

Informe de Laboratorio 20

Tema: Definición de Clases de Usuario, Herencia y Polimorfismo.

Nota

Estudiante	Escuela	Asignatura
Hernan Andy Choquehuanca Zapana hchoquehuanca@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 1701213

Laboratorio	Tema	Duración
20	Definición de Clases de Usuario, Herencia y Polimorfismo.	10 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 08 Enero 2024	Al 10 Enero 2024

1. Tarea

- Crear diagrama de clases UML y programa.
- Crear los miembros de cada clase de la forma más adecuada: como miembros de clase o de instancia.
- Crear la clase Mapa, que esté constituida por el tablero antes visto, que posicione soldados en ciertas posiciones aleatorias (entre 1 y 10 soldados por cada ejército, sólo 1 ejército por reino). Se deben generar ejércitos de 2 reinos. No se admite guerra civil. El Mapa tiene como atributo el tipo de territorio que es (bosque, campo abierto, montaña, desierto, playa). La cantidad de soldados, así como todos sus atributos se deben generar aleatoriamente.
- Dibujar el Mapa con las restricciones que sólo 1 soldado como máximo en cada cuadrado.
- El mapa tiene un solo tipo de territorio.
- Considerar que el territorio influye en los resultados de las batallas, así cada reino tiene bonus según el territorio: Inglaterra-¿bosque, Francia-¿campo abierto, Castilla-Aragón-¿montaña, Moros-¿desierto, Sacro Imperio Romano-Germánico-¿bosque, playa, campo abierto. En dichos casos, se aumenta el nivel de vida en 1 a todos los soldados del reino beneficiado.
- En la historia, los ejércitos estaban conformados por diferentes tipos de soldados, que tenían similitudes, pero también particularidades.

- Basándose en la clase Soldado crear las clases Espadachín, Arquero, Caballero y Lancero. Las cuatro clases heredan de la superclase Soldado pero aumentan atributos y métodos, o sobrescriben métodos heredados.
- Los espadachines tienen como atributo particular "longitud de espada" como acción "crear un muro de escudos" que es un tipo de defensa en particular.
- Los caballeros pueden alternar sus armas entre espada y lanza, además de desmontar (sólo se realiza cuando está montando e implica defender y cambiar de arma a espada), montar (sólo se realiza cuando está desmontado e implica montar, cambiar de arma a lanza y vestir). El caballero también puede vestir, ya sea montando o desmontando, cuando es desmontado equivale a atacar 2 veces pero cuando está montando implica a atacar 3 veces.
- Los arqueros tienen un número de flechas disponibles las cuales pueden dispararse y se gastan cuando se hace eso.
- Los lanceros tienen como atributo particular, "longitud de lanza" como acción "schiltrom" (como una falange que es un tipo de defensa en particular y que aumenta su nivel de defensa en 1).
- Tendrá 2 Ejércitos que pueden ser constituidos sólo por espadachines, caballeros, arqueros y lanceros. No se acepta guerra civil. Crear una estructura de datos conveniente para el tablero. Los soldados del primer ejército se almacenarán en un arreglo estándar y los soldados del segundo ejército se almacenarán en un ArrayList. Cada soldado tendrá un nombre autogenerado: Espadachin0X1, Arquero1X1, Caballero2X2, etc., un valor de nivel de vida autogenerado aleatoriamente, la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado) y valores autogenerados para el resto de atributos.
- Todos los caballeros tendrán los siguientes valores: ataque 13, defensa 7, nivel de vida [10..12] (el nivel de vida actual empieza con el valor del nivel de vida).
- Todos los arqueros tendrán los siguientes valores: ataque 7, defensa 3, nivel de vida [3..5] (el nivel de vida actual empieza con el valor del nivel de vida).
- Todos los espadachines tendrán los siguientes valores: ataque 10, defensa 8, nivel de vida [8..10] (el nivel de vida actual empieza con el valor del nivel de vida).
- Todos los lanceros tendrán los siguientes valores: ataque 5, defensa 10, nivel de vida [5..8] (el nivel de vida actual empieza con el valor del nivel de vida).
- Mostrar el tablero, distinguiendo los ejércitos y los tipos de soldados creados. Además, se debe mostrar todos los datos de todos los soldados creados para ambos ejércitos. Además de los datos del soldado con mayor vida de cada ejército, el promedio de nivel de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando algún algoritmo de ordenamiento.
- Finalmente, que muestre el resumen los 2 ejércitos, indicando el reino, cantidad de unidades, distribución del ejército según las unidades, nivel de vida total del ejército y qué ejército ganó la batalla (usar la métrica de suma de niveles de vida y porcentajes de probabilidad de victoria basado en ella). Este porcentaje también debe mostrarse.
- Hacerlo programa iterativo.

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.
- Herencia.
- Polimorfismo.
- Miembros de clase.
- Clases de Usuario.

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/fase03/lab20>

4. Trabajo del Laboratorio 20

4.1. Clase Soldado.java

Listing 1: Commit 10a3bb4: Se reutilizo las clases del laboratorio anterior

```
$ git add .  
$ git commit -m "Reutilizando las clases utilizadas en el ultimo laboratorio (12)"  
$ 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 VideoJuego7 VideoJuego7`: Una referencia al objeto de la clase VideoJuego7.
 - `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.
- `private char type`: Indica que tipo de soldado es.
- `private String reino`: Indica a que tipo de reino pertenece.

```
2 private VideoJuego7 VideoJuego7;  
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;  
16 private char type;  
17 private String reino;
```

Listing 2: Commit bb6e2c8: Agregando los atributos solicitados a la clase Soldado y a las clases derivadas de la misma

```
$ git add .  
$ git commit -m "Creando nuevas clases de Soldados vacias e implementando la clase  
Mapa.java"  
$ git push -u origin main
```

- Se han implementado tres constructores para la clase Soldado:
 - El primer constructor recibe todos los atributos como parámetros posibles y necesarios.

```
19 public Soldado(VideoJuego7 VideoJuego7,String name, int row, char column, char team,  
20               int attackLevel, int levelDefense, int levelLife, int speed,  
21               String attitude, boolean lives){  
22     this.name = name;  
23     this.row = row;  
24     this.column = column;  
25     this.team = team;  
26     this.attackLevel = attackLevel;
```

```
27     this.levelDefense = levelDefense;
28     this.levelLife = levelLife;
29     this.actualLife = levelLife;
30     this.speed = speed;
31     this.attitude = attitude;
32     this.lives = lives;
33     this.VideoJuego7 = VideoJuego7;
34 }
```

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

```
36 public Soldado(VideoJuego7 VideoJuego7,String name, int row, char column, char team,
37               int attackLevel, int levelDefense, int levelLife, int speed){
38     this.name = name;
39     this.row = row;
40     this.column = column;
41     this.team = team;
42     this.attackLevel = attackLevel;
43     this.levelDefense = levelDefense;
44     this.levelLife = levelLife;
45     this.actualLife = levelLife;
46     this.speed = speed;
47     this.lives = true;
48     this.VideoJuego7 = VideoJuego7;
49 }
```

- El tercer constructor es una sobrecarga que es utilizada cuando se crean los tipos de soldado, ya que no recibe ciertos datos ya definidos según el tipo.

