

Informe de Laboratorio 22

Tema: Laboratorio 22

Nota

Estudiante	Escuela	Asignatura
Jhonatan David Arias Quispe jariasq@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de Programación 2 Semestre: II Código: 1701213

Laboratorio	Tema	Duración
22	Laboratorio 22	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 11 Enero 2024	Al 15 Enero 2024

1. Actividades

1. Crear diagrama de clases UML y programa.
2. Crear los miembros de cada clase de la forma más adecuada: como miembros de clase o de instancia.
3. 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.
4. Dibujar el Mapa con las restricciones que solo 1 soldado como máximo en cada cuadrado.
5. El mapa tiene un solo tipo de territorio.
6. 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.
7. En la historia, los ejércitos estaban conformados por diferentes tipos de soldados, que tenían similitudes, pero también particularidades.
8. 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.

9. Los espadachines tienen como atributo particular "longitud de espada" como acción "crear un muro de escudos" que es un tipo de defensa en particular.
10. 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.
11. Los arqueros tienen un número de flechas disponibles las cuales pueden dispararse y se gastan cuando se hace eso.
12. 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).
13. 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.
14. 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).
15. 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).
16. 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).
17. 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).
18. 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.
19. 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.
20. Hacerlo programa iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo ArchCraft GNU Linux 64 bits Kernel
- NeoVim
- OpenJDK 64-Bit 20.0.1

- Git 2.42.0
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Java Swing with JFrame

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/JhonatanDczel/fp2-23b.git`
- URL para el laboratorio 22 en el Repositorio GitHub.
- `https://github.com/JhonatanDczel/fp2-23b/tree/main/fase03/lab22`
- El trabajo de este laboratorio es en su mayor parte lo mismo que en otros laboratorios, por lo que las variaciones serán mínimas

4. Proyecto lab12

- Creamos un directorio en fase03 que contenga los archivos del laboratorio y copiamos los archivos del anterior laboratorio.
- Para el tablero se usará un array bidimensional simple y agregaremos una representación de este en la interfaz gráfica
- La estructura del laboratorio presente es:

```
>>> lab22 git:(main) tree -L 2
.
├── Arquero.java
├── CaballeroFranco.java
├── Caballero.java
├── CaballeroMono.java
├── Ejercito.java
├── EspadachinConquistador.java
├── Espadachin.java
├── EspadachinReal.java
├── EspadachinTeutonico.java
├── Lancero.java
├── latex
│   ├── acts
│   ├── foot
│   ├── head
│   ├── img
│   ├── informe-latex.aux
│   ├── informe-latex.log
│   ├── informe-latex.out
│   ├── informe-latex.pdf
│   ├── informe-latex.tex
│   └── src
├── Mapa.java
├── Soldado.java
├── TableroGUI.java
└── Videojuego.java

7 directories, 19 files
>>> lab22 git:(main)
```

5. Clase Soldado

La clase `Soldado` es una clase abstracta que proporciona las propiedades y métodos básicos que se aplican a todos los soldados en el juego. Algunos de los atributos más relevantes son `nombre`, `nivelAtaque`, `nivelDefensa`, `nivelVida`, `vidaActual`, `velocidad`, `actitud`, `vive`, `fila`, y `columna`. Esta clase también incluye métodos para realizar acciones comunes, como `atacar`, `defender`, `avanzar`, `retroceder`, `serAtacado`, `huir`, y `morir`.

```
1 public abstract class Soldado {
2     // Atributos de la clase Soldado
3
4     // Constructor
5     public Soldado(String ejercito, int i) { ... }
6
7     // Metodo genrico para atacar a otro soldado
8     public void atacar(Soldado enemigo) { ... }
9
10    // Otros mtodos como defender, avanzar, retroceder, serAtacado, huir, morir, entre otros.
11 }
```

Descripción: La clase `Soldado` establece las propiedades y comportamientos básicos que todos los soldados en el juego comparten. Los atributos incluyen información sobre el estado actual del soldado, como su nombre, niveles de ataque, defensa, vida, posición en el campo, entre otros. Además, se definen métodos que permiten a los soldados interactuar entre sí y realizar acciones específicas.

