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 Programacion 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 espadaz como acción çrear 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 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.

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 lanzaz 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 Kernell
- 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 seran minimas

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 representacion 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`.

```java
public abstract class Soldado {
  // Atributos de la clase Soldado

  // Constructor
  public Soldado(String ejercito, int i) { ... }

  // Mtodo  genrico  para atacar a otro soldado
  public void atacar(Soldado enemigo) { ... }

  // Otros mtodos como defender, avanzar, retroceder, serAtacado, huir, morir, entre otros.
}
```

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.

```
class Espadachin extends Soldado {
  // Atributos adicionales para Espadachin

  // Constructor especfico para Espadachin
  public Espadachin(String ejercito, int i) { ... }

  // Mtodo especfico para Espadachin
  public void crearMuroEscudo() { ... }
}
```

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`.

```
class EspadachinReal extends Soldado {
  // Atributos y mtodos especficos para EspadachinReal

  // Constructor especfico para EspadachinReal
  public EspadachinReal(String ejercito, int i) { ... }

  // Mtodo especfico para EspadachinReal
  public void lanzarCuchillo() { ... }
  public void aumentarNivel() { ... }
}
```

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`.

```
class EspadachinTeutonico extends Espadachin {
  // Atributos y mtodos especficos para EspadachinTeutonico

  // Constructor especfico para EspadachinTeutonico
  public EspadachinTeutonico(String ejercito, int i) { ... }
```

```
6
7   // Mtodo especfico 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 especficos para EspadachinConquistador
3
4    // Constructor especfico para EspadachinConquistador
5    public EspadachinConquistador(String ejercito, int i) { ... }
6
7    // Mtodo especfico 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 especficos para Arquero
3
4    // Constructor especfico para Arquero
5    public Arquero(String ejercito, int i) { ... }
6    public Arquero() { ... }
7
8    // Mtodo especfico 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:

```
class Caballero extends Soldado {
  // Atributos y mtodos especficos para Caballero

  // Constructor especfico para Caballero
  public Caballero(String ejercito, int i) { ... }

  // Mtodos especficos para Caballero
  public void alternarArma() { ... }
  public void desmontar() { ... }
  public void montar() { ... }
  public void envestir() { ... }
}
```

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 `envestir`.

—

12. Clase CaballeroFranco

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

Código Relevante:

```
class CaballeroFranco extends Caballero {
  // Atributos y mtodos especficos para CaballeroFranco

  // Constructor especfico para CaballeroFranco
  public CaballeroFranco(String ejercito, int i) { ... }

  // Mtodos especficos para CaballeroFranco
  public void lanzarLanzas() { ... }
  public void aumentarNivel() { ... }
}
```

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:

```
class CaballeroMoro extends Caballero {
  // Atributos y mtodos especficos para CaballeroMoro

  // Constructor especfico para CaballeroMoro
  public CaballeroMoro(String ejercito, int i) { ... }

  // Mtodos especficos para CaballeroMoro
  public void lanzarFechas() { ... }
  public void aumentarNivel() { ... }
}
```

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:

```
class Lancero extends Soldado {
  // Atributos y mtodos especficos para Lancero

  // Constructor especfico para Lancero
  public Lancero(String ejercito, int i) { ... }

  // Mtodo especfico para Lancero
  public void schiltrom() { ... }
}
```

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`

```
private String terreno;
```

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

15.1.2. MAX_SOLDADOS

```
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

```
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

```
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

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

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

15.2.3. agregarEjercito

```
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

```
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`

```
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`

```
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.

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

Este fragmento de código representa la verificación para el reino Ïnglaterraz 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`

```
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.

```
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`

```
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.