```
50 public Soldado(VideoJuego7 VideoJuego7,String name, int row, char column, char team,
51               int speed, String attitude, boolean lives, String reino){
52     this.name = name;
53     this.row = row;
54     this.column = column;
55     this.team = team;
56     this.actualLife = levelLife;
57     this.speed = speed;
58     this.lives = true;
59     this.VideoJuego7 = VideoJuego7;
60     this.reino = reino;
61 }
```

Listing 3: Commit bb6e2c8: Se agregó el constructor para recibir datos de los tipos de soldado, excepto los ya definidos

```
$ git add .
$ git commit -m "Cuarta version de las clases de tipos de Soldado, ademas de modificar la
    clase principal (VideoJuego7.java)"
$ git push -u origin main
```

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

- Setters:

```
64 public void setName(String n){
65     name = n;
66 }
67 public void setRow(int r){
68     row = r;
69 }
70 public void setColumn(char c){
71     column = c;
72 }
73 public void setTeam(char t){
74     team = t;
75 }
76 public void setPosition(char p){
77     position = p;
78 }
79
80 public void setAttackLevel(int al){
81     attackLevel = al;
82 }
83 public void setLevelDefense(int ad){
84     levelDefense = ad;
85 }
86 public void setLevelLife(int ll){
87     levelLife = ll;
88 }
89 public void setActualLife(int al){
90     actualLife = al;
91 }
92 public void setSpeed(int s){
93     speed = s;
94 }
95 public void setAttitude(String a){
96     attitude = a;
97 }
98 public void setLives(boolean l){
99     lives = l;
100 }
101 public void setType(char ty){
102     type = ty;
103 }
104 public void setReino(char re){
105     type = re;
106 }
```

- Getters:

```
109 public String getName(){
110     return name;
111 }
112 public int getRow(){
113     return row;
114 }
115 public char getColumn(){
116     return column;
117 }
118 public char getTeam(){
119     return team;
120 }
121 public int getPosition(){
122     return position;
123 }
124 public int getAttackLevel(){
125     return attackLevel;
126 }
127 public int getLevelDefense(){
128     return levelDefense;
129 }
130 public int getLevelLife(){
131     return levelLife;
132 }
133 public int getActualLife(){
134     return actualLife;
135 }
136 public int getSpeed(){
137     return speed;
138 }
139 public String getAttitude(){
140     return attitude;
141 }
142 public boolean getLives(){
143     return lives;
144 }
145 public VideoJuego7 getVideoJuego7() {
146     return VideoJuego7;
147 }
148 public char getType(){
149     return type;
150 }
151 public String getReino(){
152     return reino;
153 }
```

Listing 4: Commit fa2cce4: Se completaron métodos que se utilizarán en la clase principal VideoJuego7.java

```
$ git add .
$ git commit -m "Tercera version de las clases de tipos de Soldado, ademas de modificar la
    clase principal (VideoJuego7.java)"
$ git push -u origin main
```

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

```
173 public void attack(int i){
174     advance();
175 }
176 public void defend(){
177     speed = 0;
178 }
179 public void advance(){
180     speed++;
181 }
182 public void back(){
183     if (speed > 0)
184         defend();
185     else
186         speed--;
187 }
188 public void beAttacked(){
189     actualLife--;
190     if (actualLife == 0)
191         die();
192 }
193 public void flee(){
194     speed += 2;
195 }
196 public void die(){
197     VideoJuego7.removeSoldier(this);
198     this.lives = false;
199 }
200 }
```


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

```
148 public char getType(){
149     return type;
150 }
151 public String getReino(){
152     return reino;
153 }
154
155 @Override
156 public String toString() {
157     return "Data { " +
158         "\n Name: "      + name      +
159         "\n Row: "       + row       +
160         "\n Column: "    + column    +
161         "\n Team: "      + team      +
162         "\n AttackLevel: " + attackLevel +
163         "\n LevelDefense: " + levelDefense +
164         "\n LevelLife: "   + levelLife   +
165         "\n ActualLife: "  + actualLife  +
166         "\n Speed: "      + speed      +
167         "\n Attitude: "   + attitude   +
168         "\n Lives: "      + lives      +
169         "\n Type: "      + type       +
170         "\n}\n";
171 }
172
173 public void attack(int i){
174     advance();
```

4.2. Clase Mapa.java

4.2.1. Atributos

- `private String territory`: El tipo de territorio generado.
- `String[] typesTerritory`: Contiene los tipos de territorios existentes.

4.2.2. Métodos

- `randomMapa()`: Le asigna un tipo de terreno al azar al mapa.
- Finalmente contamos con el getter y setter del atributo `territory`.

```
1 public class Mapa {
2     private String territory;
3     String[] typesTerritory = {"bosque", "campoAbierto", "montaa", "desierto", "playa"};
4
5     public String randomMapa () {
6         this.territory = typesTerritory[(int) (Math.random() * typesTerritory.length)];
7         return territory;
8     }
9
10    public void setTerritory(String territory) {
11        this.territory = territory;
12    }
13
14    public String getTerritory() {
15        return territory;
16    }
17 }
```

Listing 5: Commit e7c3920: Primera version de la clase Mapa, ya que luego se fue adaptando según se realizaban cambios en el código

```
$ git add .
$ git commit -m "Creando nuevas clases de Soldados vacias e implementando la clase
  Mapa.java"
$ git push -u origin main
```

4.3. Clase Arquero.java

4.3.1. Atributos

- `private int numFlechas`: Contiene el número de flechas que posee el Arquero.

4.3.2. Constructor

- `Arquero(...)`: Crea el Arquero, recibe algunos parámetros que serán útiles para mandarlo al superconstructor de la clase que heredó (`Soldado.java`).

4.3.3. Métodos

- `disparar()`: Contiene la acción de ataque verificando siempre que el Arquero no se quede sin flechas.

```
1 public class Arquero extends Soldado{
2     private int numFlechas;
3
4     public Arquero(VideoJuego7 VideoJuego7,String name, int row, char column, char team,
5     int speed, String attitude, boolean lives, String reino){
6         super(VideoJuego7, name, row, column, team, speed, attitude, lives, reino);
7         setAttackLevel(7);
8         setLevelDefense(3);
9         setLevelLife((int)(Math.random() * 3) + 3);
10        setActualLife(getLevelLife());
11        setType('A');
12    }
13
14    public void disparar(){
15        if (numFlechas > 0)
16            attack(1);
17            numFlechas--;
18    }
19 }
```

4.4. Clase Caballero.java

4.4.1. Atributos

- `private String arma`: Contendrá el String del arma que se tenga en acción (lanza o espada).
- `private boolean montado`: true si se encuentra montado en el caballo, de lo contrario será false.

```
1 public class Caballero extends Soldado{
2     private String arma;
3     private boolean montado;
```

4.4.2. Constructor

- `Caballero(...)`: Crea el Arquero, recibe algunos parámetros que serán útiles para mandarlo al superconstructor de la clase que heredó (`Soldado.java`).

```
5 public Caballero(VideoJuego7 VideoJuego7,String name, int row, char column, char team,
6 int speed, String attitude, boolean lives, String reino){
7     super(VideoJuego7, name, row, column, team, speed, attitude, lives, reino);
8     setAttackLevel(13);
9     setLevelDefense(7);
10    setLevelLife((int)(Math.random() * 3) + 10);
11    setActualLife(getLevelLife());
12    setType('C');
13 }
```

4.4.3. Métodos

- `alternarArma()`: Consulta que arma es la que se tiene en la mano del Caballero para intercambiarla.
- `desmontar()`: Hará que el Caballero desmonte el caballo y tenga la espada en mano.
- `montar()`: Hará que el Caballero monte el caballo y tenga la lanza en mano.
- `envestir()`: Realiza el ataque especial de envestir que tiene el Caballero, este sólo si está montado.

```
15 public void alternarArma(){
16     if (arma.equals("espada"))
17         arma = "lanza";
18     else
19         arma = "espada";
20 }
21
22 public void desmontar(){
23     if (montado){
24         montado = false;
25         arma = "espada";
26     }
27 }
28
```

```
29 public void montar(){
30     if (!montado){
31         montado = true;
32         arma = "lanza";
33     }
34 }
35
36 public void envestir(){
37     if(montado)
38         attack(3);
39     else
40         attack(2);
41 }
42 }
```

4.5. Clase Espadachin.java

4.5.1. Atributos

- `private int longitudEspada`: Guarda el tamaño de la espada que tiene el Espadachin.

```
1 public class Espadachin extends Soldado{  
2     private int longitudEspada;
```

4.5.2. Constructor

- `Espadachin(...)`: Crea el Arquero, recibe algunos parámetros que serán útiles para mandarlo al superconstructor de la clase que heredó (`Soldado.java`).

```
4 public Espadachin(VideoJuego7 VideoJuego7,String name, int row, char column, char team,  
5 int speed, String attitude, boolean lives, String reino){  
6     super(VideoJuego7, name, row, column, team, speed, attitude, lives, reino);  
7     setAttackLevel(10);  
8     setLevelDefense(8);  
9     setLevelLife((int) (Math.random() * 3) + 8);  
10    setActualLife(getLevelLife());  
11    setType('E');  
12 }
```

4.5.3. Métodos

- `crearMuroEscudo()`: Se explica que es una habilidad del Espadachin, pero no da indicaciones de las acciones que se realiza en la misma.

