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. La clase Soldado y clases heredadas

La clase `Soldado` y sus clases heredadas (`Arquero`, `Caballero`, `Espadachin` y `Lancero`) representan las entidades que participan en el juego de estrategia. A continuación, se proporciona un análisis detallado de los métodos y funcionalidades de estas clases.

15.1. Clase `Soldado`

La clase `Soldado` es la clase base que encapsula las características y comportamientos comunes de todos los soldados en el juego. Aquí hay un resumen de sus principales métodos y atributos:

Atributos:

- `nombre`: Nombre del soldado.
- `fila`: Posición en la fila.
- `columna`: Posición en la columna.
- `nivelVida`: Nivel de vida máximo.
- `vidaActual`: Vida actual del soldado.
- `actitud`: Actitud del soldado (ofensiva o defensiva).
- `vive`: Indica si el soldado está vivo.
- `team`: Equipo al que pertenece el soldado.
- `MAX`: Constante con un valor de 0.
- `numSoldados, numTeam1, numTeam2`: Contadores estáticos para el número total de soldados y la cantidad en cada equipo.

Constructores:

- `Soldado(String t)`: Constructor que inicializa el equipo y actualiza el contador según el equipo.
- `Soldado(int nV, String t)`: Constructor que también inicializa el nivel de vida.

Métodos de Ataque y Defensa:

- `atacar(), defender()`: Cambian la actitud del soldado.

Métodos de Estado y Vida:

- `serAtacado()`: Reduce la vida actual y verifica si el soldado muere.
- `morir()`: Actualiza los contadores y establece que el soldado ha muerto.
- `isLive()`: Retorna si el soldado está vivo.
- `toString()`: Retorna una representación en cadena del soldado.

Métodos de Acceso y Modificación:

- Métodos `get` y `set` para acceder y modificar los atributos.

Métodos Estáticos:

- `random(int n)`: Genera un número aleatorio entre 1 y n.
- `sumar(Soldado s)`: Crea un nuevo soldado sumando los niveles de vida de dos soldados.
- `getNumSoldados(), getNumTeam1(), getNumTeam2()`: Retorna los contadores estáticos.

15.2. Clase `Arquero`

La clase `Arquero` hereda de `Soldado` e introduce el concepto de flechas. Aquí hay un resumen de sus principales métodos y atributos:

Atributos Adicionales:

- `numFlechas`: Número de flechas disponibles.

Constructor:

- `Arquero(int nivelVida, String team)`: Llama al constructor de `Soldado` y establece el número de flechas.

Métodos de Ataque Adicionales:

- `disparar()`: Simula el disparo de una flecha y muestra la cantidad restante.

15.3. Clase `Caballero`

La clase `Caballero` hereda de `Soldado` e introduce el concepto de armas y estados. Aquí hay un resumen de sus principales métodos y atributos:

Atributos Adicionales:

- `armaActual`: Tipo de arma actual (lanza o espada).
- `state`: Estado actual del caballero (montado o desmontado).

Constructor:

- `Caballero(int nivelVida, String team)`: Llama al constructor de `Soldado` y establece la arma y estado.

Métodos de Cambio de Estado y Ataque:

- `desmontar(), emvestir()`: Cambian el estado y simulan un ataque según el estado.

Métodos de Obtención de Estado y Arma:

- `getState(), getArmaActual()`: Obtienen el estado y el tipo de arma actual.

15.4. Clase `Espadachin`

La clase `Espadachin` hereda de `Soldado` e introduce el concepto de longitud de espada y la capacidad de crear un muro. Aquí hay un resumen de sus principales métodos y atributos:

Atributos Adicionales:

- `logitudEspada`: Longitud de la espada del espadachín.

Constructor:

- `Espadachin(int nivelVida, String team, int lEspada)`: Llama al constructor de `Soldado` y establece la longitud de la espada.

Métodos Adicionales:

- `crearMuro()`: Simula la creación de un muro de escudos.