```
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

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

- $\texttt{MAX}_SOLDADOS$: $Defineeltamañomáximodeltableroyseutilizaparalacreacióndematricesylistas.\texttt{SIMBOL}_EJERCITO1y$

16.2. Método Principal `main`

```
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:**

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

2. **Creación de Ejércitos y Soldados:**

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

3. **Preparación del Tablero y Visualización:**

```
preTablero(reino1.get(0), tablero);
preTablero(reino2.get(0), tablero);
mapa.mostrarTableroSoldados(tablero);
reino1.get(0).mostrarSoldados();
reino2.get(0).mostrarSoldados();
```

4. **Visualización Gráfica con Swing:**

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

5. **Información Detallada de los Ejércitos:**

```
System.out.println("\n-----Ejercito Nro1(Z)----");
// ... (informacin detallada sobre el Ejrcito 1)

System.out.println("\n-----Ejercito Nro2(X)----");
// ... (informacin detallada sobre el Ejrcito 2)
```

6. **Resumen y Determinación del Ganador:**

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

7. **Pregunta para Continuar Jugando:**

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

16.3. Métodos Adicionales

16.3.1. `crearReino`

```
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`

```
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`

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

Este método elimina un soldado de la lista de soldados.

16.3.4. `promedioVida`

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

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

16.3.5. `determinarGanador`

```
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`

```
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.

16.4. Conclusiones

La clase `Videojuego` organiza la lógica central del juego, abordando aspectos como la creación

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`

```
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

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

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

17.1.3. Constructor

```
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`

```
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`

```
@Override
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`

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

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`

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

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

```
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

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

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.

17.3. Conclusiones

La clase `TableroGUI` proporciona una interfaz gráfica para visualizar el estado del tablero de soldados en el juego. Su diseño modular y capacidad para interactuar con la información de otras clases facilita la representación visual de la dinámica del juego. Además, el uso de colores y la representación

de símbolos permiten al jugador identificar fácilmente a qué ejército pertenece cada soldado en el tablero.

Ejecución

Estado Inicial del Tablero:

```
    A     B     C     D     E     F     G     H     I     J
  ------------------------------------------------------------------------
1 |     |     |     |     |     |     |     |     |     |     |
  ------------------------------------------------------------------------
2 |     |     |     |     |     |     | #/22 |     |     |     |
  ------------------------------------------------------------------------
3 |     |     |     |     | */6 | */14 | */22 |     |     |     |
  ------------------------------------------------------------------------
4 |     |     |     |     |     |     |     | #/12 |     |     |
  ------------------------------------------------------------------------
5 |     | */30 |     |     |     |     |     |     | */30 | */6 |
  ------------------------------------------------------------------------
6 |     |     |     |     | #/15 |     |     |     |     |     |
  ------------------------------------------------------------------------
7 |     |     |     |     |     | #/6 |     | #/3 |     |     |
  ------------------------------------------------------------------------
8 |     | */6 |     |     | #/10 |     |     |     |     |     |
  ------------------------------------------------------------------------
9 |     | */18 |     |     | #/26 |     |     |     |     |     |
  ------------------------------------------------------------------------
10| #/4 |     | */23 |     |     |     |     | #/15 | #/22 |     |
  ------------------------------------------------------------------------
```

Descripción del Movimiento: Ficha a mover: EspadachinX0 (ubicado en G,3).
Equipo: *
Nueva posición deseada: G,2.