```
13 public void crearMuroEscudo(){  
14  
15 }
```

4.6. Clase Lancero.java

4.6.1. Atributos

- `private int longitudEspada`: Guarda el tamaño de la espada que tiene el Espadachin.

```
1 public class Lancero extends Soldado{
2     private int longLanza;
```

4.6.2. Constructor

- `Espadachin(...)`: Crea el Arquero, recibe algunos parámetros que serán útiles para mandarlo al superconstructor de la clase que heredó (`Soldado.java`).

```
4 public Lancero(VideoJuego7 VideoJuego7,String name, int row, char column, char team,
5 int speed, String attitude, boolean lives, String reino){
6     super(VideoJuego7, name, row, column, team, speed, attitude, lives, reino);
7     setAttackLevel(5);
8     setLevelDefense(10);
9     setLevelLife((int)(Math.random() * 4) + 5);
10    setActualLife(getLevelLife());
11    setType('L');
12 }
```

4.6.3. Métodos

- `schiltrom()`: Se explica que es una habilidad del Lancero la cual aumenta su nivel de defensa en 1.

```
14 public void schiltrom(){
15     setLevelDefense(getLevelDefense() + 1);
16 }
```

4.7. Clase VideoJuego7.java

4.7.1. Variables de clase

- `static HashMap <Integer, Soldado> army = new HashMap<>()`: Guarda a ambos ejércitos, es una estructura que almacena el tablero.
- `static HashMap <Integer, Soldado> army1DA = new HashMap<>()`: Almacena los soldados del ejército A.
- `static HashMap <Integer, Soldado> army1DB = new HashMap<>()`: Almacena los soldados del ejército B.
- `static String map`: Contiene el mapa en el que se desarrolla el juego.
- `static String tA`: Almacena el tipo de reino del ejército A.
- `static String tB`: Almacena el tipo de reino del ejército B.

```
11 static HashMap <Integer, Soldado> army = new HashMap<>();
12 static HashMap <Integer, Soldado> army1DA = new HashMap<>();
13 static HashMap <Integer, Soldado> army1DB = new HashMap<>();
14 static String map;
15 static String tA;
16 static String tB;
```

4.7.2. 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 `VideoJuego7`.
- Llama al método `createArmy()` para crear los ejércitos al inicio del juego.
- Llama al método `mainInterfaz()` para gestionar la interfaz principal del juego.

```
17 public static void main(String [] args){
18     VideoJuego7 videoJuego = new VideoJuego7();
19     videoJuego.createArmy();
20     videoJuego.mainInterfaz();
21 }
```


4.7.3. 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.

```
22 public void mainInterfaz(){
23     Scanner sc = new Scanner(System.in);
24     System.out.println( "1. Quick game"
25                        + "\n2. Custom game"
26                        + "\n3. Exit");
27     int action = sc.nextInt();
28     switch (action){
29         case 1 -> quickGame();
30         case 2 -> customGame();
31         case 3 -> System.out.println("Exiting the program.");
32         default -> {
33             System.out.println("Choose a valid option");
34             mainInterfaz();
35         }
36     }
37 }
```

4.7.4. 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.

```
38 public void quickGame(){
39     Scanner sc = new Scanner(System.in);
40     System.out.println("1. Crear un nuevo ejercito"
41         +"\n2. Mostrar los datos de los ejercitos"
42         +"\n3. Mostrar la tabla con los ejercitos"
43         +"\n4. Mostrar el promedio de vida de los ejercitos"
44         +"\n5. Mostrar los soldados de ejercitos con mayor vida"
45         +"\n6. Ordenar los soldados de ejercitos segun la vida"
46         +"\n7. 1v1 battle"
47         +"\n8. Salir del juego");
48     int action = sc.nextInt();
49     switch (action){
50         case 1 -> { // Crear un nuevo ejercito
51             army.clear();
52             army1DA.clear();
53             army1DB.clear();
54             createArmy();
55             quickGame();
56         }
57         case 2 -> { // Mostrar los datos de los ejercitos
58             System.out.println("DATOS DE LOS SOLDADOS CREADOS:\n");
59             showArmyData(army1DA, 'A');
60             showArmyData(army1DB, 'B');
61             quickGame();
62         }
63         case 3 -> { // Mostrar la tabla con los ejercitos
64             showArmyTable(army);
65             quickGame();
66         }
67         case 4 -> { // Mostrar el promedio de vida de los ejercitos
68             System.out.println("El promedio de vida del Ejercito A es: " +
69                 averageHealth(army1DA));
70             System.out.println("El promedio de vida del Ejercito B es: " +
71                 averageHealth(army1DB));
72             quickGame();
73         }
74         case 5 -> { // Mostrar los soldados de ejercitos con mayor vida
75             moreHelath(army1DA, 'A');
76             moreHelath(army1DB, 'B');
77             quickGame();
78         }
79         case 6 -> { // Ordenar los soldados de ejercitos segun la vida
80             System.out.println("\nEjercitos ordenados (insertionSort) segun la vida: ");
81             printArmyHealth(insertionSort(army1DA), 'A');
82             printArmyHealth(insertionSort(army1DB), 'B');
83             quickGame();
84         }
85         case 7 -> { // Jugar de 2
86             gameIntefaz();
87             quickGame();
88         }
89         case 8 -> { // Salir del juego
90             mainInterfaz();
91         }
92         default -> {
93             System.out.println("Selecciona una opcion valida");
94         }
95     }
96 }
```

```
92     quickGame();
93   }
94 }
95 }
```

4.7.5. Método para la creación de los ejército

- El método tiene como nombre `createArmy()`.
- Primero crea el mapa, para luego llamar al método `randomMapa` y establecer aleatoriamente el territorio del tablero.
- Segundo 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, el argumento tipo char para el identificador de cada equipo y ahí mismo aleatoriamente le asigna un tipo de reino, según el arreglo de reinos.

```
97 public void createArmy(){
98     Mapa m = new Mapa();
99     map = m.randomMapa();
100     int numSoldiersA = (int) (Math.random() * 10) + 1;
101     int numSoldiersB = (int) (Math.random() * 10) + 1;
102
103     String[] reinos = {"Castilla", "Aragon", "Moros", "Sacro Imperio Romano", "Germanico"};
104     army1DA = createArmyTeam(numSoldiersA, army1DA, 'A', reinos[(int) (Math.random() *
105         reinos.length)], map);
106     army1DB = createArmyTeam(numSoldiersB, army1DB, 'B', reinos[(int) (Math.random() *
107         reinos.length)], map);
108 }
```

4.7.6. 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, un char que va al final del nombre de los soldados, esto último nos ayuda a identificarlos mejor, y un String con el tipo de reino del ejército.
- Utilizando un do while se crean posiciones aleatorias en el HashMap bidimensional, tomando como condición que dicha posición sea distinta de `null`.
- Usando `Math.random` se asigna el tipo de soldado que será, ya que según sea el número aleatorio decidirá la posición en el arreglo de `typeSoldier`.
- Luego se realiza con ayuda de un switch el aumento de vida en caso el reino merezca la bonificación de nivel de vida.
- 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.

