Informe de Laboratorio 12

Tema: Definición de Clases de Usuario

Nota

Estudiante	Escuela	Asignatura
Hernan Andy Choquehuanca Zapana hchoquehuancaz@unsa.edu.pe	Escuela Profesional de Ingenieria de Sistemas	Fundamentos de la Programacion II Semestre: II Código: 1701213

Laboratorio	Tema	Duración
12	Definición de Clases de Usuario	06 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 06 Diciembre 2023	Al 11 Diciembre 2023

1. Tarea

- 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)
  - 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)

- 3. Salir

2. Equipos, materiales y temas utilizados

- Sistema Operativo Windows 11 Pro 22H2 64 bits.
- Visual Studio Code.
- Git 2.42.0.
- Cuenta en GitHub con el correo institucional.
- Editor LaTeX en línea Overleaf.
- Variables Simples
- Métodos.
- Métodos de Búsqueda y Ordenamiento.
- HashMap

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/hernanchoquehuanca/fp2-23b.git`
- URL para el laboratorio 07 en el Repositorio GitHub.
- `https://github.com/hernanchoquehuanca/fp2-23b/tree/main/fase02/lab12`

4. Trabajo del Laboratorio 12

4.1. Clase Soldado.java

Listing 1: Commit 9172d39: Se reutilizo las clases del laboratorio anterior

```
$ git add .
$ git commit -m "Se reutilizo las clases del laboratorio anterior"
$ git push -u origin main
```

- La clase `Soldado` se ha diseñado para representar a un soldado en un juego. Seguidamente, se describirán sus principales atributos y métodos:
  - `private VideoJuego6 videoJuego6`: Una referencia al objeto de la clase `VideoJuego6`.
  - `private String name`: El nombre del soldado.
  - `private int row`: La fila en la que se encuentra el soldado.
  - `private char column`: La columna en la que se encuentra el soldado.
  - `private char team`: El equipo al que pertenece el soldado.
  - `private int position`: La posición del soldado.
  - `private int attackLevel`: El nivel de ataque del soldado.
  - `private int levelDefense`: El nivel de defensa del soldado.
  - `private int levelLife`: El nivel de vida original del soldado.
  - `private int actualLife`: La vida actual del soldado.
  - `private int speed`: La velocidad del soldado.
  - `private String attitude`: La actitud del soldado.
  - `private boolean lives`: Indica si el soldado está vivo o no.

```
2  private VideoJuego6 videoJuego6;
3  private String name;
4  private int row;
5  private char column;
6  private char team;
7  private int position;
8
9  private int attackLevel;
10 private int levelDefense;
11 private int levelLife;
12 private int actualLife;
13 private int speed;
14 private String attitude;
15 private boolean lives;
```

Listing 2: Commit 903d1c9: Agregando los atributos solicitados a la clase Soldado

```
$ git add .
$ git commit -m "Agregando los atributos solicitados a la clase Soldado"
$ git push -u origin main
```

- Se han implementado dos constructores para la clase `Soldado`:
  - El primer constructor recibe todos los atributos como parámetros, inicializa los valores y asigna la referencia al objeto `VideoJuego6`.

```
17  public Soldado(VideoJuego6 videoJuego6,String name, int row, char column, char team,
18              int attackLevel, int levelDefense, int levelLife, int speed,
19              String attitude, boolean lives){
20    this.name = name;
21    this.row = row;
22    this.column = column;
23    this.team = team;
24    this.attackLevel = attackLevel;
25    this.levelDefense = levelDefense;
26    this.levelLife = levelLife;
27    this.actualLife = levelLife;
28    this.speed = speed;
29    this.attitude = attitude;
30    this.lives = lives;
31    this.videoJuego6 = videoJuego6;
32  }
```

  - El segundo constructor es una sobrecarga que asume que el soldado está vivo (`lives` establecido como `true`) y no recibe el parámetro `attitude`.

```
34  public Soldado(VideoJuego6 videoJuego6,String name, int row, char column, char team,
35              int attackLevel, int levelDefense, int levelLife, int speed){
36    this.name = name;
37    this.row = row;
38    this.column = column;
39    this.team = team;
40    this.attackLevel = attackLevel;
41    this.levelDefense = levelDefense;
42    this.levelLife = levelLife;
43    this.actualLife = levelLife;
44    this.speed = speed;
45    this.lives = true;
46    this.videoJuego6 = videoJuego6;
47  }
```

Listing 3: Commit 8afc11f: Segundo avance del menu personalizado, se agrego un segundo constructor e implemento crear y eliminar soldados

```
$ git add .
$ git commit -m "Segundo avance del menu personalizado, se agrego un segundo constructor e
    implemento crear y eliminar soldados"
$ git push -u origin main
```

- Luego de ello también se implementaron los getters y setters para cada atributo de nuestra clase Soldado.
  - Setters:

```
49  //Setters
50  public void setName(String n){
51    name = n;
52  }
53  public void setRow(int r){
54    row = r;
55  }
56  public void setColumn(char c){
57    column = c;
58  }
59  public void setTeam(char t){
60    team = t;
61  }
62  public void setPosition(char p){
63    position = p;
64  }
65
66  public void setAttackLevel(int al){
67    attackLevel = al;
68  }
69  public void setLevelDefense(int ad){
70    levelDefense = ad;
71  }
72  public void setLevelLife(int ll){
73    levelLife = ll;
74  }
75  public void setActualLife(int al){
76    actualLife = al;
77  }
78  public void setSpeed(int s){
79    speed = s;
80  }
81  public void setAttitude(String a){
82    attitude = a;
83  }
84  public void setLives(boolean l){
85    lives = l;
86  }
```

- Getters:

```
88  //Getters
89  public String getName(){
90    return name;
91  }
92  public int getRow(){
93    return row;
94  }
95  public char getColumn(){
96    return column;
97  }
98  public char getTeam(){
99    return team;
100 }
101 public int getPosition(){
102   return position;
103 }
104
105 public int getAttackLevel(){
106   return attackLevel;
107 }
108 public int getLevelDefense(){
109   return levelDefense;
110 }
111 public int getLevelLife(){
112   return levelLife;
113 }
114 public int getActualLife(){
115   return actualLife;
116 }
117 public int getSpeed(){
118   return speed;
119 }
120 public String getAttitude(){
121   return attitude;
122 }
123 public boolean getLives(){
124   return lives;
125 }
126 public VideoJuego6 getVideoJuego6() {
127   return videoJuego6;
128 }
```

Listing 4: Commit 7c8c0c0: Se completaron métodos que se utilizarán en la clase principal VideoJuego6.java

```
$ git add .
$ git commit -m "Completando los metodos de la clase soldado, ademas de adaptar el
    contructor principal"
$ git push -u origin main
```

- El método `toString()` se ha sobreescrito para proporcionar una representación en cadena de los atributos del soldado.

```
130 @Override
131 public String toString() {
132   return "Data { " +
133             "\n Name:  "        + name         +
134             "\n Row:   "        + row          +
135             "\n Column: "       + column       +
136             "\n Team: "         + team         +
137             "\n AttackLevel: "  + attackLevel  +
138             "\n LevelDefense: " + levelDefense +
139             "\n LevelLife: "    + levelLife    +
140             "\n ActualLife: "   + actualLife   +
141             "\n Speed: "        + speed        +
142             "\n Attitude: "     + attitude     +
143             "\n Lives: "        + lives        +
144             "\n}\n";
145
146 }
```

- Se han implementado varios métodos de acción para el soldado, como `attack()`, `defend()`, `advance()`, `back()`, `beAttacked()`, `flee()` y `die()`.

```
148 public void attack(){
149   advance();
150 }
151 public void defend(){
152   speed = 0;
153 }
154 public void advance(){
155   speed++;
156 }
157 public void back(){
158   if (speed > 0)
159     defend();
160   else
161     speed--;
162 }
163 public void beAttacked(){
164   actualLife--;
165   if (actualLife == 0)
166     die();
167 }
168 public void flee(){
169   speed += 2;
170 }
171 public void die(){
172   videoJuego6.removeSoldier(this);
173   this.lives = false;
174 }
```

