Informe de Laboratorio 20

Tema: Miembros de clase

Nota

Estudiante	Escuela	Asignatura
Ryan Fabian Valdivia Segovia rvaldiviase@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
20	Miembros de clase	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 08 de Enero 2023	Al 15 de Enero 2023

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 espadaz como acción çrear 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 envestir). El caballero también puede envestir, 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 lanzaz 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 Home Single Language 64 bits 22621.2283
- VIM 9.0.
- Visual Studio Code 64 bits 1.82.2

- OpenJDK 64-Bits 11.0.16.1
- Git 2.41.0.windows.1
- IntelliJ IDEA 2023.3 Runtime version: 17.0.9+7-b1087.7 amd64
- Cuenta en GitHub con el correo institucional.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/RyanValdivia/fp2-23b`
- URL para el laboratorio 20 en el Repositorio GitHub.
- `https://github.com/RyanValdivia/fp2-23b/tree/main/fase03/lab20`

4. Actividades

- En primer lugar, y basándome en los laboratorios anteriores, me di cuenta que podría reducir bastante la carga en la clase VideoJuego si creaba clases auxiliares y las utilizaba para realizar diferentes tareas, es por ello que dividí y definí varias clases
- Primero, tengo la clase Soldado, para la cual reutilicé código de laboratorios anteriores.

Listing 1: Superclase Soldado

```
  public class Soldier {
private String name;
private int HP;
private int cHP;
private int row;
private int col;
private boolean status;
private int attack;
private int defense;
private String flag;
private String alias;

public Soldier() {
    this.name = "          ";
    this.flag = "          ";
    this.status = false;
    this.alias = "  ";
}

public Soldier(String name) {
    this.name = name;
    this.status = true;
}

public void setName(String name) {
    this.name = name;
}
```

```
public void setAttack(int attack) {
    this.attack = attack;
}

public void setDefense(int defense) {
    this.defense = defense;
}

public void setHP(int HP) {
    this.HP = HP;
}

public void setCHP(int cHP) {
    this.cHP = cHP;
}

public void setRow(int row) {
    this.row = row;
}

public void setCol(int col) {
    this.col = col;
}

public void setStatus(boolean status) {
    this.status = status;
}

public void setFlag(String flag) {
    this.flag = flag;
}

public void setAlias(String alias) {
    this.alias = alias;
}

public String getAlias() {
    return this.alias;
}

public String getFlag() {
    return this.flag;
}

public String getName() {
    return this.name;
}

public int getAttack() {
    return this.attack;
}

public int getHP() {
    return this.HP;
}
```

```
    public int getcHP() {
        return this.cHP;
    }

    public int getCol() {
        return this.col;
    }

    public int getRow() {
        return this.row;
    }

    public int getDefense() {
        return this.defense;
    }

    public boolean getStatus() {
        return this.status;
    }

    @Override
    public String toString() {
        return "Soldado " + this.name + "\n"
               + "HP: " + this.HP + "/" + this.cHP + "\n"
               + "Posicion: " + this.row + ", " + this.col + "\n";
    }
}
```

- En esta clase Soldier, se encuentran todos sus getters, setters y atributos necesarios para el funcionamiento del videojuego, además de un método toString que nos servirá para más adelante cuando toque imprimir soldados.
- A continuación, empecé a trabajar con la clase Mapa, constituida por el tablero elaborado en laboratorios anteriores, por tanto, decidí reutilizar código para el método de imprimir o mostrar el mapa.
- Pero para poder trabajar con la clase Mapa, necesito que se relacione con los ejércitos, y para ello, decidí crear una nueva clase llamada Army, para poder facilitar el control de los ejércitos.