```
108 public HashMap <Integer, Soldado> createArmyTeam(int numSoldiers, HashMap <Integer,
    Soldado> army1D, char t, String r, String mapp){
109     if ('A' == t)
110         tA = r;
111     if ('B' == t)
112         tB = r;
113     String[] typeSoldier = {"Espadachin", "Caballero", "Arquero", "Lancero"};
114     for (int i = 0; i < numSoldiers; i++){
115         int row, col;
116         do {
117             row = (int) (Math.random() * 10);
118             col = (int) (Math.random() * 10);
119         } while (army.containsKey(row * 10 + col));
120
121         int typ = (int) (Math.random() * typeSoldier.length);
122         switch (typ) {
123             case 1 -> {
124                 Espadachin e = new Espadachin(VideoJuego7.this, "Soldier" + i + "X" + t, row + 1,
                    (char) (col + 'A'), t,
125                 (int)(Math.random() * 5) + 1, "Defensiva", true, r);
126                 if (e.getReino().equals(mapp))
127                     e.setActualLife(e.getActualLife() + 1);
128                 army1D.put(i, e);
129                 army.put(row * 10 + col, e);
130             }
131             case 2 -> {
132                 Caballero c = new Caballero(VideoJuego7.this, "Soldier" + i + "X" + t, row + 1,
                    (char) (col + 'A'), t,
133                 (int)(Math.random() * 5) + 1, "Defensiva", true, r);
134                 if (c.getReino().equals(mapp))
135                     c.setActualLife(c.getActualLife() + 1);
136                 army1D.put(i, c);
137                 army.put(row * 10 + col, c);
138             }
139             case 3 -> {
140                 Arquero a = new Arquero(VideoJuego7.this, "Soldier" + i + "X" + t, row + 1, (char)
                    (col + 'A'), t,
141                 (int)(Math.random() * 5) + 1, "Defensiva", true, r);
142                 if (a.getReino().equals(mapp))
143                     a.setActualLife(a.getActualLife() + 1);
144                 army1D.put(i, a);
145                 army.put(row * 10 + col, a);
146             }
147             case 4 -> {
148                 Lancero l = new Lancero(VideoJuego7.this, "Soldier" + i + "X" + t, row + 1, (char)
                    (col + 'A'), t,
149                 (int)(Math.random() * 5) + 1, "Defensiva", true, r);
150                 if (l.getReino().equals(mapp))
151                     l.setActualLife(l.getActualLife() + 1);
152                 army1D.put(i, l);
153                 army.put(row * 10 + col, l);
154             }
155         }
156     }
157     return army1D;
158 }
```

Listing 6: Commit bb6e2c8: Dentro de este commit se adaptó el código para incluir el tema de los tipos de territorio y soldados.

```
$ git add .
$ git commit -m "Cuarta version de las clases de tipos de Soldado, ademas de modificar la
  clase principal (VideoJuego7.java)"
$ git push -u origin main
```

4.7.7. 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, siendo su funcionalidad la requerida y solicitada en las consignas del laboratorio.

```
160 public void showArmyTable(HashMap <Integer, Soldado> army){
161     System.out.println("\t" + map + "\n");
162     String azul = "\u001B[34m";
163     String rojo = "\u001B[31m";
164     String reset = "\u001B[0m";
165     System.out.println("\n                TABLA CON LAS UBICACIONES DE LOS SOLDADOS CREADOS:
166         \n");
167     String linesDown = "
168         |_____|_____|_____|_____|_____|_____|_____|_____|_____|_____|";
169     System.out.println("         A         B         C         D         E         F         G         H         I
170         J\n"
171         + "
172         -----");
173     for (int r = 0; r < 10; r++){
174         System.out.print(" |");
175         for (int c = 0; c < 10; c++){
176             System.out.print(" " + (army.get(r*10+c) != null ? ((army.get(r*10+c).getTeam() ==
177                 'A' ? azul: rojo) + "\' " + army.get(r*10+c).getTeam() + "\' "
178                 + army.get(r*10+c).getType() + army.get(r*10+c).getName().charAt(7)+ reset + " |") :
179                 " |"));
180         }
181         System.out.print("\n" + (r+1) + ((r != 9) ? " |" : " |"));
182         for (int c = 0; c < 10; c++){
183             System.out.print(" " + (army.get(r*10+c) != null ? (army.get(r*10+c).getTeam() == 'A'
184                 ? azul: rojo) + "HP: " + army.get(r*10+c).getActualLife() + reset +
185                 ((army.get(r*10+c).getActualLife() >= 10) ? "|": " |"): " |")
186             );
187         }
188         System.out.println("\n" + linesDown );
189     }
190     mostrarData(army1DA, "A");
191     mostrarData(army1DB, "B");
192     System.out.println();
193     mostrarInfo(army1DA, army1DB);
194 }
```

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

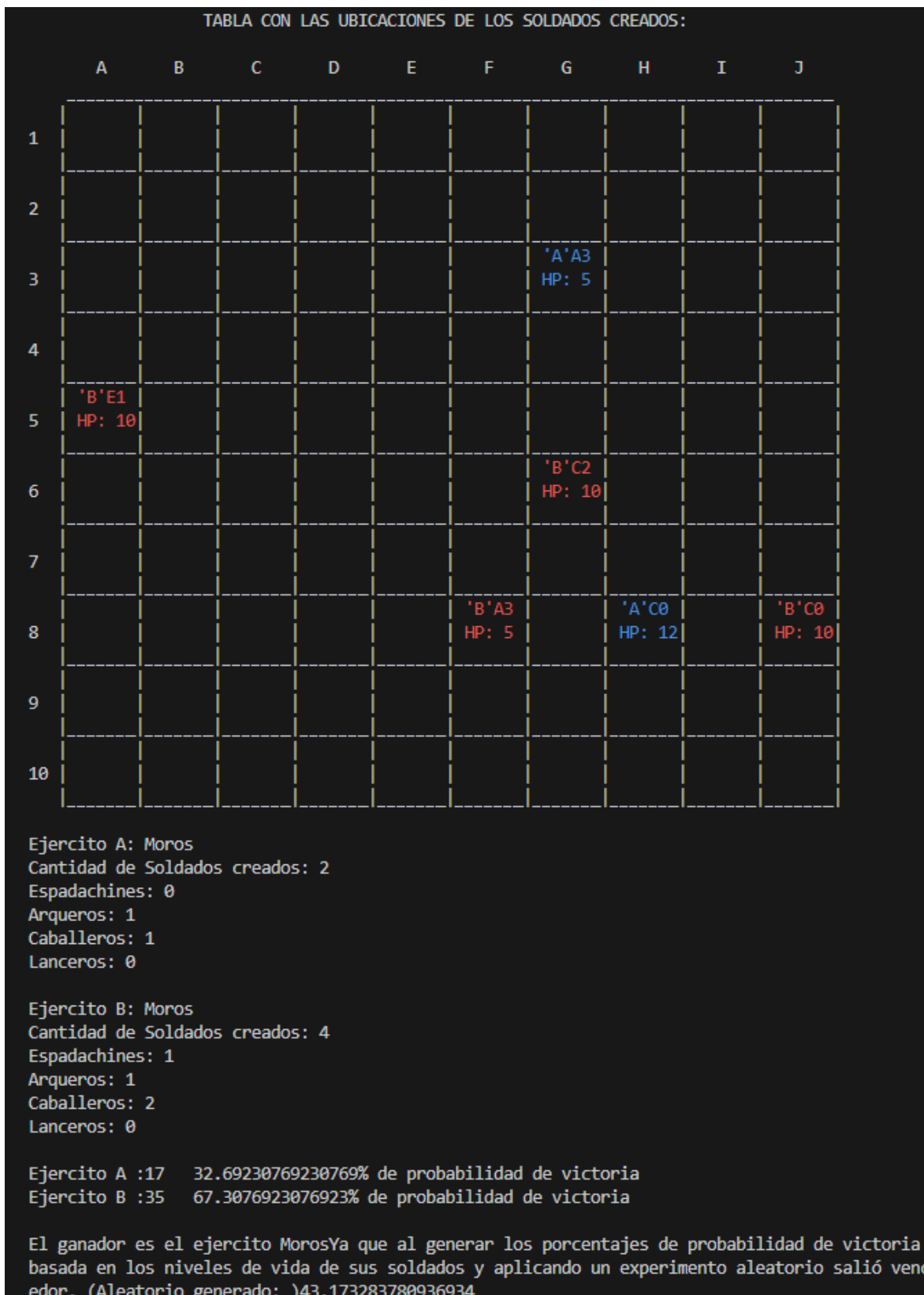


Figura 1

4.7.8. Método para mostrar datos generales del ejército respecto a los soldados

- El método tiene como nombre `mostrarData()`.
- El método accede a los tipos de soldado de cada uno perteneciente al ejército seleccionado, utilizando un `foreach` va contabilizando cuantos soldados de cada tipo tiene dicho ejército.
- Finalizando muestra el resultado del conteo.