4.2. Clase VideoJuego6.java

4.2.1. Método para la ejecución principal del juego

- El método tiene como nombre main().
- Recibe como parámetros un arreglo de cadenas `args`, aunque en este caso no se utiliza.
- Crea una instancia de la clase `VideoJuego6`.
- Llama al método createArmy() para crear los ejércitos al inicio del juego.
- Llama al método mainInterfaz() para gestionar la interfaz principal del juego.

```
14 public static void main(String [] args){
15   VideoJuego6 videoJuego = new VideoJuego6();
16   videoJuego.createArmy();
17   videoJuego.mainInterfaz();
18 }
```

4.2.2. Método para interactuar con la Interfaz Principal

- El método tiene como nombre mainInterfaz().
- Se recibirá un número del 1 al 3 y se seleccionará la forma de juego o pedirá ingresar un número válido.

```
19 public void mainInterfaz(){
20   Scanner sc = new Scanner(System.in);
21   System.out.println( "1. Quick game"
22                   +"\n2. Custom game"
23                   +"\n3. Exit");
24   int action = sc.nextInt();
25   switch (action){
26     case 1 -> quickGame();
27     case 2 -> customGame();
28     case 3 -> System.out.println("Exiting the program.");
29     default -> {
30       System.out.println("Choose a valid option");
31       mainInterfaz();
32     }
33   }
34 }
```

4.2.3. Método para mostrar la interfaz de juego rápido

- El método tiene como nombre quickGame().
- Al iniciar esta pequeña interfaz, se muestra las acciones que se pueden realizar, las cuales se eligen escribiendo un número del 1 - 8 según se desee.
- Al terminar cada caso desde el 1 al 7, se vuelve a llamar al mismo método, esto lo vuelve iterativo.
- En el caso 1, se hace el llamado al método createArmy().
- En el caso 2, se hace el llamado al método showArmyData().
- En el caso 3, se hace el llamado al método showArmyTable().
- En el caso 4, se hace el llamado al método averageHealth().
- En el caso 5, se hace el llamado al método moreHealth().
- En el caso 6, se hace el llamado al método printArmyHealth.
- En el caso 7, se hace el llamado al método armyWinnerHealth().
- En el caso 8, se muestra el mensaje Fin ya que esta opción termina el programa.
- Y por último en caso de no ser ningún número anteriormente mencionado, se muestra un mensaje indicando que se debe seleccionar una opción válida, y se hace un llamado nuevamente a la función.

```
35  public void quickGame(){
36    Scanner sc = new Scanner(System.in);
37
38    System.out.println("1. Crear un nuevo ejercito"
39                    +"\n2. Mostrar los datos de los ejercitos"
40                    +"\n3. Mostrar la tabla con los ejercitos"
41                    +"\n4. Mostrar el promedio de vida de los ejercitos"
42                    +"\n5. Mostrar los soldados de ejercitos con mayor vida"
43                    +"\n6. Ordenar los soldados de ejercitos segun la vida"
44                    +"\n7. 1v1 battle"
45                    +"\n8. Salir del juego");
46    int action = sc.nextInt();
47
48    switch (action){
49      case 1 -> { // Crear un nuevo ejercito
50        army.clear();
51        army1DA.clear();
52        army1DB.clear();
53        createArmy();
54        quickGame();
55      }
56      case 2 -> { // Mostrar los datos de los ejercitos
57        System.out.println("DATOS DE LOS SOLDADOS CREADOS:\n");
58        showArmyData(army1DA, 'A');
59        showArmyData(army1DB, 'B');
60        quickGame();
61      }
62      case 3 -> { // Mostrar la tabla con los ejercitos
63        showArmyTable(army);
```

```
64        quickGame();
65      }
66      case 4 -> { // Mostrar el promedio de vida de los ejercitos
67        System.out.println("El promedio de vida del Ejercito A es: " +
              averageHealth(army1DA));
68        System.out.println("El promedio de vida del Ejercito B es: " +
              averageHealth(army1DB));
69        quickGame();
70      }
71      case 5 -> { // Mostrar los soldados de ejercitos con mayor vida
72        moreHelath(army1DA, 'A');
73        moreHelath(army1DB, 'B');
74        quickGame();
75      }
76      case 6 -> { // Ordenar los soldados de ejercitos segun la vida
77        System.out.println("\nEjercitos ordenados (insertionSort) segun la vida: ");
78        printArmyHealth(insertionSort(army1DA), 'A');
79        printArmyHealth(insertionSort(army1DB), 'B');
80        quickGame();
81      }
82      case 7 -> { // Jugar de 2
83        gameIntefaz();
84        quickGame();
85      }
86      case 8 -> { // Salir del juego
87        mainInterfaz();
88      }
89      default -> {
90        System.out.println("Selecciona una opcion valida");
91        quickGame();
92      }
93    }
94
95
96  }
```

4.2.4. Método para la creación de los ejército

- El método tiene como nombre createArmy().
- Primero se define el número de soldados que contendrá cada ejército, haciendo uso de Math.random.
- Ahora se llama dos veces al método createArmyTeam para realizar la creación de los dos ejércitos a partir de los tamaños ya establecidos anteriormente y además el argumento tipo char para el identificador de cada equipo.

```
98  public void createArmy(){
99    int numSoldiersA = (int) (Math.random() * 10) + 1;
100   int numSoldiersB = (int) (Math.random() * 10) + 1;
101
102   army1DA = createArmyTeam(numSoldiersA, army1DA, 'A');
103   army1DB = createArmyTeam(numSoldiersB, army1DB, 'B');
104 }
```

4.2.5. Método para la creación de ejércitos

- El método tiene como nombre createArmyTeam().
- Recibe como parámetros un entero que es el número de soldados del ejército, el HashMap de soldados a utilizar y un char que va al final del nombre de los soldados, esto último nos ayuda a identificarlos mejor.
- Utilizando un do while se crean posiciones aleatorias en el HashMap bidimensional, tomando como condición que dicha posición sea distinta de null.
- Finalmente se crea el soldado asignando su posición como clave Integer(el valor de la fila se guarda multiplicado por 10 sumándole el valor de la columna), el valor es el soldado y se almacena en cada HashMap y retorna el HashMap unidimensional con los soldados creados.

```
106 public HashMap <Integer, Soldado> createArmyTeam(int numSoldiers, HashMap <Integer,
        Soldado> army1D, char t){
107   for (int i = 0; i < numSoldiers; i++){
108     int row, col;
109
110     do {
111       row = (int) (Math.random() * 10);
112       col = (int) (Math.random() * 10);
113     } while (army.containsKey(row * 10 + col));
114
115     Soldado s = new Soldado(VideoJuego6.this,"Soldier" + i + "X" + t, row + 1, (char) (col
            + 'A'), t,
116                         (int)(Math.random() * 5) + 1, (int)(Math.random() * 5) + 1,
                                (int)(Math.random() * 5) + 1,
117                         0, "Defensiva", true);
118     army1D.put(i, s);
119     army.put(row * 10 + col, s);
120   }
121   return army1D;
122 }
```