Listing 2: Clase Army

```
112 public class Army {
113     private List<Soldier> soldiers = new ArrayList<>();
114     private String[] reinos = new String[] {
115             "Inglaterra", "Francia", "Castilla-Aragon", "Moros", "Sacro Imperio Romano
                Germanico" };
116     private String reino;
117     private int nArmy;
118     private int nArchers = 0;
119     private int nKnights = 0;
120     private int nSwords = 0;
121     private int nSpears = 0;
122
123     Random random = new Random();
```

```
124
125     public Army(int n) {
126         this.reino = reinos[random.nextInt(reinos.length)];
127         this.nArmy = n;
128     }
129
130     public void initialize() {
131         String flag = (this.nArmy == 1) ? "############" : "************";
132         for (int i = 0; i < random.nextInt(10) + 1; i++) {
133             switch (random.nextInt(4) + 1) {
134                 case 1:
135                     Archer a = new Archer("Archer" + i + "X" + nArmy);
136                     a.setFlag(flag);
137                     a.setAlias("A" + i + "X" + nArmy);
138                     nArchers++;
139                     soldiers.add(a);
140                     break;
141                 case 2:
142                     Knight k = new Knight("Knight" + i + "X" + nArmy);
143                     k.setFlag(flag);
144                     k.setAlias("K" + i + "X" + nArmy);
145                     nKnights++;
146                     soldiers.add(k);
147                     break;
148                 case 3:
149                     Spearman s = new Spearman("Spearman" + i + "X" + nArmy);
150                     s.setFlag(flag);
151                     s.setAlias("S" + i + "X" + nArmy);
152                     nSpears++;
153                     soldiers.add(s);
154                     break;
155                 case 4:
156                     Swordsman w = new Swordsman("Swordsman" + i + "X" + nArmy);
157                     w.setFlag(flag);
158                     w.setAlias("W" + i + "X" + nArmy);
159                     nSwords++;
160                     soldiers.add(w);
161                     break;
162             }
163         }
164     }
165
166     public void perk() {
167         for (Soldier s : soldiers) {
168             s.setHP(s.getcHP() + 1);
169             s.setCHP(s.getHP());
170         }
171     }
172
173     public int getnArmy() {
174         return this.nArmy;
175     }
176
177     public String getReino() {
178         return this.reino;
179     }
```

```
180
181     public String[] getReinos() {
182         return this.reinos;
183     }
184
185     public void setReino(String reino) {
186         this.reino = reino;
187     }
188
189     public List<Soldier> getSoldiers() {
190         return this.soldiers;
191     }
192
193     public int totalHP() {
194         int total = 0;
195         for (Soldier s : this.soldiers) {
196             total += s.getHP();
197         }
198         return total;
199     }
200
201     public void show() {
202         System.out.println("Ejercito " + this.nArmy);
203         System.out.println("Reino: " + this.reino);
204         System.out.println();
205         for (Soldier s : this.soldiers) {
206             System.out.println(s);
207         }
208         System.out.println();
209     }
210
211     public Soldier getMaxHP() {
212         int max = 0;
213         for (int i = 0; i < this.soldiers.size(); i++) {
214             if (this.soldiers.get(i).getHP() > this.soldiers.get(max).getHP()) {
215                 max = i;
216             }
217         }
218         return this.soldiers.get(max);
219     }
220
221     public void sort() {
222         for (int i = 1; i < this.soldiers.size(); i++) {
223             Soldier s = this.soldiers.get(i);
224             int j = i - 1;
225
226             while (j >= 0 && soldiers.get(j).getHP() < s.getHP()) {
227                 soldiers.set(j + 1, soldiers.get(j));
228                 j--;
229             }
230             soldiers.set(j + 1, s);
231         }
232     }
233
234     public void resume() {
235         System.out.println("Ejercito " + nArmy + ": " + this.reino);
```

```
236         System.out.println("Total de soldados creados: " + this.soldiers.size());
237         System.out.println("Espadachines: " + this.nSwords);
238         System.out.println("Arqueros: " + this.nArchers);
239         System.out.println("Caballeros: " + this.nKnights);
240         System.out.println("Lanceros: " + this.nSpears);
241         System.out.println();
242     }
```