6. Clase `Espadachin`

La clase `Espadachin` extiende la clase base `Soldado` e introduce características adicionales específicas para un tipo de soldado con espada. Se agregan atributos como `longitudEspada` y métodos como `crearMuroEscudo` que no están presentes en la clase base.

```
1 class Espadachin extends Soldado {
2     // Atributos adicionales para Espadachin
3
4     // Constructor específico para Espadachin
5     public Espadachin(String ejercito, int i) { ... }
6
7     // Método específico para Espadachin
8     public void crearMuroEscudo() { ... }
9 }
```

Descripción: La clase `Espadachin` hereda las propiedades y métodos de la clase `Soldado` pero agrega características específicas relacionadas con ser un espadachín. Se introduce el atributo `longitudEspada` y el método `crearMuroEscudo`, que reflejan la especialización de este tipo de soldado en el juego.

7. Clase `EspadachinReal`

La clase `EspadachinReal` también extiende la clase base `Soldado` y presenta atributos y métodos únicos, como `lanzarCuchillo` y `aumentarNivel`.

```
1 class EspadachinReal extends Soldado {
2     // Atributos y métodos específicos para EspadachinReal
3
4     // Constructor específico para EspadachinReal
5     public EspadachinReal(String ejercito, int i) { ... }
6
7     // Método específico para EspadachinReal
8     public void lanzarCuchillo() { ... }
9     public void aumentarNivel() { ... }
10 }
```

Descripción: La clase `EspadachinReal` hereda de `Soldado` y añade funcionalidades particulares, como la capacidad de lanzar cuchillos (`lanzarCuchillo`) y aumentar su nivel (`aumentarNivel`). Estas adiciones hacen que esta clase sea distintiva en comparación con otras clases de soldados.

8. Clase `EspadachinTeutonico`

La clase `EspadachinTeutonico` extiende la clase `Espadachin` y agrega atributos y métodos específicos como `lanzarJabalina` y `aumentarNivel`.

```
1 class EspadachinTeutonico extends Espadachin {
2     // Atributos y métodos específicos para EspadachinTeutonico
3
4     // Constructor específico para EspadachinTeutonico
5     public EspadachinTeutonico(String ejercito, int i) { ... }
```

```
6
7 // Metodo especifico para EspadachinTeutonico
8 public void lanzarJabalina() { ... }
9 public void aumentarNivel() { ... }
10 }
```

Descripción: La clase `EspadachinTeutonico` hereda de `Espadachin` y añade características particulares, como la capacidad de lanzar jabalinas (`lanzarJabalina`) y la posibilidad de aumentar su nivel (`aumentarNivel`). Estas adiciones la distinguen dentro de la jerarquía de clases.

9. Clase `EspadachinConquistador`

La clase `EspadachinConquistador` también extiende la clase `Espadachin` y presenta atributos y métodos únicos, como `lanzarHacha` y `aumentarNivel`.

```
1 class EspadachinConquistador extends Espadachin {
2     // Atributos y mtodos especificos para EspadachinConquistador
3
4     // Constructor especifico para EspadachinConquistador
5     public EspadachinConquistador(String ejercito, int i) { ... }
6
7     // Metodo especifico para EspadachinConquistador
8     public void lanzarHacha() { ... }
9     public void aumentarNivel() { ... }
10 }
```

Descripción: La clase `EspadachinConquistador` hereda de `Espadachin` y añade funcionalidades particulares, como la capacidad de lanzar hachas (`lanzarHacha`) y la posibilidad de aumentar su nivel (`aumentarNivel`). Estas adiciones la hacen única en el contexto del juego.

10. Clase `Arquero`

La clase `Arquero` extiende la clase `Soldado` e introduce atributos y métodos específicos como flechas y `dispararFlechas`.

```
1 class Arquero extends Soldado {
2     // Atributos y mtodos especificos para Arquero
3
4     // Constructor especifico para Arquero
5     public Arquero(String ejercito, int i) { ... }
6     public Arquero() { ... }
7
8     // Metodo especifico para Arquero
9     public void dispararFlechas() { ... }
10 }
```

Descripción: La clase `Arquero` hereda de `Soldado` y añade características únicas, como el atributo flechas y el método `dispararFlechas`. Estos elementos reflejan las habilidades específicas de un arquero en el juego.