4.2.6. Método para mostrar la tabla con el ejército

- El método tiene como nombre showArmyTable().
- El algoritmo utilizado para imprimir el tablero, ya fue explicado en los laboratorios anteriores, por ello se omite la explicación.

```
124 public void showArmyTable(HashMap <Integer, Soldado> army){
125   System.out.println("\n              TABLA CON LAS UBICACIONES DE LOS SOLDADOS CREADOS:
          \n");
126   String linesDown = "
          |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|";
127   System.out.println("    A       B       C       D       E       F       G       H       I
          J\n"
128                      +"
                             ------------------------------------------------------------------------------");
129
130   for (int r = 0; r < 10; r++){
131     System.out.print(" |");
```

```
132     for (int c = 0; c < 10; c++)
133       System.out.print(" " + (army.get(r*10+c) != null ? ("\'" + army.get(r*10+c).getTeam()
              + "\'"
134       + "S" + army.get(r*10+c).getName().charAt(7) + " |") : " |"));
135
136     System.out.print("\n" + (r+1) + ((r != 9) ? " |" : " |"));
137     for (int c = 0; c < 10; c++)
138       System.out.print(" " + (army.get(r*10+c) != null ? "HP: " +
              army.get(r*10+c).getActualLife() : " ") + " |");
139
140     System.out.println("\n" + linesDown );
141   }
142   System.out.println();
143 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
                    TABLA CON LAS UBICACIONES DE LOS SOLDADOS CREADOS:

      A       B       C       D       E       F       G       H       I       J
   _____________________________________________________________________________
   |       |       |       |       |       |       |       |       |       |       |
1  |       |       |       |       |       |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |'A'S3  |       |       |       |       |       |       |
2  |       |       |       |HP: 3  |       |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |'A'S0  |       |'B'S1  |       |'A'S1  |       |
3  |       |       |       |       |HP: 2  |       |HP: 1  |       |HP: 3  |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |       |       |'B'S0  |       |       |       |
4  |       |       |       |       |       |       |HP: 2  |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |       |       |       |       |       |       |
5  |       |       |       |       |       |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |       |       |       |       |'B'S4  |       |
6  |       |       |       |       |       |       |       |       |HP: 5  |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |'A'S2  |       |       |       |       |       |
7  |       |       |       |       |HP: 5  |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |'A'S4  |       |'B'S2  |       |       |       |
8  |       |       |       |       |HP: 4  |       |HP: 4  |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |       |       |       |       |       |       |
9  |       |       |       |       |       |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
   |       |       |       |       |'B'S3  |       |       |       |       |       |
10 |       |       |       |       |HP: 1  |       |       |       |       |       |
   |_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|
```

Figura 1

4.2.7. Método para mostrar los datos de los soldados de un ejército

- El método tiene como nombre showArmyData().
- Este usa un for each para recorrer el HashMap unidimensional de Soldado, y luego mostrar sus datos con el System.out.println, que a su vez este sigue el formato que se estableció en el método toString() de la clase Soldado.java.
- La impresión en consola será la misma que la figura 1.

```
145 public void showArmyData(HashMap <Integer, Soldado> army1D, char t){
146   System.out.println("EJERCITO \"" + t + "\"");
147   for (Soldado s : army1D.values())
148     System.out.println(s);
149 }
```

4.2.8. Método para mostrar aquellos soldados con más vida de un ejército

- El método tiene como nombre moreHealt().
- Este recibe el HashMap con los soldados de un ejército, además de un char que indica el nombre del equipo.
- Primero recorre el HashMap unidimensional de soldados haciendo uso de un bucle for, de esta manera obtendrá el máximo de vida del ejército, el cual será almacenado en un entero maxHealth.
- Finalmente imprimirá aquellos soldados que tengan la vida igual a maxHealth, con un for y un if que controlará aquello.
- En la impresión se incluye el nombre del ejército antes de mostrar a los soldados.

```
151 public void moreHelath(HashMap <Integer, Soldado> army1MH, char t){
152   int maxHealth = -1;
153   for(Soldado s : army1MH.values())
154     if (s.getActualLife() > maxHealth)
155       maxHealth = s.getActualLife();
156 
157   System.out.println("Soldado(s) con mayor vida del Ejercito " + t + ": ");
158   for (Soldado s : army1MH.values())
159     if (s.getActualLife() == maxHealth)
160       System.out.println("Nombre: " + s.getName() + " Vida: " + s.getActualLife());
161   System.out.println();
162 }
```

4.2.9. Método para hallar el promedio de vida en un ejército

- El método tiene como nombre averageHealth().
- De manera breve como el método, este retorna una división entre la suma de la vida del ejército, haciendo uso del método sumHealth() y dividiendo entre el tamaño del ejército.

```
164 public double averageHealth(HashMap <Integer, Soldado> army1DAH){
165   return sumHealth(army1DAH) / army1DAH.size();
166 }
```

4.2.10. Método para hallar la suma de vida en un ejército

- El método tiene como nombre sumHealth().
- Se utiliza un entero inicializado en 0 para mientras que se recorre el HashMap unidimensional de Soldado con un bucle for each, este entero (sum) va almacenando la vida de todos los soldados.
- Finalmente se retorna el entero sum.

```
168 public int sumHealth(HashMap <Integer, Soldado> army1DSH){
169   int sum = 0;
170   for (Soldado s : army1DSH.values())
171     sum += s.getActualLife();
172   return sum;
173 }
```

4.2.11. Método de ordenamiento InsertionSort

- El método tiene como nombre insertionSort().
- El método tiene como finalidad ordenar el HashMap unidimensional de Soldado haciendo uso del algoritmo InsertionSort, tomando en cuenta la vida de los soldados que contiene cada soldado de dicho HashMap. Este algoritmo se extrajo de Geeksforgeeks y fue adaptado a este proyecto.
- https://www.geeksforgeeks.org/insertion-sort/
- CONCLUSIONES:
  - El algoritmo recorre el HashMap desde el segundo soldado hasta el último, considerando cada vez al soldado como una "llavez compara su salud con la de los soldados anteriores.
  - Los soldaos son recorridos hacia la derecha en el HashMap hasta que se encuentren en la posición correcta según la vida del soldado "llave.ªctual.
  - El algoritmo de ordenamiento por inserción es más eficiente que el burbuja, pero a pesar de ello su complejidad sigue siendo de O(n^2).

```
174 public HashMap <Integer, Soldado> insertionSort(HashMap <Integer, Soldado> army1DIS) {
175   int n = army1DIS.size();
176   HashMap<Integer, Soldado> army1DCopyInsertion = new HashMap<>(army1DIS);
177 
178   for (int i = 1; i < n; i++) {
179     Soldado key = army1DCopyInsertion.get(i);
180     int j = i - 1;
181 
182     while (j >= 0 && army1DCopyInsertion.get(j).getActualLife() < key.getActualLife()) {
183       army1DCopyInsertion.put(j+1, army1DCopyInsertion.get(j));
184       j = j - 1;
185     }
186     army1DCopyInsertion.put(j+1, key);
187   }
188   return army1DCopyInsertion;
189 }
```

4.2.12. Método para imprimir los soldados de un ejército, ordenados según su vida

- El método tiene como nombre printArmyHealth().
- Este método recibirá un HashMap unidimensional de Soldado (previamente ordenado), lo recorrerá usando un bucle for y mostrará los soldados, teniendo en cuenta que primero se mostrarán los de mayor vida hasta los de menor.

```
190 public void printArmyHealth(HashMap <Integer, Soldado> armyPrint, char t){
191   System.out.println("EJERCITO " + t + " : ");
192   for(int i = 0; i < armyPrint.size(); i++)
193     System.out.println((i + 1) + ". " + armyPrint.get(i).getName() + " Vida: " +
          armyPrint.get(i).getActualLife());
194 }
```

4.2.13. Método para seleccionar el equipo a personalizar

- El método tiene como nombre customGame().
- Se muestra los equipos disponibles a modificar, luego se recibe un entero que será la elección del usuario.
- Usando condicionales se elige el método adecuado para la elección.
- Se valida la entrada y luego se hace llamado al método customGameArmy() enviando los respectivos argumentos.

```
196 public void customGame(){
197   Scanner sc = new Scanner(System.in);
198   System.out.println("Select army to customize:\n 1. A\n 2. B");
199   int actionTeam = sc.nextInt();
200   if (actionTeam == 1) // A
201     customGameArmy(army1DA, 'A');
202   else if(actionTeam == 2) // B
203     customGameArmy(army1DB, 'B');
204   else{
205     System.out.println("Invalid army, try again");
206     customGame();
207   }
208 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

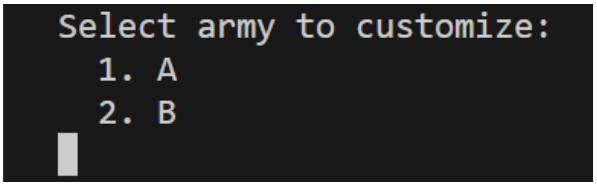

Figura 2

4.2.14. Método para mostrar la interfaz de juego personalizado

- El método tiene como nombre customGameArmy().
- Se muestra en pantalla las opciones y se recibe un entero valido que será la opción elegida por el usuario, para luego hacer un llamado al método correspondiente.