```
187 public void mostrarData(HashMap<Integer, Soldado> am, String t){
188     System.out.println("\nEjercito " + t + ": " + (t == "A" ? tA : tB));
189     System.out.println("Cantidad de Soldados creados: " + am.size());
190     int es = 0;
191     int ar = 0;
192     int ca = 0;
193     int la = 0;
194     for(Map.Entry<Integer, Soldado> e : am.entrySet()){
195         es += e.getValue().getType() == 'E' ? 1 : 0;
196         ar += e.getValue().getType() == 'A' ? 1 : 0;
197         ca += e.getValue().getType() == 'C' ? 1 : 0;
198         la += e.getValue().getType() == 'L' ? 1 : 0;
199     }
200     System.out.println("Espadachines: " + es);
201     System.out.println("Arqueros: " + ar);
202     System.out.println("Caballeros: " + ca);
203     System.out.println("Lanceros: " + la);
204 }
```

4.7.9. Método para calcular las probabilidades de los ejércitos y mostrar un ganador

- El método tiene como nombre `mostrarInfo()`.
- Siguiendo la métrica dada en las indicaciones se calcula la probabilidad de ser vencedores en cada ejército, mostrarlas y finalmente dar un ganador en base a las probabilidades realizadas.

```
206 public void mostrarInfo(HashMap<Integer, Soldado> a1, HashMap<Integer, Soldado> a2){
207     int sumA = 0;
208     int sumB = 0;
209     for(Map.Entry<Integer, Soldado> e : a1.entrySet())
210         sumA += e.getValue().getLevelLife();
211     for(Map.Entry<Integer, Soldado> e : a2.entrySet())
212         sumB += e.getValue().getLevelLife();
213
214     double probA = ((double) sumA / (sumA + sumB)) * 100;
215     double probB = ((double) sumB / (sumA + sumB)) * 100;
216     System.out.println("Ejercito A : " + sumA + "\t " + probA + "% de probabilidad de
    victoria");
217     System.out.println("Ejercito B : " + sumB + "\t " + probB + "% de probabilidad de
    victoria\n");
218
219     Random random = new Random();
220     double rand = random.nextDouble() * (probA + probB);
221     String win;
222     if (rand < probA) {
223         win = tA;
```

```
224     } else {
225         win = tB;
226     }
227     System.out.println("El ganador es el ejercito " + win + "Ya que al generar los
        porcentajes " +
228     "de probabilidad de victoria basada en los niveles de vida de sus soldados y aplicando
        un " +
229     "experimento aleatorio sali vencedor. (Aleatorio generado: )" + rand + "\n");
230 }
```

4.7.10. 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.

```
232 public void showArmyData(HashMap <Integer, Soldado> army1D, char t){
233     System.out.println("EJERCITO \"" + t + "\"");
234     for (Soldado s : army1D.values())
235         System.out.println(s);
236 }
```

4.7.11. 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.

```
238 public void moreHelath(HashMap <Integer, Soldado> army1MH, char t){
239     int maxHealth = -1;
240     for(Soldado s : army1MH.values())
241         if (s.getActualLife() > maxHealth)
242             maxHealth = s.getActualLife();
243
244     System.out.println("Soldado(s) con mayor vida del Ejercito " + t + ": ");
245     for (Soldado s : army1MH.values())
246         if (s.getActualLife() == maxHealth)
247             System.out.println("Nombre: " + s.getName() + " Vida: " + s.getActualLife());
248     System.out.println();
249 }
```


4.7.12. 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.

```
251 public double averageHealth(HashMap <Integer, Soldado> army1DAH){  
252     return sumHealth(army1DAH) / army1DAH.size();  
253 }
```

4.7.13. 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`.

```
255 public int sumHealth(HashMap <Integer, Soldado> army1DSH){  
256     int sum = 0;  
257     for (Soldado s : army1DSH.values())  
258         sum += s.getActualLife();  
259     return sum;  
260 }
```

4.7.14. 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 "llave" compara su salud con la de los soldados anteriores.
 - Los soldados 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 $O(n^2)$.

```
272 public HashMap <Integer, Soldado> insertionSort(HashMap <Integer, Soldado> army1DIS) {  
273     int n = army1DIS.size();  
274     HashMap<Integer, Soldado> army1DCopyInsertion = new HashMap<>(army1DIS);  
275     for (int i = 1; i < n; i++) {  
276         Soldado key = army1DCopyInsertion.get(i);  
277         int j = i - 1;  
278  
279         while (j >= 0 && army1DCopyInsertion.get(j).getActualLife() < key.getActualLife()) {
```

```
280     army1DCopyInsertion.put(j+1, army1DCopyInsertion.get(j));
281     j = j - 1;
282 }
283     army1DCopyInsertion.put(j+1, key);
284 }
285     return army1DCopyInsertion;
286 }
```

4.7.15. 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.

```
278 public void printArmyHealth(HashMap <Integer, Soldado> armyPrint, char t){
279     System.out.println("EJERCITO " + t + " : ");
280     for(int i = 0; i < armyPrint.size(); i++)
281         System.out.println((i + 1) + ". " + armyPrint.get(i).getName() + " Vida: " +
282             armyPrint.get(i).getActualLife());
282 }
```

4.7.16. 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.

```
284 public void customGame(){
285     Scanner sc = new Scanner(System.in);
286     System.out.println("Select army to customize:\n 1. A\n 2. B");
287     int actionTeam = sc.nextInt();
288     if (actionTeam == 1) // A
289         customGameArmy(army1DA, 'A');
290     else if (actionTeam == 2) // B
291         customGameArmy(army1DB, 'B');
292     else{
293         System.out.println("Invalid army, try again");
294         customGame();
295     }
296 }
```

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

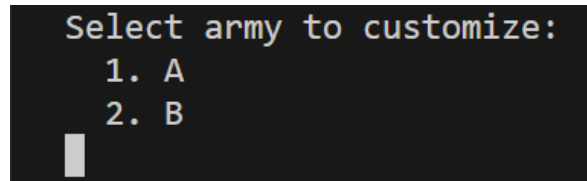


Figura 2

4.7.17. 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 válido que será la opción elegida por el usuario, para luego hacer un llamado al método correspondiente.

```

298 public void customGameArmy(HashMap <Integer, Soldado> armyA, char t){
299     Scanner sc = new Scanner(System.in);
300     System.out.println("Selected " + t + " army");
301     System.out.println( " 1. Create Soldier"
302                         + "\n 2. Delete Soldier"
303                         + "\n 3. Clone Soldier"
304                         + "\n 4. Modify Soldier"
305                         + "\n 5. Compare Soldiers"
306                         + "\n 6. Swap Soldiers"
307                         + "\n 7. View Soldier"
308                         + "\n 8. See Army"
309                         + "\n 9. Add Levels"
310                         + "\n10. Play"
311                         + "\n11. Return");
312     int action = sc.nextInt();
313     switch (action){
314         case 1 -> createSoldier(armyA, t);
315         case 2 -> deleteSoldier(armyA, t);
316         case 3 -> cloneSoldier(armyA, t);
317         case 4 -> modifySoldier(armyA, t);
318         case 5 -> compareSoldiers(armyA, t);
319         case 6 -> swapSoldiers(armyA, t);
320         case 7 -> viewSoldier(armyA, t);
321         case 8 -> seeArmy(armyA, t);
322         case 9 -> addLevels(armyA, t);
323         case 10 -> play();
324         case 11 -> mainInterfaz();
325         default -> {
326             System.out.println("Choose a valid option");
327             customGameArmy(armyA, t);
328         }
329     }
330     System.out.println("Returning to the menu");
331     customGameArmy(armyA, t);
332 }

```

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

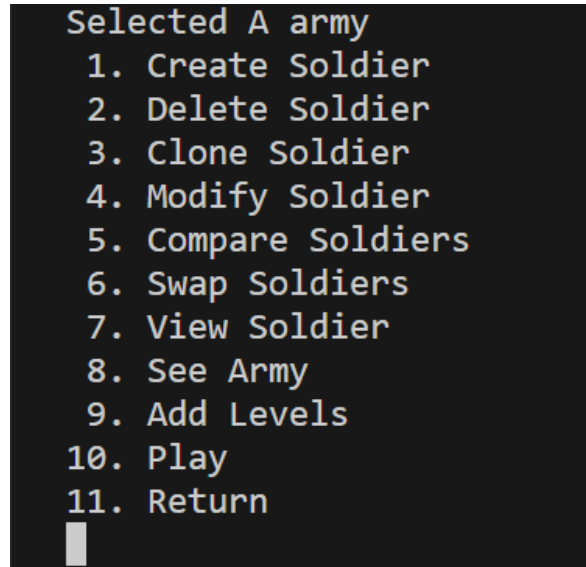


Figura 3

4.7.18. Método para asignar los cambios realizados al ejército 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.