11. Clase `Caballero`

La clase `Caballero` extiende la clase `Soldado` e introduce atributos y métodos adicionales, como `montado`, `armaActual`, `alternarArma`, `desmontar`, y `montar`.

Código Relevante:

```
1 class Caballero extends Soldado {  
2     // Atributos y mtodos especificos para Caballero  
3  
4     // Constructor especifico para Caballero  
5     public Caballero(String ejercito, int i) { ... }  
6  
7     // Mtodos especificos para Caballero  
8     public void alternarArma() { ... }  
9     public void desmontar() { ... }  
10    public void montar() { ... }  
11    public void investir() { ... }  
12 }
```

Descripción:

La clase `Caballero` hereda de `Soldado` y añade funcionalidades particulares, como la capacidad de estar montado, cambiar de armaActual con `alternarArma`, y realizar acciones específicas como `desmontar`, `montar`, y `investir`.

12. Clase CaballeroFranco

La clase `CaballeroFranco` extiende la clase `Caballero` e introduce atributos y métodos adicionales, como `lanzarLanzas` y `aumentarNivel`.

Código Relevante:

```
1 class CaballeroFranco extends Caballero {  
2     // Atributos y mtodos especificos para CaballeroFranco  
3  
4     // Constructor especifico para CaballeroFranco  
5     public CaballeroFranco(String ejercito, int i) { ... }  
6  
7     // Mtodos especificos para CaballeroFranco  
8     public void lanzarLanzas() { ... }  
9     public void aumentarNivel() { ... }  
10 }
```

Descripción:

La clase `CaballeroFranco` hereda de `Caballero` y añade características particulares, como la capacidad de `lanzarLanzas` y la posibilidad de `aumentarNivel`. Estas adiciones reflejan las habilidades únicas de este tipo de caballero en el juego.

13. Clase CaballeroMoro

La clase `CaballeroMoro` extiende la clase `Caballero` e introduce atributos y métodos adicionales, como `lanzarFlechas` y `aumentarNivel`.

Código Relevante:

```
1 class CaballeroMoro extends Caballero {
2     // Atributos y mtodos específicos para CaballeroMoro
3
4     // Constructor específico para CaballeroMoro
5     public CaballeroMoro(String ejercito, int i) { ... }
6
7     // Mtodos específicos para CaballeroMoro
8     public void lanzarFlechas() { ... }
9     public void aumentarNivel() { ... }
10 }
```

Descripción:

La clase `CaballeroMoro` hereda de `Caballero` y añade funcionalidades particulares, como la capacidad de `lanzarFlechas` y la posibilidad de `aumentarNivel`. Estas adiciones la hacen única en el contexto del juego.

14. Clase Lancero

La clase `Lancero` extiende la clase `Soldado` e introduce atributos y métodos adicionales, como `longitudDeLanza` y `schiltrom`.

Código Relevante:

```
1 class Lancero extends Soldado {
2     // Atributos y mtodos específicos para Lancero
3
4     // Constructor específico para Lancero
5     public Lancero(String ejercito, int i) { ... }
6
7     // Metodo específico para Lancero
8     public void schiltrom() { ... }
9 }
```

Descripción:

La clase `Lancero` hereda de `Soldado` y añade características particulares, como el atributo `longitudDeLanza` y el método `schiltrom`. Estos elementos reflejan las habilidades únicas de un lancero en el juego.

15. Clase Mapa

Ahora, analizaremos la clase `Mapa` que representa el tablero de juego donde se ubican los ejércitos y soldados. Esta clase es esencial para la interacción y el desarrollo del juego, ya que gestiona la disposición de los ejércitos en el terreno y realiza acciones asociadas a esta ubicación.

15.1. Atributos

15.1.1. terreno


```
1 private String terreno;
```

El atributo `terreno` almacena el tipo de terreno del mapa, como "bosque", "campo abierto", "playa", "montaña", "desierto". Este atributo se inicializa en el constructor de la clase.

15.1.2. MAX_SOLDADOS

```
1 private static final int MAX_SOLDADOS = 10;
```

El atributo `MAX_SOLDADOS` establece el tamaño máximo del tablero en ambas dimensiones, limitando el número de filas y columnas.