```
209 public void customGameArmy(HashMap <Integer, Soldado> armyA, char t){
210   Scanner sc = new Scanner(System.in);
211   System.out.println("Selected " + t + " army");
212   System.out.println( " 1. Create Soldier"
213                   +"\n 2. Delete Soldier"
214                   +"\n 3. Clone Soldier"
215                   +"\n 4. Modify Soldier"
216                   +"\n 5. Compare Soldiers"
217                   +"\n 6. Swap Soldiers"
218                   +"\n 7. View Soldier"
219                   +"\n 8. See Army"
220                   +"\n 9. Add Levels"
221                   +"\n10. Play"
222                   +"\n11. Return");
223   int action = sc.nextInt();
224   switch (action){
225     case 1 -> createSoldier(armyA, t);
226     case 2 -> deleteSoldier(armyA, t);
227     case 3 -> cloneSoldier(armyA, t);
228     case 4 -> modifySoldier(armyA, t);
229     case 5 -> compareSoldiers(armyA, t);
230     case 6 -> swapSoldiers(armyA, t);
231     case 7 -> viewSoldier(armyA, t);
232     case 8 -> seeArmy(armyA, t);
233     case 9 -> addLevels(armyA, t);
234     case 10 -> play();
235     case 11 -> mainInterfaz();
236     default -> {
237       System.out.println("Choose a valid option");
238       customGameArmy(armyA, t);
239     }
240   }
241   System.out.println("Returning to the menu");
242   customGameArmy(armyA, t);
243 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

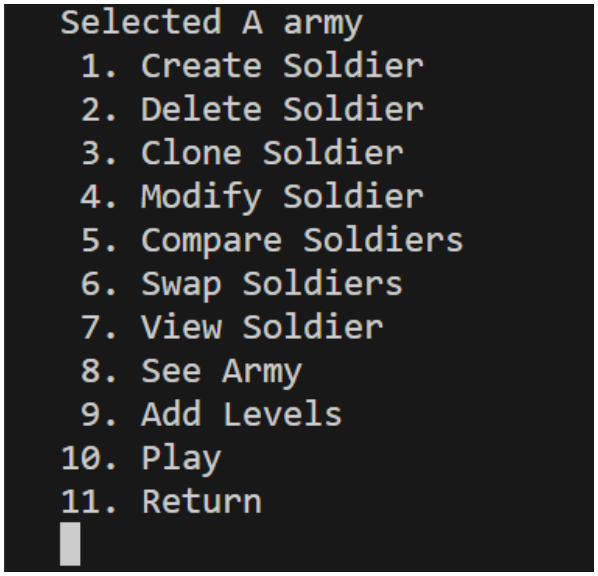

Figura 3

4.2.15. Método para asignar los cambios realizados al ejercito elegido

- El método tiene como nombre assignModification().
- Se recibe el ejército personalizado y el char que contiene al equipo.
- Haciendo uso de una condicional se asigna al HashMap de clase del ejército correcto.

```
244 public void assignModification(HashMap <Integer, Soldado> armyMod, char t){
245   if (t == 'A')
246     army1DA = armyMod;
247   else
248     army1DB = armyMod;
249 }
```

4.2.16. Método para crear un soldado personalizado

- El método tiene como nombre createSoldier().
- En este método se recibe los datos del soldado a crear si el tamaño del ejército es menor a 10.
- Hace uso del método createPosition() para los valores de la posición, en los demás casos los hace con el scanner y finalizada la recepción hace un llamado al segundo constructor.

```
250 public void createSoldier(HashMap <Integer, Soldado> armyMod, char t){
251   Scanner sc = new Scanner(System.in);
252   if (armyMod.size() < 10){
253     System.out.print("ENTER THE DATA: ");
254
```

```
255       System.out.print("Enter name: ");
256       String name = sc.next();
257       int position = createPosition(armyMod);
258       System.out.print("Enter attack level (1 - 5): ");
259       int attackLevel = sc.nextInt();
260       System.out.print("Enter level deffense (1 - 5): ");
261       int levelDefense = sc.nextInt();
262       System.out.print("Enter level life (1 - 5): ");
263       int levelLife = sc.nextInt();
264       System.out.print("Enter speed: ");
265       int speed = sc.nextInt();
266       Soldado s = new Soldado(VideoJuego6.this,name, position / 10 + 1, (char)(position % 10
              + 'A'), t, attackLevel, levelDefense, levelLife, speed);
267       armyMod.put(armyMod.size(), s);
268       army.put(position / 10 + position % 10, s);
269     }
270     else
271       System.out.println("The army reached the limit of allowed soldiers");
272     assignModification(armyMod, t);
273   }
```

- Un ejemplo de como se muestra en la siguiente imagen:

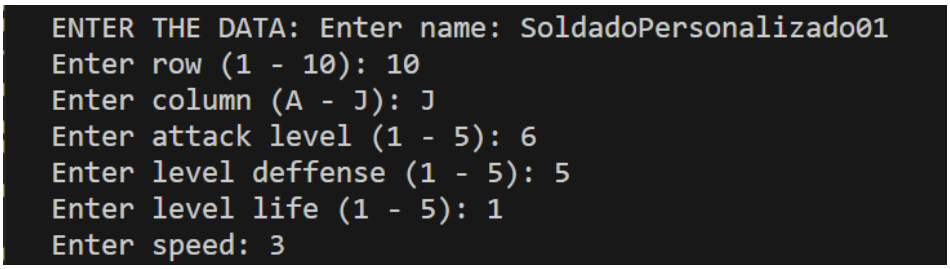

Figura 4

4.2.17. Método para crear una posición en el tablero

- El método tiene como nombre createPosition().
- El método recibe las coordenadas del soldado a crear y verifica que no esté ocupado, ya que en caso sea así este pedirá ingresar los datos nuevamente hasta que se ingrese un casillero vacío.

```
274   public int createPosition(HashMap <Integer, Soldado> armyMod){
275     Scanner sc = new Scanner(System.in);
276     int row;
277     char column;
278     do{
279       System.out.print("Enter row (1 - 10): ");
280       row = sc.nextInt();
281       System.out.print("Enter column (A - J): ");
282       column = sc.next().charAt(0);
283     } while(armyMod.get((row-1)*10 + (column - 'A')) != null);
284     return (row-1)*10 + (column - 'A');
285   }
```

4.2.18. Método para eliminar un soldado

- El método tiene como nombre deleteSoldier().
- Se muestra los soldados y luego se recibe el nombre del soldado a eliminar, se verifica que exista usando bucles y condicionales, en caso de que sí se eliminan del tablero principal y de l Hashmap de su equipo.

```
286 public void deleteSoldier(HashMap <Integer, Soldado> armyMod, char t){
287   Scanner sc = new Scanner(System.in);
288   if (armyMod.size() > 1){
289     System.out.println("ARMY \"" + t + "\"");
290     for (Soldado s : armyMod.values())
291       System.out.println(s.getName());
292 
293     System.out.println("Enter the name of the soldier to be eliminated");
294     String sName = sc.next();
295     Soldado soldierToDelete = null;
296     for (Soldado s : armyMod.values())
297       if (s.getName().equals(sName)){
298         soldierToDelete = s;
299         break;
300       }
301     if (soldierToDelete != null) {
302       armyMod.remove(Integer.parseInt(soldierToDelete.getName().substring(7,8)));
303       army.remove((soldierToDelete.getRow() - 1) * 10 + (soldierToDelete.getColumn() -
              'A'));
304       System.out.println("Soldier successfully eliminated: " + soldierToDelete.getName());
305     } else {
306       System.out.println("Soldier not found. Try again.");
307       deleteSoldier(armyMod, t);
308     }
309   } else {
310     System.out.println("No soldiers in the " + t + " army.");
311   }
312   assignModification(armyMod, t);
313 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
ARMY "A"
Soldier0XA
Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
SoldadoPersonalizado01
Enter the name of the soldier to be eliminated
123414515
Soldier not found. Try again.
```