```
334 public void assignModification(HashMap <Integer, Soldado> armyMod, char t){  
335     if (t == 'A')  
336         army1DA = armyMod;  
337     else  
338         army1DB = armyMod;  
339 }
```

4.7.19. 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.

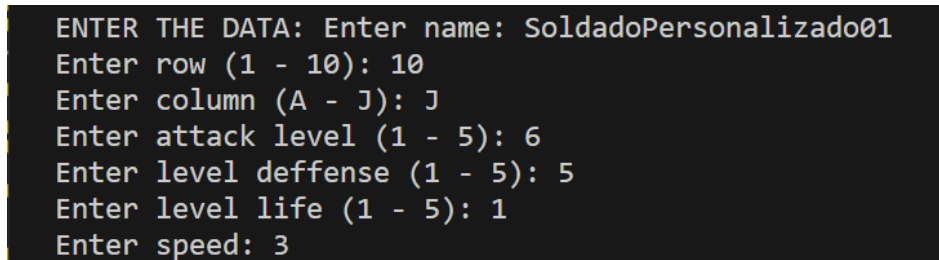
```
341 public void createSoldier(HashMap <Integer, Soldado> armyMod, char t){  
342     Scanner sc = new Scanner(System.in);  
343     if (armyMod.size() < 10){  
344         System.out.print("ENTER THE DATA: ");  
345         System.out.print("Enter name: ");
```

```

346     String name = sc.next();
347     int position = createPosition(armyMod);
348     System.out.print("Enter attack level (1 - 5): ");
349     int attackLevel = sc.nextInt();
350     System.out.print("Enter level deffense (1 - 5): ");
351     int levelDefense = sc.nextInt();
352     System.out.print("Enter level life (1 - 5): ");
353     int levelLife = sc.nextInt();
354     System.out.print("Enter speed: ");
355     int speed = sc.nextInt();
356     Soldado s = new Soldado(VideoJuego7.this, name, position / 10 + 1, (char)(position % 10
        + 'A'), t, attackLevel, levelDefense, levelLife, speed);
357     armyMod.put(armyMod.size(), s);
358     army.put(position / 10 + position % 10, s);
359 }
360 else
361     System.out.println("The army reached the limit of allowed soldiers");
362     assignModification(armyMod, t);
363 }

```

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



```

ENTER THE DATA: Enter name: SoldadoPersonalizado01
Enter row (1 - 10): 10
Enter column (A - J): J
Enter attack level (1 - 5): 6
Enter level deffense (1 - 5): 5
Enter level life (1 - 5): 1
Enter speed: 3

```

Figura 4

4.7.20. 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.

```

365 public int createPosition(HashMap <Integer, Soldado> armyMod){
366     Scanner sc = new Scanner(System.in);
367     int row;
368     char column;
369     do{
370         System.out.print("Enter row (1 - 10): ");
371         row = sc.nextInt();
372         System.out.print("Enter column (A - J): ");
373         column = sc.next().charAt(0);
374     } while(armyMod.get((row-1)*10 + (column - 'A')) != null);
375     return (row-1)*10 + (column - 'A');
376 }

```

4.7.21. 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.

```

378 public void deleteSoldier(HashMap <Integer, Soldado> armyMod, char t){
379     Scanner sc = new Scanner(System.in);
380     if (armyMod.size() > 1){
381         System.out.println("ARMY \" + t + "\"");
382         for (Soldado s : armyMod.values())
383             System.out.println(s.getName());
384
385         System.out.println("Enter the name of the soldier to be eliminated");
386         String sName = sc.next();
387         Soldado soldierToDelete = null;
388         for (Soldado s : armyMod.values())
389             if (s.getName().equals(sName)){
390                 soldierToDelete = s;
391                 break;
392             }
393         if (soldierToDelete != null) {
394             armyMod.remove(Integer.parseInt(soldierToDelete.getName().substring(7,8)));
395             army.remove((soldierToDelete.getRow() - 1) * 10 + (soldierToDelete.getColumn() -
396                 'A'));
396             System.out.println("Soldier successfully eliminated: " + soldierToDelete.getName());
397         } else {
398             System.out.println("Soldier not found. Try again.");
399             deleteSoldier(armyMod, t);
400         }
401     } else {
402         System.out.println("No soldiers in the " + t + " army.");
403     }
404     assignModification(armyMod, t);
405 }

```

- 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

4.7.22. 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.

```
407 public void cloneSoldier(HashMap<Integer, Soldado> armyMod, char t) {
408     Scanner sc = new Scanner(System.in);
409     if (armyMod.size() < 10) {
410         for (Soldado s : armyMod.values())
411             System.out.println(s.getName());
412
413         System.out.println("Enter the name of the soldier to clone:");
414         String originalSoldierName = sc.next();
415
416         Soldado originalSoldado = null;
417         for (Soldado s : armyMod.values())
418             if (s.getName().equals(originalSoldierName)) {
419                 originalSoldado = s;
420                 break;
421             }
422         if (originalSoldado != null){
423             int row, col;
424             do {
425                 System.out.print("Enter the row (1 - 10) for the cloned soldier: ");
426                 row = sc.nextInt();
427                 System.out.print("Enter the column (A - J) for the cloned soldier: ");
428                 col = Character.toUpperCase(sc.next().charAt(0)) - 'A';
429             } while (armyMod.get(row * 10 + col) != null);
430
431             Soldado clonedSoldado = new Soldado(originalSoldado.getVideoJuego7(), "Soldier" +
432                 armyMod.size() + "X" + t,
433                 row + 1, (char) (col + 'A'), t, originalSoldado.getAttackLevel(),
434                 originalSoldado.getLevelDefense(),
435                 originalSoldado.getLevelLife(),
436                 originalSoldado.getSpeed(), "Defensiva", true);
437             armyMod.put(armyMod.size(), clonedSoldado);
438             army.put(row * 10 + col, clonedSoldado);
439
440             System.out.println("Soldier cloned successfully: " + clonedSoldado.getName());
441         }
442         else {
443             System.out.println("Soldier not found. Try again.");
444             cloneSoldier(armyMod, t);
445         }
446     } else
447         System.out.println("The army reached the limit of allowed soldiers.");
448     assignModification(armyMod, t);
449 }
```

- 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

4.7.23. 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.

```
448 public void modifySoldier(HashMap<Integer, Soldado> armyMod, char t) {
449     Scanner sc = new Scanner(System.in);
450     System.out.println("ARMY \" + t + "\"");
451     for (Soldado s : armyMod.values())
452         System.out.println(s.getName());
453
454     System.out.println("Enter the name of the soldier to modify:");
455     String soldierName = sc.next();
456     Soldado soldierToModify = null;
457     for (Soldado s : armyMod.values())
458         if (s.getName().equals(soldierName)) {
459             soldierToModify = s;
460             break;
461         }
462
463     if (soldierToModify != null) {
464         System.out.println("Modify Soldier - " + soldierToModify.getName());
465         System.out.println("1. Modify Attack Level");
466         System.out.println("2. Modify Defense Level");
467         System.out.println("3. Modify Life Level");
468         System.out.println("4. Return");
469         int choice = sc.nextInt();
470         switch (choice) {
471             case 1 -> {
472                 System.out.print("Enter new Attack Level (1 - 5): ");
473                 int newAttackLevel = sc.nextInt();
```



```
474     soldierToModify.setAttackLevel(newAttackLevel);
475 }
476 case 2 -> {
477     System.out.print("Enter new Defense Level (1 - 5): ");
478     int newDefenseLevel = sc.nextInt();
479     soldierToModify.setLevelDefense(newDefenseLevel);
480 }
481 case 3 -> {
482     System.out.print("Enter new Life Level (1 - 5): ");
483     int newLifeLevel = sc.nextInt();
484     soldierToModify.setLevelLife(newLifeLevel);
485 }
486 case 4 -> System.out.println("Returning to the menu");
487 default -> {
488     System.out.println("Invalid choice. Returning to the menu.");
489 }
490 }
491 } else {
492     System.out.println("Soldier not found. Try again.");
493     modifySoldier(armyMod, t);
494 }
495 }
```

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

```
ARMY "A"
Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
Soldier9XA
Enter the name of the soldier to modify:
Soldier1XA
Modify Soldier - Soldier1XA
1. Modify Attack Level
2. Modify Defense Level
3. Modify Life Level
4. Return
1
Enter new Attack Level (1 - 5): 5
```

Figura 7

4.7.24. 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.