15.5. Clase `Lancero`

La clase `Lancero` hereda de `Soldado` e introduce el concepto de longitud de lanza y la capacidad de formar un schiltrom. Aquí hay un resumen de sus principales métodos y atributos:

Atributos Adicionales:

- `logitudLanza`: Longitud de la lanza del lancero.

Constructor:

- `Lancero(int nivelVida, String team, int lLanza)`: Llama al constructor de `Soldado` y establece la longitud de la lanza.

Métodos Adicionales:

- `schiltrom()`: Simula la formación de un schiltrom.

Observaciones Generales:

- La clase `Soldado` proporciona funcionalidades esenciales y se utiliza como base para los diferentes tipos de soldados.
- Las clases heredadas (`Arquero`, `Caballero`, `Espadachin`, `Lancero`) extienden las funcionalidades de `Soldado` y agregan comportamientos específicos para cada tipo de soldado.
- Los métodos adicionales en las clases heredadas permiten simular acciones específicas de cada tipo de soldado, como disparar flechas, cambiar de arma o formar formaciones defensivas.

16. Clase Mapa

La clase `Mapa` se encarga de representar el terreno y la disposición de los ejércitos en un tablero. Aquí hay un análisis detallado de la clase:

```
public class Mapa {
  final private String[] TERRENOS = {"bosque", "campo", "montaa", "desierto", "playa"};
  private Ejercito[][] table = new Ejercito[10][10];
  private String terreno = TERRENOS[random(5)];
```

Atributos:

- `TERRENOS`: Un array de strings que representa los tipos de terrenos posibles.
- `table`: Una matriz de objetos `Ejercito` que representa la disposición de los ejércitos en el mapa.
- `terreno`: Una cadena que representa el tipo de terreno del mapa.

Constructor:

- No hay un constructor explícito en la clase. La elección del tipo de terreno se realiza automáticamente al instanciar un objeto de la clase.

Métodos:

- `getTable()`: Retorna la matriz de ejércitos (`table`).
- `getTerreno()`: Retorna el tipo de terreno actual.

```
  private int random(int n) {
    return (int) (Math.random() * n);
  }
```

Método Auxiliar:

- `random(int n)`: Un método auxiliar privado que genera un número aleatorio entre 0 (inclusive) y `n` (exclusive).

Observaciones:

- La clase `Mapa` encapsula la información sobre el terreno y la disposición de los ejércitos en un tablero bidimensional.
- La elección del tipo de terreno se realiza automáticamente al crear un objeto de la clase, utilizando el método `random` para seleccionar un índice aleatorio del array `TERRENOS`.
- La matriz `table` tiene un tamaño fijo de 10x10, y cada celda puede contener un objeto `Ejercito` o ser `null` si no hay ningún ejército en esa posición.
- Los métodos `getTable` y `getTerreno` permiten acceder a la información del mapa desde otras clases.

Ejemplo de Uso:

```
Mapa miMapa = new Mapa();
Ejercito[][] matrizEjercitos = miMapa.getTable();
String tipoTerreno = miMapa.getTerreno();
```

En este ejemplo, se crea un objeto `Mapa` que automáticamente selecciona un tipo de terreno. Luego, se puede acceder a la matriz de ejércitos y al tipo de terreno utilizando los métodos proporcionados.

17. Clase VideoJuego

La clase `VideoJuego` es la clase principal que ejecuta el programa del videojuego. Aquí se analiza cada parte del código considerando los requisitos del enunciado:

```
public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  while (true) {
    // ... (declaracin de variables y configuracin inicial)
    play(table, reino1, reino2);
  }
}
```

Ciclo Principal: La ejecución del juego está envuelta en un bucle infinito, que representa el ciclo principal del videojuego. Esto permite que el juego continúe ejecutándose indefinidamente hasta que sea interrumpido.

Selección de Reinos: Se generan dos reinos aleatorios y se asegura que no sean el mismo, garantizando que no haya "guerra civil".

Creación del Mapa: Se instancia un objeto de la clase `Mapa` llamado `table`, que representa el mapa del juego.