Figura 5

4.2.19. Método para clonar un soldado

- El método tiene como nombre cloneSoldier().
- Sigue la misma lógica de los métodos anteriores, solo que en este caso pide una posición para colocar el soldado clonado ya que no pueden haber 2 en la misma posición del tablero.
- En cuanto a lo demás realiza una copia de los atributos menos el del nombre ya que si se desearía utilizar en otro método no habría manera de saber a cual llamar.

```
314  public void cloneSoldier(HashMap<Integer, Soldado> armyMod, char t) {
315    Scanner sc = new Scanner(System.in);
316    if (armyMod.size() < 10) {
317      for (Soldado s : armyMod.values())
318        System.out.println(s.getName());
319
320      System.out.println("Enter the name of the soldier to clone:");
321      String originalSoldierName = sc.next();
322
323      Soldado originalSoldado = null;
324      for (Soldado s : armyMod.values())
325        if (s.getName().equals(originalSoldierName)) {
326          originalSoldado = s;
327          break;
328        }
329      if (originalSoldado != null){
330        int row, col;
331        do {
332          System.out.print("Enter the row (1 - 10) for the cloned soldier: ");
333          row = sc.nextInt();
334          System.out.print("Enter the column (A - J) for the cloned soldier: ");
335          col = Character.toUpperCase(sc.next().charAt(0)) - 'A';
336        } while (armyMod.get(row * 10 + col) != null);
337
338        Soldado clonedSoldado = new Soldado(originalSoldado.getVideoJuego6(),"Soldier" +
               armyMod.size() + "X" + t,
339                    row + 1, (char) (col + 'A'), t, originalSoldado.getAttackLevel(),
                         originalSoldado.getLevelDefense(),
340                    originalSoldado.getLevelLife(),
                         originalSoldado.getSpeed(),"Defensiva",true);
341        armyMod.put(armyMod.size(), clonedSoldado);
342        army.put(row * 10 + col, clonedSoldado);
343
344        System.out.println("Soldier cloned successfully: " + clonedSoldado.getName());
345      }
346      else {
347        System.out.println("Soldier not found. Try again.");
348        cloneSoldier(armyMod, t);
349      }
350    } else
351        System.out.println("The army reached the limit of allowed soldiers.");
352    assignModification(armyMod, t);
353  }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
SoldadoPersonalizado01
Enter the name of the soldier to clone:
SoldadoPersonalizado01
Enter the row (1 - 10) for the cloned soldier: 1
Enter the column (A - J) for the cloned soldier: H
Soldier cloned successfully: Soldier9XA
```

Figura 6

4.2.20. Método para modificar un soldado

- El método tiene como nombre modifySoldier().
- Primeramente se muestra el ejército y se solicita ingresar el nombre del soldado a modificar.
- Se verifica que exista y luego de ello se muestra las opciones a modificar, se recibe la elección y se usa un switch para cada opción.
- Por último solo se le asigna el nuevo valor recibido al soldado.

```
354  public void modifySoldier(HashMap<Integer, Soldado> armyMod, char t) {
355    Scanner sc = new Scanner(System.in);
356    System.out.println("ARMY \"" + t + "\"");
357    for (Soldado s : armyMod.values())
358      System.out.println(s.getName());
359
360    System.out.println("Enter the name of the soldier to modify:");
361    String soldierName = sc.next();
362
363    Soldado soldierToModify = null;
364    for (Soldado s : armyMod.values())
365      if (s.getName().equals(soldierName)) {
366        soldierToModify = s;
367        break;
368      }
369
370    if (soldierToModify != null) {
371      System.out.println("Modify Soldier - " + soldierToModify.getName());
372      System.out.println("1. Modify Attack Level");
373      System.out.println("2. Modify Defense Level");
374      System.out.println("3. Modify Life Level");
375      System.out.println("4. Return");
376
377      int choice = sc.nextInt();
378
379      switch (choice) {
380        case 1 -> {
381          System.out.print("Enter new Attack Level (1 - 5): ");
382          int newAttackLevel = sc.nextInt();
383          soldierToModify.setAttackLevel(newAttackLevel);
```

```
384         }
385         case 2 -> {
386           System.out.print("Enter new Defense Level (1 - 5): ");
387           int newDefenseLevel = sc.nextInt();
388           soldierToModify.setLevelDefense(newDefenseLevel);
389         }
390         case 3 -> {
391           System.out.print("Enter new Life Level (1 - 5): ");
392           int newLifeLevel = sc.nextInt();
393           soldierToModify.setLevelLife(newLifeLevel);
394         }
395         case 4 -> System.out.println("Returning to the menu");
396         default -> {
397           System.out.println("Invalid choice. Returning to the menu.");
398         }
399       }
400     } else {
401       System.out.println("Soldier not found. Try again.");
402       modifySoldier(armyMod, t);
403     }
404   }
```

- Un ejemplo de como se muestra en la siguiente imagen:

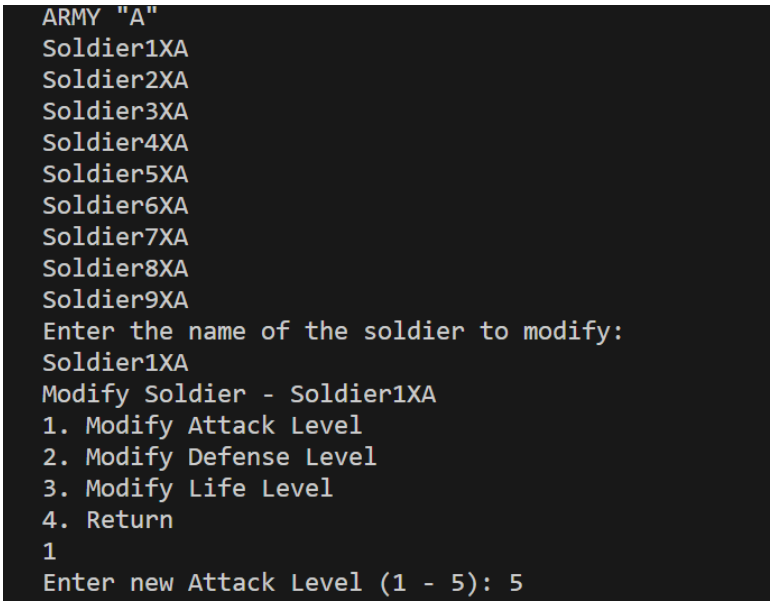

Figura 7

4.2.21. Método para comparar soldados

- El método tiene como nombre compareSoldiers().
- Pide los nombres de los dos soldados a comparar, para luego verificar que exista y posteriormente hacer uso del método compareSoldadoAttributes() y con uso de condicionales mostrar si son idénticos o no.

```
405 public void compareSoldiers(HashMap<Integer, Soldado> armyMod, char t){
406   Scanner sc = new Scanner(System.in);
407   for (Soldado s : armyMod.values())
408     System.out.println(s.getName());
409
410   System.out.println("Enter the name of the first soldier:");
411   String soldierName1 = sc.next();
412   System.out.println("Enter the name of the second soldier:");
413   String soldierName2 = sc.next();
414
415   Soldado soldier1 = findSoldado(armyMod, soldierName1);
416   Soldado soldier2 = findSoldado(armyMod, soldierName2);
417
418   if (soldier1 != null && soldier2 != null){
419     if (compareSoldadoAttributes(soldier1, soldier2))
420       System.out.println("Soldiers are identical.");
421     else
422       System.out.println("Soldiers are different.");
423   }
424   else{
425     System.out.println("Soldier not found. Try again.");
426     compareSoldiers(armyMod, t);
427   }
428 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

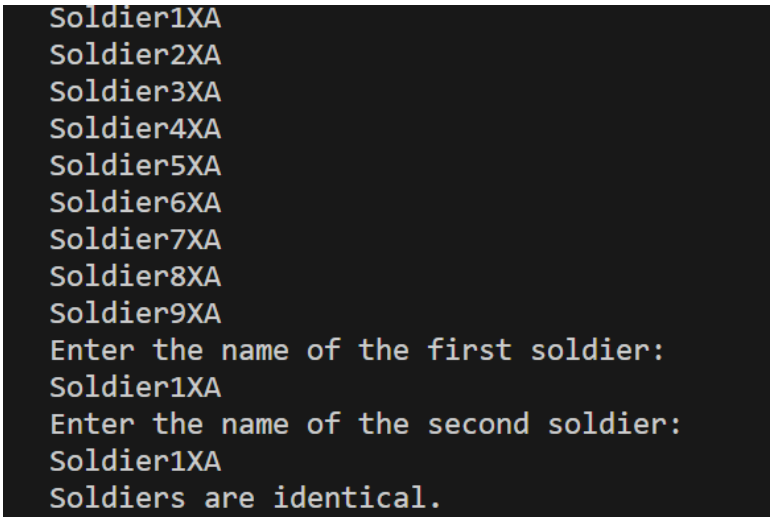

Figura 8

4.2.22. Método para buscar un soldado

- El método tiene como nombre findSoldado().
- Se envia el nombre del soldado como atributo junto a la estructura de datos que contiene su ejército, se usa un bucle for each para recorrer y retornar el soldado en caso exista.

```
429 public Soldado findSoldado(HashMap<Integer, Soldado> armyMod, String soldierName) {
430   for (Soldado s : armyMod.values())
431     if (s.getName().equals(soldierName))
432       return s;
433   return null;
434 }
```

4.2.23. Método para comparar los atributos de un soldado

- El método tiene como nombre compareSoldadoAttributes().
- El método obtiene los atributos de ambos soldados y los compara dentro del mismo return.

```
435 public boolean compareSoldadoAttributes(Soldado s1, Soldado s2) {
436   return s1.getName().equals(s2.getName()) &&
437         s1.getAttackLevel() == s2.getAttackLevel() &&
438         s1.getLevelDefense() == s2.getLevelDefense() &&
439         s1.getActualLife() == s2.getActualLife() &&
440         s1.getLives() == s2.getLives();
441 }
```

4.2.24. Método para intercambiar posiciones de 2 soldados

- El método tiene como nombre swapSoldiers().
- Muestra los soldados y luego recibe los nombres de los soldados a intercambiar verificando que existan.
- Posteriormente realiza las modificaciones tanto de sus atributos como en el tablero principal.

```
442 public void swapSoldiers(HashMap<Integer, Soldado> armyMod, char t) {
443   Scanner sc = new Scanner(System.in);
444
445   System.out.println("ARMY \"" + t + "\"");
446   for (Soldado soldado : armyMod.values())
447     System.out.println(soldado.getName());
448
449   System.out.println("Enter the name of the first soldier to swap:");
450   String sName1 = sc.next();
451   Soldado soldier1 = findSoldado(armyMod, sName1);
452
453   System.out.println("Enter the name of the second soldier to swap:");
454   String sName2 = sc.next();
455   Soldado soldier2 = findSoldado(armyMod, sName2);
456
457   if (soldier1 != null && soldier2 != null) {
```

```
458     System.out.println("Swapping Soldiers - " + soldier1.getName() + " and " +
            soldier2.getName());
459     int position1 = (soldier1.getRow() - 1) * 10 + (soldier1.getColumn() - 'A');
460     int position2 = (soldier2.getRow() - 1) * 10 + (soldier2.getColumn() - 'A');
461
462     armyMod.remove(position1);
463     armyMod.remove(position2);
464
465     armyMod.put(position1, soldier2);
466     armyMod.put(position2, soldier1);
467
468     System.out.println("Soldiers swapped successfully.");
469   } else {
470     System.out.println("One or both soldiers not found. Try again.");
471     swapSoldiers(armyMod, t);
472   }
473 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

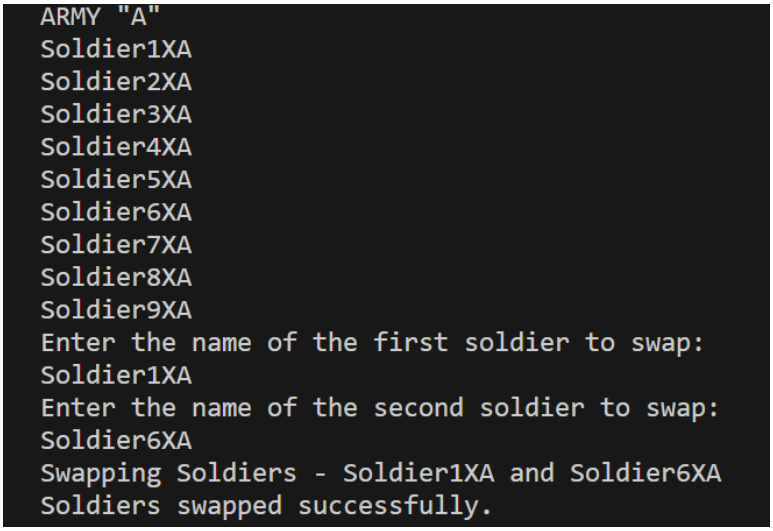

Figura 9

Listing 5: Commit c2bd05c: Septimo avance del metodo para jugabilidad de 2 jugadores, concluido hasta el lab11

```
$ git add .
$ git commit -m "Septimo avance del metodo para jugabilidad de 2 jugadores, concluido hasta
    el lab11"
$ git push -u origin main
```

4.2.25. Método para ver los datos de un soldado

- El método tiene como nombre viewSoldier().
- Muestra los soldados y solicita el nombre del soldado que se desea mostrar datos. Luego se busca el soldado y finalmente se muestra sus atributos.

```
474 public void viewSoldier(HashMap<Integer, Soldado> armyMod, char t) {
475   Scanner sc = new Scanner(System.in);
476   for (Soldado soldado : armyMod.values())
477     System.out.println(soldado.getName());
478   System.out.println("Enter the name of the soldier to view:");
479   String soldierName = sc.next();
480
481   Soldado soldierToView = findSoldado(armyMod, soldierName);
482
483   if (soldierToView != null) {
484     System.out.println("Soldier Details:");
485     System.out.println("Name: " + soldierToView.getName());
486     System.out.println("Team: " + soldierToView.getTeam());
487     System.out.println("Position: " + soldierToView.getRow() + " " +
          soldierToView.getColumn());
488     System.out.println("Attack Level: " + soldierToView.getAttackLevel());
489     System.out.println("Defense Level: " + soldierToView.getLevelDefense());
490     System.out.println("Life Level: " + soldierToView.getLevelLife());
491     System.out.println("Speed: " + soldierToView.getSpeed());
492   } else {
493     System.out.println("Soldier not found. Try again.");
494     viewSoldier(armyMod, t);
495   }
496 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
Soldier9XA
Soldier1XA
Soldier6XA
Enter the name of the soldier to view:
Soldier6XA
Soldier Details:
Name: Soldier6XA
Team: A
Position: 5 G
Attack Level: 1
Defense Level: 1
Life Level: 5
Speed: 0
```