15.1.3. tableroReinos

```
1 private Ejercito[] [] tableroReinos = new Ejercito[MAX_SOLDADOS][MAX_SOLDADOS];
```

El atributo `tableroReinos` es una matriz bidimensional de tipo `Ejercito` que representa el tablero donde se ubicarán los ejércitos. Esta matriz tiene dimensiones definidas por `MAX_SOLDADOS` y se inicializa en el constructor.

15.2. Métodos

15.2.1. Constructor

```
1 public Mapa(int i)
```

El constructor inicializa el atributo `terreno` con un valor correspondiente al índice proporcionado en el arreglo `terreno`. Este índice se utiliza para seleccionar el tipo de terreno del mapa.

15.2.2. tablero

```
1 public Ejercito[] [] tablero()
```

El método `tablero` devuelve la matriz `tableroReinos`, permitiendo acceder a la configuración actual del tablero.

15.2.3. agregarEjercito

```
1 public void agregarEjercito(int fila, int columna, Ejercito ejercito, int totalSoldados)
```

El método `agregarEjercito` coloca un ejército en una posición específica del tablero. Además, ajusta la vida de los soldados del ejército en función de su reino y del tipo de terreno en el que se encuentran.

15.2.4. mostrarTableroEjercitos

```
1 public void mostrarTableroEjercitos()
```

El método `mostrarTableroEjercitos` imprime en la consola el estado actual del tablero, mostrando los nombres de los ejércitos y la vida total de cada ejército en el tablero.

15.2.5. mostrarTableroSoldados

```
1 public void mostrarTableroSoldados(Soldado[] [] tablero)
```

El método `mostrarTableroSoldados` imprime en la consola el estado actual del tablero de soldados. Muestra información detallada sobre cada soldado, incluyendo su nombre y código.

15.3. Funcionamiento Detallado

15.3.1. Método agregarEjercito

```
1 public void agregarEjercito(int fila, int columna, Ejercito ejercito, int totalSoldados)
```

Este método coloca un ejército en una posición específica del tablero y ajusta la vida de los soldados del ejército en función de su reino y del tipo de terreno en el que se encuentran. Se realiza una verificación de la correspondencia entre el reino del ejército y el terreno, incrementando la vida de los soldados en 1 si la condición se cumple.

```
1 if (ejercito.getReino().equals(nombresReinos[0]) && this.terreno.equals(terreno[0]))  
2     for (Soldado s : ejercito.iterar())  
3         s.actualizarVida(1);
```

Este fragmento de código representa la verificación para el reino `Inglaterra` y el terreno `"bosque"`. Si se cumple la condición, se itera sobre los soldados del ejército y se incrementa su vida en 1.

15.3.2. Método mostrarTableroEjercitos

```
1 public void mostrarTableroEjercitos()
```

Este método imprime en la consola el estado actual del tablero de ejércitos, mostrando los nombres de los ejércitos y la vida total de cada ejército en el tablero.

```
1 System.out.print("|" + columna.getNombreEjercito() + "-" + vida);
```

Este fragmento de código imprime el nombre del ejército y la vida total en el tablero. Se utiliza formato para asegurar que la vida se muestre con dos dígitos.

15.3.3. Método mostrarTableroSoldados

```
1 public void mostrarTableroSoldados(Soldado[] [] tablero)
```

Este método imprime en la consola el estado actual del tablero de soldados, mostrando información detallada sobre cada soldado, incluyendo su nombre y código.

```
1 System.out.printf("| %s-%4s", nombre.substring(nombre.length()-2, nombre.length()-1),  
    columna.getNombreCode());
```

Este fragmento de código muestra información detallada sobre cada soldado, incluyendo el nombre abreviado y el código del soldado. La abreviatura del nombre se toma de los dos últimos caracteres del nombre completo del soldado.

16. Clase Videojuego

La clase Videojuego representa la lógica principal del juego. A continuación, se realiza un análisis detallado de su estructura y funcionamiento.