Generación de Ejércitos: Se generan ejércitos para cada reino con un número aleatorio de soldados (entre 1 y 10). Se utiliza el método `addEjercito` para agregar soldados al mapa y a las listas de ejércitos.

Inicio del Juego: Se inicia el juego llamando al método `play` y pasando como argumentos el mapa y las listas de ejércitos.

```
public static void play(Mapa table, ArrayList<Ejercito> reino1, ArrayList<Ejercito> reino2) {
  // ... (declaracin de variables y configuracin inicial)
  while (true) {
```

```
    printTable(table);
    mover(table, reino1, reino2, turno);
    turno = turno == 1 ? 2 : 1;
  }
}
```

Ciclo del Juego: Este método maneja el flujo del juego en un bucle infinito. Dentro del bucle, se imprime el estado actual del tablero, se realiza un movimiento y se alterna el turno entre los dos reinos.

Impresión del Tablero: Se utiliza el método `printTable` para mostrar visualmente el estado actual del tablero, incluyendo la ubicación de los ejércitos y sus estadísticas.

Movimiento de Ejércitos: Se llama al método `mover` para que los jugadores realicen sus movimientos, lo que implica seleccionar un soldado y moverlo a una nueva posición en el tablero.

```
public static void addEjercito(Mapa t, ArrayList<Ejercito> r, String equipo, int i, String reino) {
  // ... (declaracin de variables y configuracin inicial)
  Ejercito ejercito = new Ejercito(equipo);
  // ... (configuracin del ejrcito, asignacin de posicin y adicin al tablero y la lista del reino)
}
```

Generación de Ejércitos: Este método se encarga de añadir un ejército al tablero y a la lista correspondiente a un reino específico. Se asegura de que no haya soldados en la misma posición y asigna atributos aleatorios a cada soldado.

```
public static void printTable(Mapa t) {
  // ... (impresin visual del tablero, mostrando la posicin y estadsticas de los soldados)
}
```

Impresión del Tablero: Este método muestra visualmente el estado actual del tablero, representando la posición de los soldados, sus equipos y estadísticas en cada celda.

```
public static void mover(Mapa t, ArrayList<Ejercito> e1, ArrayList<Ejercito> e2, int turno) {
  // ... (declaracin de variables y configuracin inicial)
  while (true) {
    // ... (solicitud de entrada del jugador y manejo de movimientos)
  }
}
```

Movimiento de Soldados: Este método permite a los jugadores mover sus soldados en el tablero. Se solicita la posición actual y la nueva posición deseada. Se valida que el movimiento sea legal antes de realizarlo.

```
public static void atacarEjercito(Ejercito e1, Ejercito e2) {
  // ... (declaracin de variables y configuracin inicial)
  while (true) {
    // ... (generacin y visualizacin del mapa de la batalla)
    moverSoldados(mapa, e1, e2, turno);
    turno = turno == 1 ? 2 : 1;
    sum1 = 0;
    sum2 = 0;
  }
}
```

Ataque a Ejército: Este método simula un enfrentamiento entre dos ejércitos. Se generan las posiciones y se muestran visualmente los movimientos en el mapa de la batalla. Luego, los soldados atacan hasta que se decide un ganador.

```
public static void atacarSoldados(Soldado[][] m, Soldado s1, Soldado s2) {
  // ... (declaracin de variables y configuracin inicial)
```

```
  while (true) {
    // ... (simulacin y visualizacin del combate entre dos soldados)
  }
}
```

Ataque a Soldados: Este método simula un combate entre dos soldados en el mapa de la batalla. Se evalúa la probabilidad de victoria y se realiza el combate.

Conclusiones: La clase `VideoJuego` implementa de manera completa la lógica central del juego, incluyendo la generación de ejércitos, el manejo de movimientos en el tablero, la simulación de batallas y la presentación visual del estado actual del juego. La estructura modular del código permite una fácil comprensión y mantenimiento.

Se ha logrado cumplir con los requisitos del enunciado, y la implementación proporciona una experiencia de juego iterativa que se ajusta a las reglas y características establecidas.

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	4	
Total		20		19	