Figura 10

4.2.26. Método para ver un ejército

- El método tiene como nombre seeArmy.
- Sigue la misma lógica que el método anterior para ver soldados, de hecho podría reutilizarse para este método.

```
497 public void seeArmy(HashMap<Integer, Soldado> armyMod, char t) {
498   System.out.println("EJERCITO \"" + t + "\"");
499   for (Soldado s : armyMod.values()) {
500     System.out.println("Soldier Details:");
501     System.out.println("Name: " + s.getName());
502     System.out.println("Team: " + s.getTeam());
503     System.out.println("Position: " + s.getRow() + " " + s.getColumn());
504     System.out.println("Attack Level: " + s.getAttackLevel());
505     System.out.println("Defense Level: " + s.getLevelDefense());
506     System.out.println("Life Level: " + s.getLevelLife());
507     System.out.println("Speed: " + s.getSpeed());
508     System.out.println();
509   }
510   customGameArmy(armyMod, t);
511 }
```

4.2.27. Método para ver la sumatoria de niveles de un ejército

- El método tiene como nombre addLevels().
- El método usa un bucle for each para recorrer el ejército, luego va aumentando los valores de cada nivel de soldado para luego contenerlo en una variable entera y ser mostrada al final.

```
512 public void addLevels(HashMap<Integer, Soldado> armyMod, char t) {
513   int totalAttackLevel = 0;
514   int totalDefenseLevel = 0;
515   int totalLifeLevel = 0;
516   int totalSpeed = 0;
517 
518   for (Soldado soldier : armyMod.values()) {
519       totalAttackLevel += soldier.getAttackLevel();
520       totalDefenseLevel += soldier.getLevelDefense();
521       totalLifeLevel += soldier.getLevelLife();
522       totalSpeed += soldier.getSpeed();
523   }
524   System.out.println("Sumatoria de niveles del Ejercito " + t + ":");
525   System.out.println("Sumatoria de Nivel de Ataque: " + totalAttackLevel);
526   System.out.println("Sumatoria de Nivel de Defensa: " + totalDefenseLevel);
527   System.out.println("Sumatoria de Nivel de Vida: " + totalLifeLevel);
528   System.out.println("Sumatoria de Velocidad: " + totalSpeed);
529 }
```

- Un ejemplo de como se muestra en la siguiente imagen:

```
Sumatoria de niveles del Ejercito A:
Sumatoria de Nivel de Ataque: 41
Sumatoria de Nivel de Defensa: 31
Sumatoria de Nivel de Vida: 38
Sumatoria de Velocidad: 3
Returning to the menu
```

Figura 11

4.2.28. Método para jugar ejército contra ejército

- El método tiene como nombre play().
- En este caso solo hace llamada al método gameInterfaz(). que es la interfaz del juego 1v1.

```
530 public void play(){
531   gameIntefaz();
532 }
```

4.2.29. Método para mostrar al ejército ganador

- El método tiene como nombre amryWinner().
- Este método imprimirá al ejército ganador, usando como condición su tamaño.
- El método sólo será llamado cuando se verifique que uno de los dos ejércitos está vacío.

```
535 public void armyWinner(){
536   if(army1DA.size() > army1DB.size())
537     System.out.print("A");
538   else
539     System.out.print("B");
540 }
```

4.2.30. Método para remover un soldado

- El método tiene como nombre removeSoldier().
- Obtiene el ejército al que pertenece y lo elimina tanto del HashMap de su ejército y del tablero de juego.

```
542 public void removeSoldier(Soldado s){
543   if (s.getTeam() == 'A')
544     army1DA.remove(s.getName().charAt(7) - '0');
545   else
546     army1DB.remove(s.getName().charAt(7) - '0');
547   army.remove((s.getRow()-1)*10 + (s.getColumn() -'A'));
548 }
```

4.2.31. Método para ejecutar la interfaz del juego

- El método tiene como nombre gameInterfaz().
- Se controla que el tamaño de los ejércitos no sea 0, de esta manera cuando lo sea se rompera el bucle while y se dará el nombre del ganador usando el método amryWinner().

```
549 public void gameIntefaz(){
550   Scanner sc = new Scanner(System.in);
551   System.out.println("Starting Game...");
552   boolean noEnd = true;
553   while (noEnd) {
554     System.out.println("Team A Turn");
555     turn('A');
556     if(army1DA.size() == 0)
557       break;
558     System.out.println("Team B Turn");
559     turn('B');
560     if(army1DB.size() == 0)
561       break;
562   }
563
564   System.out.println("Winning team: ");
565   armyWinner();
566 }
```

4.2.32. Método para la ejecución del turno del jugador

- El método tiene como nombre turn().
- Primero se comienza mostrando el tablero de juego. Luego se recibe las coordenadas del soldado a mover, para verificar se hace uso del método checkSoldier1() y así llamar a turn2().

```
567 public void turn(char teamT){
568   showArmyTable(army);
569   Scanner sc = new Scanner(System.in);
570   System.out.println("Indicate the coordinates of the soldier to move. Ej: 1 A (put the
          space in the middle)");
571   int row1 = sc.nextInt() - 1;
572   char l1 = sc.next().charAt(0);
573   int col1 = Character.toUpperCase(l1) - 'A';
574   if (checkSoldier1(row1, col1, teamT))
575     turn2(row1, col1, teamT);
576   else
577     turn(teamT);
578
579 }
```

Listing 6: Commit d8d54ce: Se agregó los métodos para la jugabilidad de 2 (hasta lab10)

```
$ git add .
$ git commit -m "Sexto avance del metodo para jugabilidad de 2 jugadores, concluido hasta
    el lab10"
$ git push -u origin main
```

4.2.33. Método para revisar que el soldado elegido sea válido

- El método tiene como nombre checkSoldier1().
- Este fue utilizado en el método anterior, verifica que las coordenadas estén dentro del tablero y pertenezca al equipo del cual es turno.

```
580 public boolean checkSoldier1(int row, int col, char team){
581   if(row > 9 || col > 9)
582     return false;
583   if (army.get(row*10 + col) != null){
584     if(army.get(row*10 + col).getTeam() == team)
585       return true;
586     else{
587       System.out.println("Choose a soldier from your team");
588       return false;
589     }
590   }
591   return false;
592 }
```

4.2.34. Método para analizar la posición a mover

- El método tiene como nombre checkSoldier2().
- Esta es la segunda verificación ya que evalúa si se va a producir un movimiento a un casillero libre, hay un enemigo o aliado. Luego de eso regresa el entero según sea el caso.

```
593 public int checkSoldier2(int row, int col, char team){
594   if(row > 9 || col > 9)
595     return 4;
596   if (army.get(row*10 + col) == null)
597     return 1;
598   else if (army.get(row*10 + col).getTeam() != team)
599     return 3;
600   return 2;
601 }
```

4.2.35. Método para ejecutar el movimiento elegido

- El método tiene como nombre turn2().
- Esté método recibe la coordenada a donde se desea mover el soldado que fue elegido previamente.
- Además de contener los posibles movimientos en caso sea válido, por ejemplo una pelea de soldados.

```
602 public void turn2(int row1, int col1, char teamT){
603   Scanner sc = new Scanner(System.in);
604   System.out.println("Indicate the coordinates where you want to move the soldier. Ej: 1 A
          (put the space in the middle)");
605   int row2 = sc.nextInt() - 1;
606   char l2 = sc.next().charAt(0);
```

```
607     int col2 = Character.toUpperCase(l2) - 'A';
608
609     switch(checkSoldier2(row2, col2, teamT)){
610       case 1 -> moveSoldier(row1, col1, row2, col2);
611       case 2 -> {
612         System.out.println("There cannot be two soldiers in the same position, try again");
613         turn2(row1, col1, teamT);
614       }
615       case 3 ->{
616         System.out.println("--SOLDIERS FIGHT--");
617         System.out.println(army.get(row1*10 + col1).getName() + " vs " + army.get(row2*10 +
                col2).getName());
618         Soldado sW = soldiersFight(army.get(row1*10 + col1), army.get(row2*10 + col2), row2,
                col2, row1, col1);
619       }
620       case 4 ->{
621         System.out.println("Invalid position, try again");
622         turn2(row1, col1, teamT);
623       }
624     }
625
626   }
```