16.1. Atributos Estáticos

```
1 private static final int MAX_SOLDADOS=10;  
2 private static final String SIMBOL_EJERCITO1="Z";  
3 private static final String SIMBOL_EJERCITO2="X";
```

- $MAX_SOLDADOS$: Define el tamaño máximo del tablero y se utiliza para la creación de matrices y listas. $SIMBOL_EJERCITO1$

16.2. Método Principal main

```
1 public static void main(String[] args)
```

Este método inicia la ejecución del juego. A continuación, se describen los pasos principales realizados:

1. Selección de Reinos:

```
1 String[] nombresReinos={"Inglaterra", "Francia", "Sacro Imperio", "Castilla-Aragon", "Moros"};  
2 System.out.println("Jugador 1 y 2, escojan un reino(diferentes):");  
3 int select1=sc.nextInt()-1;  
4 int select2=sc.nextInt()-1;  
5 String reinoJugador1=nombresReinos[select1];  
6 String reinoJugador2=nombresReinos[select2];  
7 Mapa mapa=new Mapa(random.nextInt(4));
```

2. Creación de Ejércitos y Soldados:

```
1 Soldado[][] tablero=new Soldado[MAX_SOLDADOS][MAX_SOLDADOS];  
2 ArrayList<Ejercito> reino1=crearReino(mapa, SIMBOL_EJERCITO1, reinoJugador1);  
3 ArrayList<Ejercito> reino2=crearReino(mapa, SIMBOL_EJERCITO2, reinoJugador2);  
4 Soldado s1=reino1.get(0).soldadoMasFuerte();  
5 Soldado s2=reino2.get(0).soldadoMasFuerte();
```

3. Preparación del Tablero y Visualización:

```
1 preTablero(reino1.get(0), tablero);  
2 preTablero(reino2.get(0), tablero);  
3 mapa.mostrarTableroSoldados(tablero);  
4 reino1.get(0).mostrarSoldados();  
5 reino2.get(0).mostrarSoldados();
```

4. Visualización Gráfica con Swing:

```
1 SwingUtilities.invokeLater(() -> {  
2     new TableroGUI(tablero);  
3 });
```

5. Información Detallada de los Ejércitos:

```
1 System.out.println("\n-----Ejercito Nro1(Z)-----");
2 // ... (información detallada sobre el Ejercito 1)
3
4 System.out.println("\n-----Ejercito Nro2(X)-----");
5 // ... (información detallada sobre el Ejercito 2)
```

6. Resumen y Determinación del Ganador:

```
1 reino1.get(0).resumenEjercito(reinoJugador1);
2 reino2.get(0).resumenEjercito(reinoJugador2);
3 determinarGanador(reino1.get(0), reino2.get(0), reinoJugador1, reinoJugador2);
```

7. Pregunta para Continuar Jugando:

```
1 System.out.println("Quieres seguir jugando?(true/false) ");
2 boolean seguirJugando=sc.nextBoolean();
3 if(seguirJugando==false)
4     jugar=false;
```

16.3. Métodos Adicionales

16.3.1. crearReino

```
1 public static ArrayList<Ejercito> crearReino(Mapa mapa, String n, String reinoN)
```

Este método crea un ejército y lo coloca en una posición aleatoria del tablero. También agrega el ejército al mapa.

16.3.2. preTablero

```
1 public static void preTablero(Ejercito ejercito, Soldado[][] tablero)
```

Este método coloca soldados en posiciones aleatorias del tablero antes de iniciar el juego.

16.3.3. eliminarSoldado

```
1 public static void eliminarSoldado(Soldado eliminar, ArrayList<Soldado> soldados)
```

Este método elimina un soldado de la lista de soldados.

16.3.4. promedioVida

```
1 public static double promedioVida(Ejercito ejercito)
```

Calcula y devuelve el promedio de vida de los soldados en un ejército.

16.3.5. determinarGanador

```
1 private static void determinarGanador(Ejercito ejercito1, Ejercito ejercito2, String reinoJugador1,
    String reinoJugador2)
```

Determina el ganador basándose en probabilidades aleatorias y la vida total de los ejércitos.

16.3.6. realizarAccion

```
1 public static void realizarAccion(Soldado[] [] tablero, Soldado soldado, ArrayList<Soldado> ejercito,  
    ArrayList<Soldado> ejercitoE, Scanner sc)
```

Realiza una acción durante el juego, como encontrar y enfrentar a un enemigo en el tablero.