```

497 public void compareSoldiers(HashMap<Integer, Soldado> armyMod, char t){
498     Scanner sc = new Scanner(System.in);
499     for (Soldado s : armyMod.values())
500         System.out.println(s.getName());
501
502     System.out.println("Enter the name of the first soldier:");
503     String soldierName1 = sc.next();
504     System.out.println("Enter the name of the second soldier:");
505     String soldierName2 = sc.next();
506
507     Soldado soldier1 = findSoldado(armyMod, soldierName1);
508     Soldado soldier2 = findSoldado(armyMod, soldierName2);
509
510     if (soldier1 != null && soldier2 != null){
511         if (compareSoldadoAttributes(soldier1, soldier2))
512             System.out.println("Soldiers are identical.");
513         else
514             System.out.println("Soldiers are different.");
515     }
516     else{
517         System.out.println("Soldier not found. Try again.");
518         compareSoldiers(armyMod, t);
519     }
520 }

```

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

```

Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
Soldier9XA
Enter the name of the first soldier:
Soldier1XA
Enter the name of the second soldier:
Soldier1XA
Soldiers are identical.

```

Figura 8

4.7.25. Método para buscar un soldado

- El método tiene como nombre `findSoldado()`.
- Se envía 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.

```
522 public Soldado findSoldado(HashMap<Integer, Soldado> armyMod, String soldierName) {  
523     for (Soldado s : armyMod.values())  
524         if (s.getName().equals(soldierName))  
525             return s;  
526     return null;  
527 }
```

4.7.26. 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.

```
529 public boolean compareSoldadoAttributes(Soldado s1, Soldado s2) {  
530     return s1.getName().equals(s2.getName()) &&  
531         s1.getAttackLevel() == s2.getAttackLevel() &&  
532         s1.getLevelDefense() == s2.getLevelDefense() &&  
533         s1.getActualLife() == s2.getActualLife() &&  
534         s1.getLives() == s2.getLives();  
535 }
```

4.7.27. 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.

```
537 public void swapSoldiers(HashMap<Integer, Soldado> armyMod, char t) {  
538     Scanner sc = new Scanner(System.in);  
539  
540     System.out.println("ARMY \" + t + "\"");  
541     for (Soldado soldado : armyMod.values())  
542         System.out.println(soldado.getName());  
543  
544     System.out.println("Enter the name of the first soldier to swap:");  
545     String sName1 = sc.next();  
546     Soldado soldier1 = findSoldado(armyMod, sName1);  
547  
548     System.out.println("Enter the name of the second soldier to swap:");  
549     String sName2 = sc.next();  
550     Soldado soldier2 = findSoldado(armyMod, sName2);  
551  
552     if (soldier1 != null && soldier2 != null) {
```

```

553     System.out.println("Swapping Soldiers - " + soldier1.getName() + " and " +
        soldier2.getName());
554     int position1 = (soldier1.getRow() - 1) * 10 + (soldier1.getColumn() - 'A');
555     int position2 = (soldier2.getRow() - 1) * 10 + (soldier2.getColumn() - 'A');
556
557     armyMod.remove(position1);
558     armyMod.remove(position2);
559
560     armyMod.put(position1, soldier2);
561     armyMod.put(position2, soldier1);
562
563     System.out.println("Soldiers swapped successfully.");
564 } else {
565     System.out.println("One or both soldiers not found. Try again.");
566     swapSoldiers(armyMod, t);
567 }
568 }

```

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

```

ARMY "A"
Soldier1XA
Soldier2XA
Soldier3XA
Soldier4XA
Soldier5XA
Soldier6XA
Soldier7XA
Soldier8XA
Soldier9XA
Enter the name of the first soldier to swap:
Soldier1XA
Enter the name of the second soldier to swap:
Soldier6XA
Swapping Soldiers - Soldier1XA and Soldier6XA
Soldiers swapped successfully.

```

Figura 9

Listing 7: Commit e1a4207: Concluyendo las peticiones requeridas en la clase VideoJuego7.java

```

$ git add .
$ git commit -m "Concluyendo las peticiones requeridas en la clase VideoJuego7.java"
$ git push -u origin main

```

4.7.28. 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.

```

570 public void viewSoldier(HashMap<Integer, Soldado> armyMod, char t) {
571     Scanner sc = new Scanner(System.in);
572     for (Soldado soldado : armyMod.values())
573         System.out.println(soldado.getName());
574     System.out.println("Enter the name of the soldier to view:");
575     String soldierName = sc.next();
576     Soldado soldierToView = findSoldado(armyMod, soldierName);
577     if (soldierToView != null) {
578         System.out.println("Soldier Details:");
579         System.out.println("Name: " + soldierToView.getName());
580         System.out.println("Team: " + soldierToView.getTeam());
581         System.out.println("Position: " + soldierToView.getRow() + " " +
582             soldierToView.getColumn());
583         System.out.println("Attack Level: " + soldierToView.getAttackLevel());
584         System.out.println("Defense Level: " + soldierToView.getLevelDefense());
585         System.out.println("Life Level: " + soldierToView.getLevelLife());
586         System.out.println("Speed: " + soldierToView.getSpeed());
587     } else {
588         System.out.println("Soldier not found. Try again.");
589         viewSoldier(armyMod, t);
590     }
591 }

```

- 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

4.7.29. 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.

```
592 public void seeArmy(HashMap<Integer, Soldado> armyMod, char t) {
593     System.out.println("EJERCITO \" + t + "\");
594     for (Soldado s : armyMod.values()) {
595         System.out.println("Soldier Details:");
596         System.out.println("Name: " + s.getName());
597         System.out.println("Team: " + s.getTeam());
598         System.out.println("Position: " + s.getRow() + " " + s.getColumn());
599         System.out.println("Attack Level: " + s.getAttackLevel());
600         System.out.println("Defense Level: " + s.getLevelDefense());
601         System.out.println("Life Level: " + s.getLevelLife());
602         System.out.println("Speed: " + s.getSpeed());
603         System.out.println();
604     }
605     customGameArmy(armyMod, t);
606 }
```

4.7.30. 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.

```
608 public void addLevels(HashMap<Integer, Soldado> armyMod, char t) {
609     int totalAttackLevel = 0;
610     int totalDefenseLevel = 0;
611     int totalLifeLevel = 0;
612     int totalSpeed = 0;
613     for (Soldado soldier : armyMod.values()) {
614         totalAttackLevel += soldier.getAttackLevel();
615         totalDefenseLevel += soldier.getLevelDefense();
616         totalLifeLevel += soldier.getLevelLife();
617         totalSpeed += soldier.getSpeed();
618     }
619     System.out.println("Sumatoria de niveles del Ejercito " + t + ":");
620     System.out.println("Sumatoria de Nivel de Ataque: " + totalAttackLevel);
621     System.out.println("Sumatoria de Nivel de Defensa: " + totalDefenseLevel);
622     System.out.println("Sumatoria de Nivel de Vida: " + totalLifeLevel);
623     System.out.println("Sumatoria de Velocidad: " + totalSpeed);
624 }
```

- 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

4.7.31. 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.

```
626 public void play(){  
627     gameIntefaz();  
628 }
```

4.7.32. Método para mostrar al ejército ganador

- El método tiene como nombre `armyWinner()`.
- 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.

```
630 public void armyWinner(){  
631     if(army1DA.size() > army1DB.size())  
632         System.out.print("A");  
633     else  
634         System.out.print("B");  
635 }
```

4.7.33. 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.

```
637 public void removeSoldier(Soldado s){  
638     if (s.getTeam() == 'A')  
639         army1DA.remove(s.getName().charAt(7) - '0');  
640     else  
641         army1DB.remove(s.getName().charAt(7) - '0');  
642     army.remove((s.getRow()-1)*10 + (s.getColumn() - 'A'));  
643 }
```

4.7.34. 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 romperá el bucle `while` y se dará el nombre del ganador usando el método `armyWinner()`.