4.2.36. Método para mover un soldado 1

- El método tiene como nombre moveSoldier() y es sobrecargado.
- El método coloca el soldado en la posición previamente recibida y la remueve de su posición anterior.

```
627   public void moveSoldier(int row1, int col1, int row2, int col2){
628     army.put(row2*10 + col2, army.get(row1*10 + col1));
629     army.remove(row1*10 + col1);
630   }
```

4.2.37. Método para mover un soldado 2

- El método tiene como nombre moveSoldier() y es sobrecargado.
- Al contrario del método anterior este cuenta con parámetros distintos ya que incluye un soldado, pero realiza la misma función que el método anterior, pero este en caso de haber un soldado ganador de una pelea.

```
631   public void moveSoldier(Soldado s,int row1, int col1, int rowD, int colD){
632     army.remove(rowD*10 + colD);
633     army.put(row1*10 + col1, s);
634   }
```

4.2.38. Método para ejecutar la pelea de dos soldados

- El método tiene como nombre soldiersFight().

- Este método es uno de los más importantes ya que ejecuta la pelea entre dos soldados.
- Crea las probabilidades proporcionalmente a la suma de vida actual de ambos soldados.
- Muestra las probabilidades y luego elije de manera aleatoria con uso de Random de java.util, para finalmente mostrar al soldado ganador y llamar a los métodos necesarios que se encarguen de eliminar y mover los soldados según sea la situación.

```
635 public Soldado soldiersFight(Soldado a, Soldado b, int rowP, int colP, int rowD, int colD){
636   double probabilityA = (double) a.getActualLife() / (a.getActualLife() +
          b.getActualLife()) * 100;
637   double probabilityB = (double) b.getActualLife() / (a.getActualLife() +
          b.getActualLife()) * 100;
638
639   System.out.println("Probability of " + a.getName() + ": " + probabilityA + "%");
640   System.out.println("Probability of " + b.getName() + ": " + probabilityB + "%");
641
642   Random random = new Random();
643   double probabilityT = probabilityA + probabilityB;
644   double randomValue = random.nextDouble() * probabilityT;
645
646   Soldado winner;
647   if (randomValue < probabilityA) {
648       System.out.println("The winner is: " + a.getName());
649       b.die();
650       winner = a;
651   } else {
652       System.out.println("The winner is: " + b.getName());
653       a.die();
654       winner = b;
655   }
656   winner.setActualLife(winner.getActualLife() + 1);
657   moveSoldier(winner, rowP, colP, rowD, colD);
658   return winner;
659 }
```

Listing 7: Commit b3523d0: Siendo este el último commit referente al trabajo del código (el último fue revisión)

```
$ git add .
$ git commit -m "Decimo avance del menu personalizado, se agrego el metodo para salir"
$ git push -u origin main
```

4.3. Diagrama de clase UML

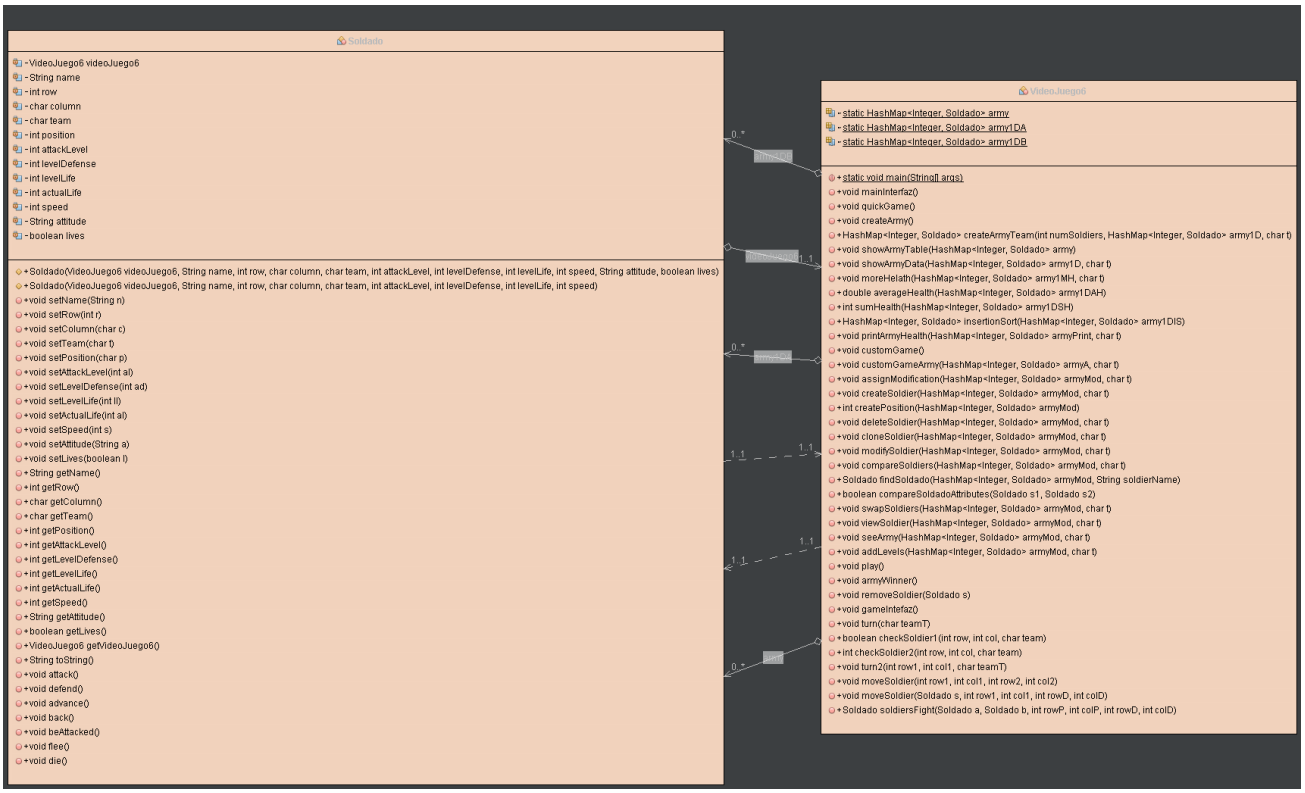

Figura 12

Diagrama 12 UML para acceder al diagrama UML del repositorio y se observe con más claridad.

4.4. Estructura de laboratorio 12

- El contenido que se entrega en este laboratorio es el siguiente:

```
lab12
|   Soldado.java
|   VideoJuego6.java
|
|------latex
    |   Informe_Lab12.pdf
    |   Informe_Lab12.tex
    |
    |------img
    |       12addLevels.png
    |       12compareSoldiers.png
    |       12customGame.png
    |       12deleteSoldier.png
    |       12swapSoldiers.png
    |       12viewSoldier.png
    |       logo_episunsa.png
    |       mainInterfaz.png
    |       12cloneSoldier.png
    |       12createSoldier.png
    |       12customGameArmy.png
    |       12modifySoldier.png
    |       12uml.png
    |       logo_abet.png
    |       logo_unsa.jpg
    |        showArmyTable.png
    |
    |------src
            Soldado.java
            VideoJuego6.java
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y fácil de leer.

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 cumplió 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 lo ítems.
- El alumno debe auto calificarse 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
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	2	
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 están dentro de los plazos de fecha de entrega establecidos.	2	X	2	
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		19	

6. Referencias

- https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
- https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html
- https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html
- https://www.geeksforgeeks.org/insertion-sort/
- https://es.stackoverflow.com/questions/108171/