17. Informe sobre la Clase TableroGUI

La clase `TableroGUI` se encarga de crear una interfaz gráfica para visualizar el tablero de soldados en el juego. A continuación, se presenta un análisis detallado de su estructura y funcionalidades, así como ejemplos de cómo interactúa con otras clases.

17.1. Características Principales

17.1.1. Herencia de JFrame

```
1 public class TableroGUI extends JFrame
```

La clase `TableroGUI` hereda de `JFrame`, la cual es una clase que proporciona una ventana contenedora para la interfaz gráfica.

17.1.2. Atributos

```
1 private Soldado[] [] tablero;
```

`tablero`: Almacena la información del tablero de soldados que se mostrará en la interfaz.

17.1.3. Constructor

```
1 public TableroGUI(Soldado[] [] tablero)
```

Este constructor recibe una matriz de soldados (`tablero`) como parámetro y configura la interfaz. Establece el título, tamaño de la ventana, operación de cierre y visibilidad. Crea un objeto `TableroPanel` (clase interna) y lo agrega a la ventana.

17.1.4. Clase Interna TableroPanel

```
1 private class TableroPanel extends JPanel
```

Esta clase interna hereda de `JPanel` y se encarga de dibujar el tablero en la interfaz.

17.1.5. Método paintComponent

```
1 @Override  
2 protected void paintComponent(Graphics g)
```

Este método sobrescrito se encarga de dibujar los elementos en el panel. Itera sobre la matriz de soldados y pinta las celdas del tablero. Utiliza colores diferentes para representar a los dos ejércitos.

17.1.6. Ejemplo de Uso en la Clase Videojuego

```
1 SwingUtilities.invokeLater(() -> {  
2     new TableroGUI(tablero);  
3 });
```

Cuando se crea un objeto `TableroGUI` en la clase `Videojuego`, se pasa la matriz de soldados (`tablero`) como parámetro. Esto muestra la interfaz gráfica del tablero.

17.2. Interacción con Otras Clases

17.2.1. Uso del Tablero en Videojuego

```
1 SwingUtilities.invokeLater(() -> {  
2     new TableroGUI(tablero);  
3 });
```

La clase `Videojuego` utiliza `TableroGUI` para mostrar gráficamente el estado del tablero de soldados en el juego. La llamada a `SwingUtilities.invokeLater` asegura que la interfaz se actualice de manera segura.

17.2.2. Acceso a la Información de los Soldados

```
1 Soldado soldado = tablero[fila][columna];
```

Dentro del método `paintComponent`, se accede a la información de cada soldado en el tablero para determinar su posición y tipo. Esto permite representar visualmente la ubicación y afiliación de cada soldado en el tablero.

17.2.3. Ejemplo de Colores según Afiliación

```
1 if ((nombre.substring(nombre.length()-2, nombre.length()-1).equals("Z"))) {  
2     g.setColor(Color.CYAN);  
3 } else {  
4     g.setColor(Color.ORANGE);  
5 }
```

El código verifica el símbolo del soldado para determinar a qué ejército pertenece y utiliza colores diferentes (cian u naranja) para representarlos en la interfaz.

18. Ejecución del juego

18.1. Configuración Inicial

El juego comienza con la bienvenida y la solicitud para que los Jugadores 1 y 2 elijan un reino diferente. En este ejemplo, el Jugador 2 selecciona el reino de Francia y la posición 4 en el tablero.

```
1 Jugador 1 y 2, escojan un reino(diferentes):  
2 (1)Inglaterra  
3 (2)Francia  
4 (3)Sacro Imperio  
5 (4)Castilla-Aragon  
6 (5)Moros  
7 2
```

8 4

18.2. Estado Inicial del Tablero

Se presenta el tablero de soldados con las posiciones iniciales de las unidades.

```
1 Tablero de soldados
2 -----
3 |-----|-----|-----|-----|-----|X-C11|-----|-----|
4 |-----|-----|-----|-----|-----|X-EC14|-----|Z-CF15|-----|
5 |-----|-----|-----|-----|-----|-----|-----|-----|
6 |-----|-----|-----|-----|-----|-----|-----|-----|
7 |-----|-----|-----|-----|-----|-----|-----|Z-C10|-----|
8 |X-C11|-----|-----|-----|X-L6|-----|-----|-----|-----|
9 |-----|-----|-----|-----|-----|-----|-----|-----|
10 |Z-L6|-----|-----|-----|-----|-----|-----|-----|-----|
11 |-----|-----|-----|-----|-----|-----|-----|-----|X-E8|
12 |-----|-----|-----|-----|-----|-----|X-L5|-----|-----|
```

