tabla de posicion ... --
Leyenda: Ejercito1 --> X | Ejercito2 --> Y

E  A  B  C  D  E  F  G  H  I  J
1  |   |   |   |   |   |   | X  | X  |   |
2  |   | X  | Y  |   |   |   |   |   |   |
3  | X  |   |   |   |   |   |   |   | X  |
4  |   |   |   |   | X  |   |   |   |   | Y  |
5  |   |   |   |   |   | Y  |   |   |   |   |
6  | Y  |   |   |   |   |   |   |   |   |   |
7  |   | Y  |   |   |   | X  |   |   |   |   |
8  |   |   |   |   |   |   |   |   |   |   |
9  |   | X  |   | Y  |   |   |   |   |   | X  |
10 |   |   |   | X  |   |   |   |   |   |   |

*****
Escoja una de estas opciones para el ejercito 1
[1] Crear Soldado
[2] Eliminar Soldado
[3] Clonar Soldado
[4] Modificar Soldado
[5] Comparar Soldados
[6] Intercambiar Soldados
[7] Ver soldado
[8] Ver ejercito
[9] Sumar Niveles
[10] Jugar
[11] Volver
1
USTED NO PUEDE CREAR MAS SOLDADOS EL MAXIMO ES 10 SOLDADOS POR EJERCITO
```

## 4.5. Ejercicio Videojuego2

- En el quinto commit agregamos la opcion numero 2 la cual nos va poder permitir eliminar a un soldado del ejercito para esto hacemos una comprobacion y eliminamos al soldado requerido mediante su ubicacion con eso vaciariamos su posicion y tambien con esta opcion no permitimos que este ejercito se autoelimine por no tener soldados anquesea vamos a dejar 1 soldado en sus filas.
- El codigo y el commit seria el siguiente:

Listing 9: Commit

```
$ git commit -m "Dando la opcion numero 2 al juego persolnalizado la cual es eliminar a un soldado del ejercito pero sin que este ejercito se vacie para esto hacemos una condicion la cual nos va poder decir que esta anquesea va a tener 1 soldado en sus filas"
```

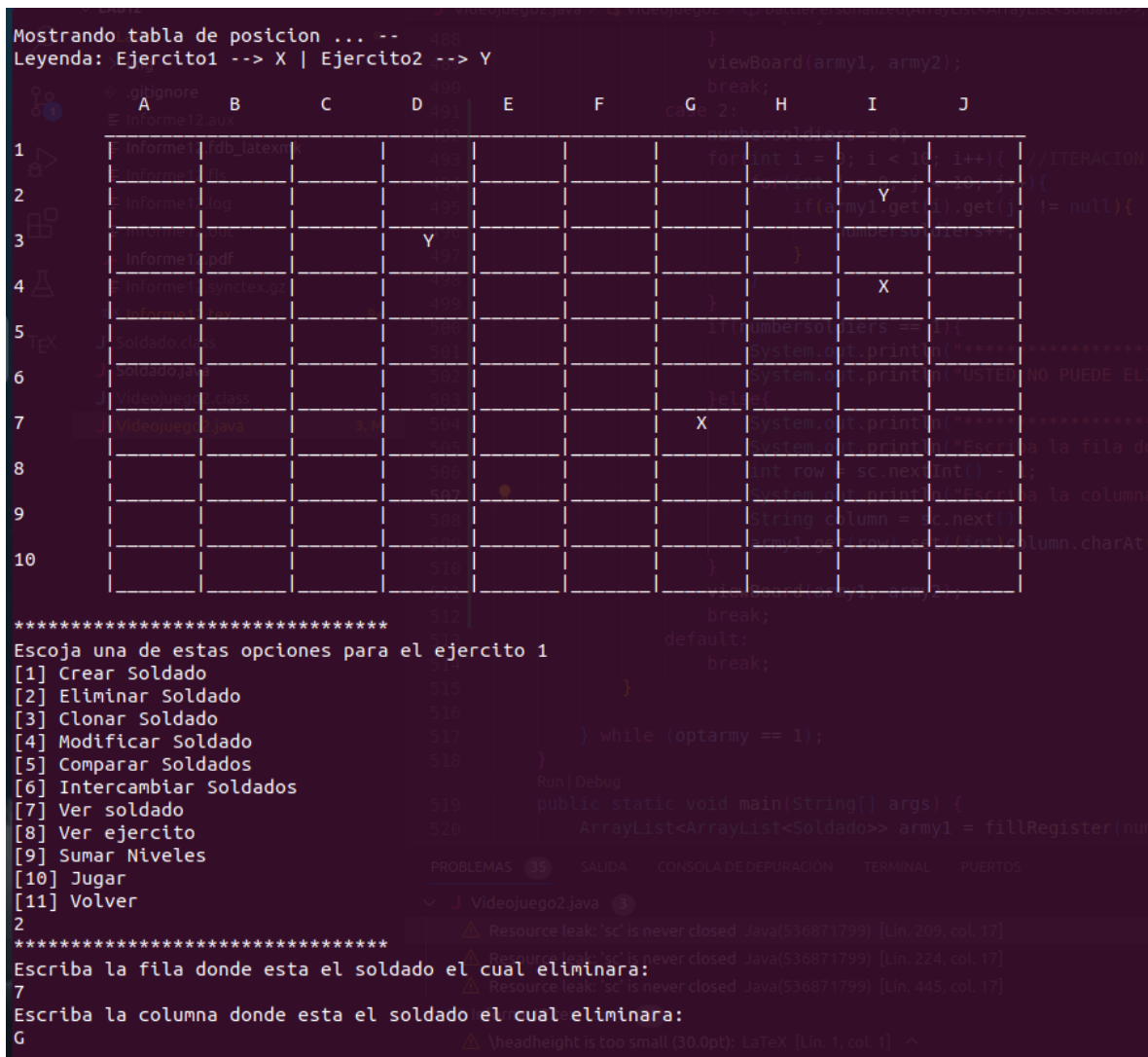
Listing 10: Las lineas de codigos del metodo creado:

```
case 2:
    numbersoldiers = 0;
    for(int i = 0; i < 10; i++){ //ITERACION CREADA PARA PODER SABER QUE SI ESTE BANDO
        DEL EJERCITO TIENE SOLDADOS PARA PODER JUGAR SI TIENE 10 ESTA OPCION ESTA
        CANCELADA
        for(int j = 0; j < 10; j++){
            if(army1.get(i).get(j) != null){
                numbersoldiers++;
            }
        }
    }
    if(numbersoldiers == 1){
```

```

System.out.println("*****");
System.out.println("USTED NO PUEDE ELIMINAR MAS SOLDADOS YA QUE ELIMINARIA A SU
    EJERCITO");
}else{
    System.out.println("*****");
    System.out.println("Escriba la fila donde esta el soldado el cual eliminara:");
    int row = sc.nextInt() - 1;
    System.out.println("Escriba la columna donde esta el soldado el cual eliminara:");
    String column = sc.next();
    army1.get(row).set((int)column.charAt(0) - 65, null);
}
viewBoard(army1, army2);
break;

```



Mostrando tabla de posicion ... --

Leyenda: Ejercito1 --> X | Ejercito2 --> Y

	A	B	C	D	E	F	G	H	I	J
1										
2										
3				Y						
4									X	
5										
6										
7							X			
8										
9										
10										

\*\*\*\*\*

Escoja una de estas opciones para el ejercito 1

- [1] Crear Soldado
- [2] Eliminar Soldado
- [3] Clonar Soldado
- [4] Modificar Soldado
- [5] Comparar Soldados
- [6] Intercambiar Soldados
- [7] Ver soldado
- [8] Ver ejercito
- [9] Sumar Niveles
- [10] Jugar
- [11] Volver

2

\*\*\*\*\*

Escriba la fila donde esta el soldado el cual eliminara:

7

Escriba la columna donde esta el soldado el cual eliminara:

G



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

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: 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 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	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	2	
<b>Total</b>		20		18	

## 6. Referencias

- <https://drive.google.com/drive/u/1/folders/19TzLF0-T77qG7b0Wmg50H7FXAMD2CrJL>