- Esta clase contiene los atributos que normalmente tendría un ejército, como la cantidad de soldados de cada tipo que existen, un método para inicializar todos los soldados de forma aleatoria así como un ArrayList que contenga los Soldados, así como nos lo dice la consigna. Esto con el fin de que el código sea mucho más polimórfico.
- Tiene sus respectivos setters y getters, pero también le coloqué métodos bastante interesantes como el método initialize, que va a inicializar todo el ArrayList y creará soldados de los diferentes tipos, para asignarles su nombre y añadirlos a la Lista.
- También cuenta con un método perk que añade 1 punto de vida a cada soldado, para poder aplicar la condición de que algunos reinos tienen ventajas en cuanto al terreno.
- Asimismo, tambíen le asocié a esta clase los métodos respectivos para obtener la vida total del ejército, para mostrar todos los soldados de un ejército, ordenar el ejército en base a algún algoritmo de ordenamiento (en este caso, usé insertion sort) , mostrar el ejército y todos sus soldados, y por último, realizar el resumen que saldría al final, mostrando su cantidad de soldados de cada tipo.
- Si nos dimos cuenta, hay cuatro clases extra aquí, la clase Archer, Spearman, Swordsman y Knight, clases que se comportan como soldados, esto es porque son cuatro subclases que heredan todos los atributos y métodos de la clase Soldier.

Listing 3: Subclase Archer

```
243 public class Archer extends Soldier {
244     private int nArrows;
245     Random random = new Random();
246 
247     public Archer(String name) {
248         super(name);
249         this.nArrows = random.nextInt(10);
250         this.setAttack(7);
251         this.setDefense(3);
252         this.setHP(random.nextInt(3) + 3);
253         this.setCHP(this.getHP());
254     }
255 
256     public void setnArrows(int nArrows) {
257         this.nArrows = nArrows;
258     }
259 
260     public int getnArrows() {
261         return this.nArrows;
262     }
263 }
```

Listing 4: Subclase Knight

```
264 public class Knight extends Soldier {
265     private String weapon;
266     private boolean isMounted;
267     Random random = new Random();
268 
269     public Knight(String name) {
270         super(name);
271         this.setAttack(13);
272         this.setDefense(7);
273         this.setHP(random.nextInt(3) + 10);
274         this.setCHP(this.getHP());
275     }
276 
277     public void unMount() {
278         this.isMounted = false;
279         this.weapon = "Sword";
280     }
281 
282     public void mount() {
283         this.isMounted = true;
284         this.weapon = "Spear";
285     }
286 
287     public String getWeapon() {
288         return this.weapon;
289     }
290 
291     public boolean isMounted() {
292         return this.isMounted;
293     }
294 }
```

Listing 5: Subclase Swordsman

```
295 public class Spearman extends Soldier {
296     private int spearLong;
297     private static int quantity;
298     Random random = new Random();
299 
300     public Spearman(String name) {
301         super(name);
302         this.spearLong = random.nextInt(10);
303         this.setAttack(5);
304         this.setDefense(8);
305         this.setHP(random.nextInt(3) + 8);
306         this.setCHP(this.getHP());
307         quantity++;
308     }
309 
310     public void setSpearLong(int spearLong) {
311         this.spearLong = spearLong;
312     }
313 
314     public int getSpearLong() {
```

```
315         return this.spearLong;
316     }
317 
318     public void schiltrom() {
319         this.setDefense(this.getDefense() + 1);
320     }
321 
322     public static int getQuantity() {
323         return quantity;
324     }
325 
326 }
```

Listing 6: Subclase Spearman

```
328 public class Swordsman extends Soldier {
329     private int swordLong;
330     Random random = new Random();
331 
332     public Swordsman(String name) {
333         super(name);
334         this.swordLong = random.nextInt(10);
335         this.setAttack(10);
336         this.setDefense(8);
337         this.setHP(random.nextInt(3) + 8);
338         this.setCHP(this.getHP());
339     }
340 
341     public int getSwordLong() {
342         return this.swordLong;
343     }
344 
345     public void setSwordLong(int swordLong) {
346         this.swordLong = swordLong;
347     }
348 
349     public void createWall() {
350         this.setDefense(this.getDefense() + 1);
351     }
352 }
```

- Cada una de estas clases es heredada de la superclase Soldier, teniendo sus atributos y métodos en común y cada clase separada tiene sus particularidades indicadas en las consignas, como el número de flechas de los arqueros, la longitud de espada y lanza para los
- Ahora que ya tenemos esa parte cubierta, hora de trabajar en el mapa ahora si.