18.3. Información Detallada de los Soldados

A continuación, se presentan los datos detallados de cada soldado en los ejércitos Z y X.

```
1 -----Datos de los soldados Z0-----
2 CaballeroZ0      (Z-C10 ) | Vida:10 | Fila:4 | Columna:7 | Atq:13 | Def:7
3 LanceroZ0       (Z-L6 ) | Vida:6 | Fila:7 | Columna:0 | Atq:5 | Def:10
4 CaballeroFrancoZ0 (Z-CF15) | Vida:15 | Fila:1 | Columna:7 | Atq:13 | Def:7

1 -----Datos de los soldados X0-----
2 EspadachinX0    (X-E8 ) | Vida:8 | Fila:8 | Columna:9 | Atq:10 | Def:8
3 CaballeroX0     (X-C11 ) | Vida:11 | Fila:0 | Columna:5 | Atq:13 | Def:7
4 CaballeroX1     (X-C11 ) | Vida:11 | Fila:5 | Columna:0 | Atq:13 | Def:7
5 LanceroX0       (X-L6 ) | Vida:6 | Fila:5 | Columna:4 | Atq:5 | Def:10
6 LanceroX1       (X-L5 ) | Vida:5 | Fila:9 | Columna:6 | Atq:5 | Def:10
7 EspadachinConquistaX0(X-EC14) | Vida:14 | Fila:1 | Columna:5 | Atq:10 | Def:8
```

18.4. Ejército Nro1 (Z) - Estadísticas y Ranking

Se presentan las estadísticas del Ejército Nro1 (Z), incluyendo el soldado con mayor nivel de vida y el ranking de poder.

```
1 -----Ejercito Nro1(Z)-----
2 Soldado con mayor nivel de vida:
3 CaballeroFrancoZ0 | Vida:15 | Fila:1 | Columna:7 | Atq:13 | Def:7
4 Promedio de nivel de vida: 10.33
```

```
1 RANKING DE PODER
2 -----Datos de los soldados Z0-----
3 CaballeroFrancoZ0 (Z-CF15) | Vida:15 | Fila:1 | Columna:7 | Atq:13 | Def:7
4 CaballeroZ0       (Z-C10 ) | Vida:10 | Fila:4 | Columna:7 | Atq:13 | Def:7
5 LanceroZ0         (Z-L6 ) | Vida:6 | Fila:7 | Columna:0 | Atq:5 | Def:10
```

18.5. Ejército Nro2 (X) - Estadísticas y Ranking

Se presentan las estadísticas del Ejército Nro2 (X), incluyendo el soldado con mayor nivel de vida y el ranking de poder.

```
1 -----Ejercito Nro2(X)-----
2 Soldado con mayor nivel de vida:
3 EspadachinConquistaX0 | Vida:14 | Fila:1 | Columna:5 | Atq:10 | Def:8
4 Promedio de nivel de vida: 9.17
```

```
1 RANKING DE PODER
2 -----Datos de los soldados X0-----
3 EspadachinConquistaX0(X-EC14) | Vida:14 | Fila:1 | Columna:5 | Atq:10 | Def:8
4 CaballeroX0 (X-C11 ) | Vida:11 | Fila:0 | Columna:5 | Atq:13 | Def:7
5 CaballeroX1 (X-C11 ) | Vida:11 | Fila:5 | Columna:0 | Atq:13 | Def:7
6 EspadachinX0 (X-E8 ) | Vida:8 | Fila:8 | Columna:9 | Atq:10 | Def:8
7 LanceroX0 (X-L6 ) |
8 Vida:6 | Fila:5 | Columna:4 | Atq:5 | Def:10
9 LanceroX1 (X-L5 ) | Vida:5 | Fila:9 | Columna:6 | Atq:5 | Def:10
```

18.6. Resumen del Juego

Se proporciona un resumen de los ejércitos creados y se evalúa el ejército con mayor probabilidad de victoria.

```
1 Ejercito1: Francia
2 Cantidad total de soldados creados: 3
3 Espadachines: 0
4 Arqueros: 0
5 Caballeros: 2
6 Lanceros: 1
```