```
645 public void gameInterfaz(){
646     Scanner sc = new Scanner(System.in);
647     System.out.println("Starting Game...");
648     boolean noEnd = true;
649     while (noEnd) {
650         System.out.println("Team A Turn");
651         turn('A');
652         if (army1DA.size() == 0)
653             break;
654         System.out.println("Team B Turn");
655         turn('B');
656         if (army1DB.size() == 0)
657             break;
658     }
659     System.out.println("Winning team: ");
660     armyWinner();
661 }
```

4.7.35. 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()`.

```
663 public void turn(char teamT){
664     showArmyTable(army);
665     Scanner sc = new Scanner(System.in);
666     System.out.println("Indicate the coordinates of the soldier to move. Ej: 1 A (put the
        space in the middle)");
667     int row1 = sc.nextInt() - 1;
668     char l1 = sc.next().charAt(0);
669     int col1 = Character.toUpperCase(l1) - 'A';
670     if (checkSoldier1(row1, col1, teamT))
671         turn2(row1, col1, teamT);
672     else
673         turn(teamT);
674 }
```


4.7.36. 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.

```
676 public boolean checkSoldier1(int row, int col, char team){
677     if(row > 9 || col > 9)
678         return false;
679     if (army.get(row*10 + col) != null){
680         if(army.get(row*10 + col).getTeam() == team)
681             return true;
682         else{
683             System.out.println("Choose a soldier from your team");
684             return false;
685         }
686     }
687     return false;
688 }
```

4.7.37. 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.

```
690 public int checkSoldier2(int row, int col, char team){
691     if(row > 9 || col > 9)
692         return 4;
693     if (army.get(row*10 + col) == null)
694         return 1;
695     else if (army.get(row*10 + col).getTeam() != team)
696         return 3;
697     return 2;
698 }
```

4.7.38. Método para ejecutar el movimiento elegido

- El método tiene como nombre `turn2()`.
- Este 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.

```
700 public void turn2(int row1, int col1, char teamT){
701     Scanner sc = new Scanner(System.in);
702     System.out.println("Indicate the coordinates where you want to move the soldier. Ej: 1 A
       (put the space in the middle)");
703     int row2 = sc.nextInt() - 1;
704     char l2 = sc.next().charAt(0);
705     int col2 = Character.toUpperCase(l2) - 'A';
706     switch(checkSoldier2(row2, col2, teamT)){
707         case 1 -> moveSoldier(row1, col1, row2, col2);
708         case 2 -> {
709             System.out.println("There cannot be two soldiers in the same position, try again");
710             turn2(row1, col1, teamT);
711         }
712         case 3 ->{
713             System.out.println("--SOLDIERS FIGHT--");
714             System.out.println(army.get(row1*10 + col1).getName() + " vs " + army.get(row2*10 +
              col2).getName());
715             Soldado sW = soldiersFight(army.get(row1*10 + col1), army.get(row2*10 + col2), row2,
              col2, row1, col1);
716         }
717         case 4 ->{
718             System.out.println("Invalid position, try again");
719             turn2(row1, col1, teamT);
720         }
721     }
722 }
```

4.7.39. 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.

```
724 public void moveSoldier(int row1, int col1, int row2, int col2){
725     army.put(row2*10 + col2, army.get(row1*10 + col1));
726     army.remove(row1*10 + col1);
727 }
```

4.7.40. 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.

```
729 public void moveSoldier(Soldado s,int row1, int col1, int rowD, int colD){
730     army.remove(rowD*10 + colD);
731     army.put(row1*10 + col1, s);
732 }
```

4.7.41. 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 elige 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.

```
734 public Soldado soldiersFight(Soldado a, Soldado b, int rowP, int colP, int rowD, int colD){
735     double probabilityA = (double) a.getActualLife() / (a.getActualLife() +
736         b.getActualLife()) * 100;
737     double probabilityB = (double) b.getActualLife() / (a.getActualLife() +
738         b.getActualLife()) * 100;
739     System.out.println("Probability of " + a.getName() + ": " + probabilityA + "%");
740     System.out.println("Probability of " + b.getName() + ": " + probabilityB + "%");
741     Random random = new Random();
742     double probabilityT = probabilityA + probabilityB;
743     double randomValue = random.nextDouble() * probabilityT;
744     Soldado winner;
745     if (randomValue < probabilityA) {
746         System.out.println("The winner is: " + a.getName());
747         b.die();
748         winner = a;
749     } else {
750         System.out.println("The winner is: " + b.getName());
751         a.die();
752         winner = b;
753     }
754     winner.setActualLife(winner.getActualLife() + 1);
755     moveSoldier(winner, rowP, colP, rowD, colD);
756     return winner;
757 }
```

Listing 8: Commit 0836847: Versión final del código (lab20)

```
$ git add .
$ git commit -m "Ultimas modificaciones, version final del lab20"
$ git push -u origin main
```

4.8. Diagrama de clase UML

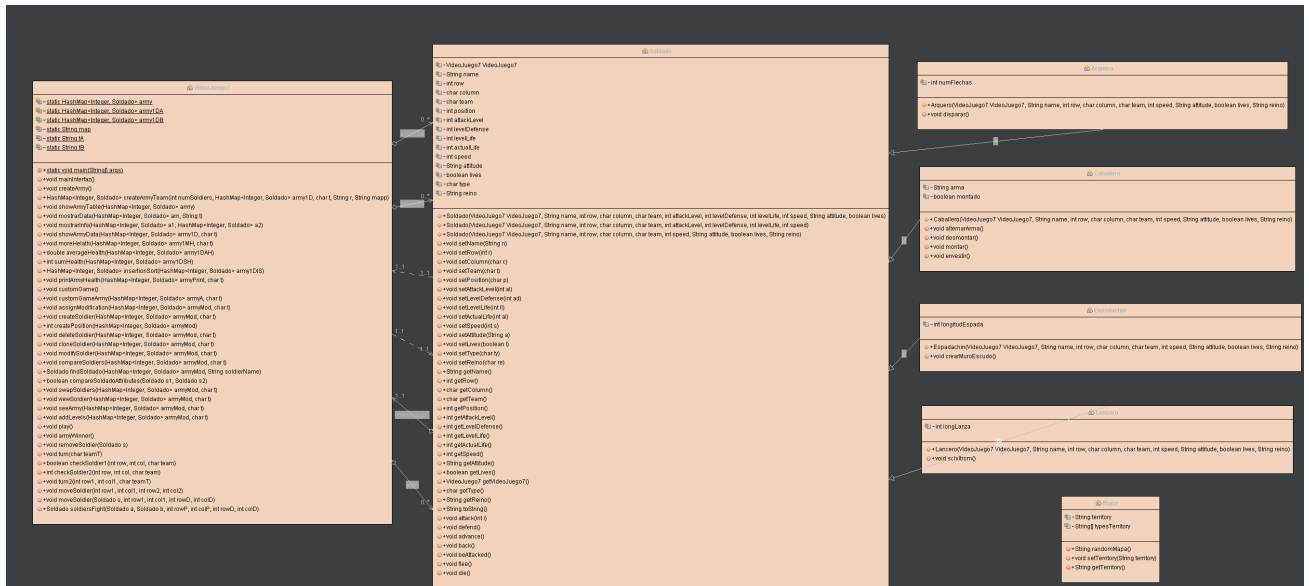


Figura 12

Diagrama 20 UML para acceder al diagrama UML del repositorio y se observe con más claridad.

4.9. Estructura de laboratorio 20

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

```
lab20
|  Soldado.java
|  VideoJuego7.java
|
|-----latex
|    Informe_Lab20.pdf
|    Informe_Lab20.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
|    20uml.png
|    logo_abet.png
|    logo_unsa.jpg
|    showArmyTable.png
|
|-----src
|    Arquero.java
|    Caballero.java
|    Espadachin.java
|    Lancero.java
|    Mapa.java
|    Soldado.java
|    VideoJuego7.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 los 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/java00/methods.html>
- <https://www.geeksforgeeks.org/insertion-sort/>
- <https://es.stackoverflow.com/questions/108171/>
- <https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html>
- <https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html>