Listing 7: Clase Mapa

```
353 public class Mapa {
354     private Soldier[][] table = new Soldier[10][10];
355     private String[] territories = new String[] {
356             "Bosque", "Campo Abierto", "Montana", "Desierto", "Playa" };
357     private String territory;
358     Random random = new Random();
```

```
359
360     public Mapa() {
361         this.territory = territories[random.nextInt(territories.length)];
362     }
363
364     public void initialize() {
365         for (int i = 0; i < table.length; i++) {
366             for (int j = 0; j < table[i].length; j++) {
367                 table[i][j] = new Soldier();
368             }
369         }
370     }
371
372     public void initialize(Army a) {
373         for (Soldier s : a.getSoldiers()) {
374             int x, y;
375             do {
376                 x = random.nextInt(10);
377                 y = random.nextInt(10);
378             } while (table[y][x].getStatus());
379             s.setRow(x);
380             s.setCol(y);
381             table[y][x] = s;
382         }
383         if (search(this.territory, this.territories) == search(a.getReino(), a.getReinos())) {
384             a.perk();
385         }
386     }
387
388     public int search(String a, String[] strs) {
389         int i = 0;
390         for (String s : strs) {
391             if (s.equals(a)) {
392                 return i;
393             }
394             i++;
395         }
396         return -1;
397     }
398
399     public void show() {
400         for (int i = 0; i < table.length; i++) {
401             System.out.println(separadorSup());
402             for (int j = 0; j < table[i].length; j++) {
403                 System.out.print("| " + table[i][j].getFlag() + " |");
404                 if (j == table[i].length - 1) {
405                     System.out.println();
406                 }
407             }
408             for (int j = 0; j < table[i].length; j++) {
409                 System.out.print("|   " + table[i][j].getAlias() + "  |");
410                 if (j == table[i].length - 1) {
411                     System.out.println();
412                 }
413             }
414             for (int j = 0; j < table[i].length; j++) {
```

```
415             if (table[i][j].getStatus()) {
416                 if (table[i][j].getHP() < 10) {
417                     System.out.print("|  " + table[i][j].getcHP() + "/" +
                            table[i][j].getHP() + " HP" + " |");
418                 } else {
419                     System.out.print("| " + table[i][j].getcHP() + "/" +
                            table[i][j].getHP() + " HP" + " |");
420                 }
421             } else {
422                 System.out.print("|           |");
423             }
424             if (j == table[i].length - 1) {
425                 System.out.println();
426             }
427         }
428         System.out.println(separadorInf());
429     }
430     System.out.println();
431 }
432
433 public String separadorSup() {
434     String franky = "";
435     for (int i = 0; i < 10; i++) {
436         franky += "_________________";
437     }
438     franky += "\n";
439     for (int i = 0; i < 10; i++) {
440         franky += "|           |";
441     }
442     return franky;
443 }
444
445 public String separadorInf() {
446     String franky = "";
447     for (int i = 0; i < 10; i++) {
448         franky += "|           |";
449     }
450     franky += "\n";
451     for (int i = 0; i < 10; i++) {
452         franky += "|_______________|";
453     }
454     return franky;
455 }
456
457 public void resume(Army a1, Army a2) {
458     double h1 = 1.0 * a1.totalHP() / (a1.totalHP() + a2.totalHP());
459     double h2 = 1.0 * a2.totalHP() / (a1.totalHP() + a2.totalHP());
460     String ganador;
461
462     double ans = random.nextDouble();
463
464     if (h1 > h2) {
465         if (ans < h2) {
466             ganador = "Ejercito " + a2.getnArmy() + ": " + a2.getReino();
467         } else {
468             ganador = "Ejercito " + a1.getnArmy() + ": " + a1.getReino();
```

```
            }
        } else {
            if (ans < h1) {
                ganador = "Ejercito " + a1.getnArmy() + ": " + a1.getReino();
            } else {
                ganador = "Ejercito " + a2.getnArmy() + ": " + a2.getReino();
            }
        }

        h1 = Math.round(h1 * 100.0);
        h2 = Math.round(h2 * 100.0);

        ans = Math.round(ans * 100.0);

        System.out.println("Ejercito " + a1.getnArmy() + ": " + a1.getReino() + ": " +
            a1.totalHP());
        System.out.println(h1 + "%" + " de probabilidades de victoria");
        System.out.println("Ejercito " + a2.getnArmy() + ": " + a2.getReino() + ": " +
            a2.totalHP());
        System.out.println(h2 + "%" + " de probabilidades de victoria");

        System.out.println("El ganador es: " + ganador);
        System.out.println(
                "Ya que al generar un valor los porcentajes de probabilidad de victoria basada
                    en los niveles\nde vida de sus soldados y aplicando un experimento
                    aleatorio salio vencedor.");
        System.out.println("(Aleatorio generado: " + ans + ")");
    }
}
```