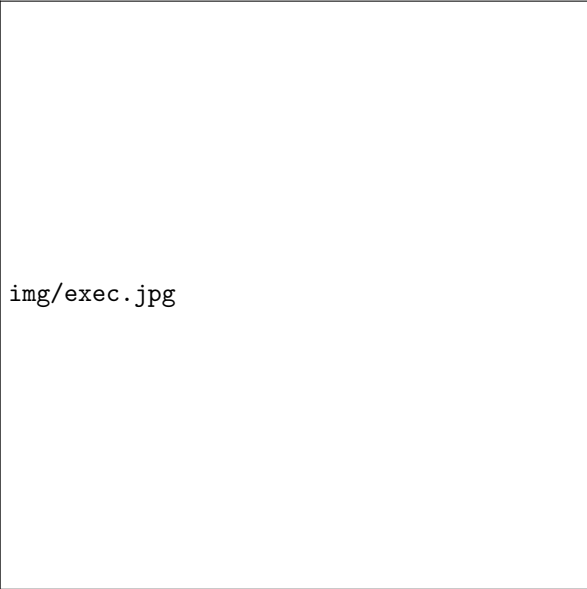
```
1 Ejercito1: Castilla-Aragon
2 Cantidad total de soldados creados: 6
3 Espadachines: 2
4 Arqueros: 0
5 Caballeros: 2
6 Lanceros: 2
```

```
1 ---Metrica: Ejercito con mas soldados---
2 Ejercito 1: Francia : 31 36.05% de probabilidad de victoria
3 Ejercito 2: Castilla-Aragon: 55 63.95% de probabilidad de victoria
4
5 El ganador es el . Ya que al generar los
6 porcentajes de probabilidad de victoria basada en los niveles de
7 vida de sus soldados y aplicando un experimento aleatorio sali
8 vencedor. (Aleatorio generado: 56.72%)
9Quieres seguir jugando?(true/false)
```

El juego "SHOOTGAME" se ejecutó exitosamente, proporcionando información detallada sobre la configuración inicial, el estado de los soldados y el análisis estratégico de los ejércitos. La probabilidad de victoria se determina según la vida de los soldados, y en este caso, el ejército Castilla-Aragon se coronó como el ganador.

18.7. Representación gráfica

- La representación visual está hecha con Java Swing y es la siguiente:



img/exec.jpg

19. Commits más recientes

- A continuación se presentan los commits más recientes.
- Los commits siguen la convención para commits de git y utilizan mensajes de forma imperativa.

Commit ad84b4904ddf4aee0af2a04163b3d8f1ead240f0

Autor: JhonatanDczel

Descripción: Método principal y atributos auxiliares.

Commit 8c10c5612c16e864bae9a9761a0da6cb42895a63

Autor: JhonatanDczel

Descripción: Método para eliminar soldado y determinar ganador.

Commit 2e327f798d6bf46aa59902a474ac0915b12418f1

Autor: JhonatanDczel

Descripción: Crea la clase videojuego.

Commit 6f576bd0bede4dbc1409cfa8c3ac46a285da00b8

Autor: JhonatanDczel

Descripción: Agrega la clase TableroGUI.

Commit 23af3f6bd08a8c715567d8ce669d873a62b19927

Autor: JhonatanDczel

Descripción: Agrega la clase Mapa.

Commit 0f3633aaeb29bc25de4523906ceecfcd0dc784fe

Autor: JhonatanDczel

Descripción: Agrega la clase lancero.

Commit 98e80c7a7793e2f4463a3d03127fc51066a7e13c

Autor: JhonatanDczel

Descripción: Sube las variedades de caballero.

Commit 95a2890d35a7c9d3fc66a8c331c689b770ea7dd1

Autor: JhonatanDczel

Descripción: Sube la clase Arquero.

Commit 7250028e9af3f271eff8292a5ee0f9ddf68f031d

Autor: JhonatanDczel

Descripción: Crea la clase Espadachín.

Commit 5abec28851700553b7f0783f268109db72e53bfe

Autor: JhonatanDczel

Descripción: Crea la clase Soldado.