Estado del Tablero Después del Movimiento:

```
    A      B      C      D      E      F      G      H      I      J
  ------------------------------------------------------------------------------
1 |      |      |      |      |      |      |      |      |      |      |
  ------------------------------------------------------------------------------
2 |      |      | */E/4 |      |      |      |      | */A/3 |      |      |
  ------------------------------------------------------------------------------
3 |      |      |      |      |      |      |      |      |      | #/L/2 |
  ------------------------------------------------------------------------------
4 | */A/3 | #/C/5 |     | #/L/5 | #/C/2 | */C/4 |     |      |      |      |
  ------------------------------------------------------------------------------
5 | #/C/2 |      |      |      |      |      |      |      |      |      |
  ------------------------------------------------------------------------------
6 |      |      |      |      |      |      |      | */E/1 |      |      |
  ------------------------------------------------------------------------------
7 |      | */L/1 |      |      |      |      |      |      | #/L/2 |      |
  ------------------------------------------------------------------------------
8 |      |      |      |      |      | #/L/2 |      |      |      |      |
  ------------------------------------------------------------------------------
9 |      |      |      |      | #/E/1 | */E/5 |      |      | #/C/1 |      |
  ------------------------------------------------------------------------------
10|      |      |      |      | */C/1 |      |      |      |      |      |
  ------------------------------------------------------------------------------

\textbf{Estado Actual de las Fichas:}
\textbf{Soldados del Ejrcito de Francia:}
- Arqueros:
  - ArqueroX0: Ubicacin: 0, 0 | Nivel de Vida: 3 | Estado: Vivo | Actitud: Ataque | Equipo: *
```

```
  - ArqueroX1: Ubicacin: 0, 0 | Nivel de Vida: 3 | Estado: Vivo | Actitud: Ataque | Equipo: *
- Espadachines:
  - EspadachinX0: Ubicacin: G, 2 | Nivel de Vida: 1 | Estado: Vivo | Actitud: Ataque | Equipo: *
  - EspadachinX1: Ubicacin: 0, 0 | Nivel de Vida: 4 | Estado: Vivo | Actitud: Ataque | Equipo: *
  - EspadachinX2: Ubicacin: 0, 0 | Nivel de Vida: 1 | Estado: Vivo | Actitud: Ataque | Equipo: *
- Caballeros:
  - CaballeroX0: Ubicacin: G, 2 | Nivel de Vida: 5 | Estado: Vivo | Actitud: Ataque | Equipo: *
  - CaballeroX1: Ubicacin: 0, 0 | Nivel de Vida: 2 | Estado: Vivo | Actitud: Ataque | Equipo: *
  - CaballeroX2: Ubicacin: 0, 0 | Nivel de Vida: 2 | Estado: Vivo | Actitud: Ataque | Equipo: *
  - CaballeroX3: Ubicacin: 0, 0 | Nivel de Vida: 1 | Estado: Vivo | Actitud: Ataque | Equipo: #

\textbf{Prximo Movimiento:}
Toca moverse al equipo *
```

18. Commits Mas importantes

- A continuación se presentan los commits más recientes.
- Los commits se realizaron siguiendo la convención para commits de git y las recomendaciones prácticas para mensajes de commits (mensajes de forma imperativa).

Commit 91e91a27f0df5999770a04b9fa43b1486a9548ca

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:45:49 2024 -0500
Descripción: Sube la clase Soldado. La clase soldado es la superclase que configura las especificaciones por defecto de cada uno de los tipos de soldados, los que heredan de él.

Commit f49cad65f7423e4b5f1db768e8163dfb452286f2

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:45:27 2024 -0500
Descripción: Sube la clase Mapa.

Commit 60748f4acd7bd3cebda27e24ca0a892e6b1bc604

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:44:51 2024 -0500
Descripción: Sube la clase Lancero. Los lanceros son un tipo de soldados que tienen sus propios atributos y métodos específicos derivados de la clase Soldado.

Commit 0bf5a9f22e1764432171d50795054ad3f6cca534

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:44:18 2024 -0500
Descripción: Sube la clase Espadachín. Contiene las especificaciones para este tipo de guerrero, así como sus atributos y métodos internos.

Commit 1d24d9dfd7317389e08b7c5c5c2809ee4a37c9c9

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:43:57 2024 -0500
Descripción: Sube la clase Ejército.

Commit 2bfce93c00b904c6d2fcbfb736beaad4e70b7579

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:43:34 2024 -0500
Descripción: Sube la clase Caballero.

Commit 82d2ac65f1585c68fa2ae8e5575c1743a0650de4

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:43:14 2024 -0500
Descripción: Sube la clase Arquero.

Commit 618ba26f1422c79db96c0f9178ae5e646f2a04a1

Autor: JhonatanDczel
Fecha: Wed Jan 10 14:22:58 2024 -0500
Descripción: Arregla un error en el manejo de paquetes.

Commit 90e62ae10da24cb3a9a1fa203235a54fa76af1e4

Autor: JhonatanDczel
Fecha: Wed Jan 10 13:47:21 2024 -0500
Descripción: Sube la clase VideoJuego. La clase VideoJuego implementa la funcionalidad principal y contiene el método `main` que ejecuta todo el proyecto.

19. Rúbricas

19.1. 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.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 1: 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 2: 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	1.5	
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	1.5	
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		18	