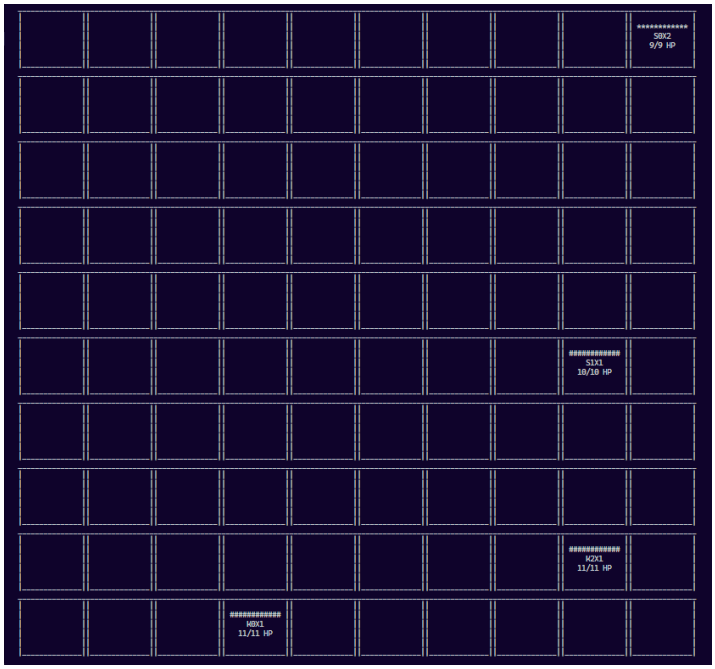
- Esta clase tiene como atributo un tablero constituido por un arreglo bidimensional, además de los territorios. Una vez creado un mapa, se usa el método initialize para inicializar todos los Soldados en el mapa. Para posteriormente inicializar cada ejército en el mapa, usando otro método sobrecargado. Este método toma todos los soldados de un ejército y los va desplegando en el mapa, generando coordenadas que no pueden repetirse (para que solo haya un soldado por casillero) y si se da la relación entre el reino y el terreno, se aumenta la vida a todo el ejército.
- Además, reutilicé el código de laboratorios anteriores para imprimir el tablero, solo que lo hice de una forma mucho más práctica.
- Por último, programé el resumen final, donde se dice cuál de los dos ejércitos es el ganador, basado en probabilidad según la cantidad de vida total por ejército. Eso es todo. Al final, solo me quedaba armar todo en la clase principal.

Listing 8: Clase Videojuego

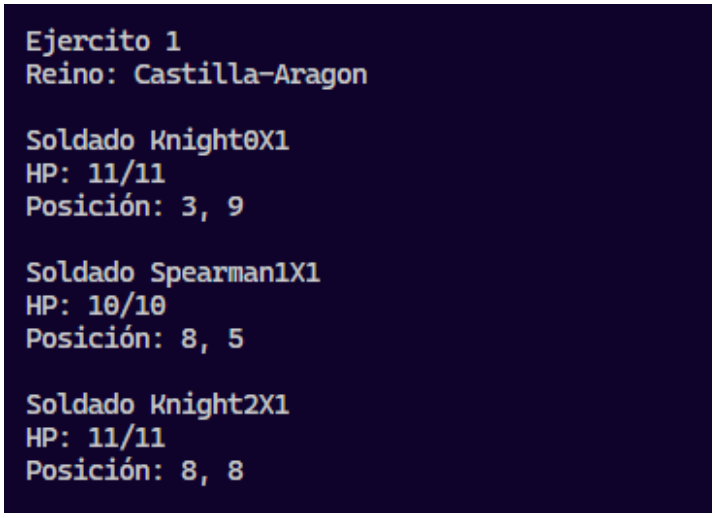
```
   public static void turn(Soldier[][] tb, ArrayList<Soldier> a, int[] c, int id) {
public class VideoJuego {
    public static void main(String[] args) {
        Army ej1 = new Army(1);
        Army ej2 = new Army(2);

        ej1.initialize();
        ej2.initialize();

```

```
504         Mapa mapa = new Mapa();
505         mapa.initialize();
506 
507         mapa.initialize(ej1);
508         mapa.initialize(ej2);
509 
510         mapa.show();
511         ej1.show();
512 
513         ej1.sort();
514         ej1.show();
515 
516         ej1.resume();
517         ej2.resume();
518 
519         mapa.resume(ej1, ej2);
520 
521     }
522 }
```

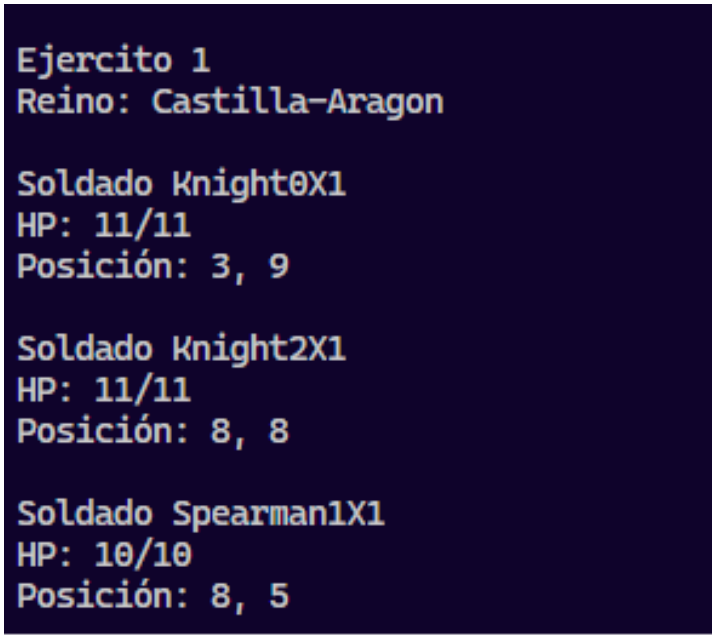
- Aqui pruebo mi código, llamando a métodos de las diferentes clases.
- A continuación colocaré capturas de pantalla de la ejecución del código.



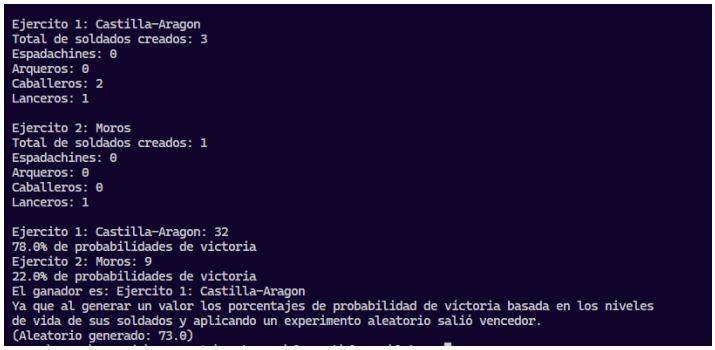
- Aqui se muestra todo el tablero, mostrando a ambos ejércitos.



- Aqui se muestra el ejército 1, junto a todos sus soldados.



- Aqui se muestra el ejército 1, ordenado según vida máxima.



- Y aquí se ve el resumen realizado al final, que muestra que ejército ganó.
- Estos cambios fueron subidos al respositorio, a través de commits.

Listing 9: Commit

```
524   $commit db3bba95fe339bfc7101f02caa29bc5f41d0b103 (HEAD -> main, origin/main)
525 Author: RYAN VALDIVIA <rvaldiviase@unsa.edu.pe>
526 Date:   Mon Jan 15 00:26:25 2024 -0500
527
528     Creando las clases y metodos necesarios para el funcionamiento principal del videojuego
```

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 facil 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 cumplio 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 items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

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

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
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	2	
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 estan 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	4	
Total		20		18	

6. Referencias

- Fundamentos de la programación 2 - Tópicos de la programación Orientada a Objetos (Marco Aedo)