Informe de Laboratorio 10

Tema: Laboratorio 10

Nota

Estudiante	Escuela	Asignatura
Victor Mamani Anahua vmamanian@unsa.edu.pe	Escuela Profesional de Ingeniería de Sistemas	Fundamentos de la Programación II Semestre: II Código: 20230489

Laboratorio	Tema	Duración
10	Laboratorio 10	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 25 Octubre 2023	Al 1 Noviembre 2023

1. Tarea

- Cree un Proyecto llamado Laboratorio10
- Crear 3 constructores sobrecargados.
- La actitud puede ser defensiva, ofensiva, fuga. Dicha actitud varía cuando el soldado defiende, ataca o huye respectivamente.
- Al atacar el soldado avanza, al avanzar aumenta su velocidad en 1. Al defender el soldado se para. Al huir aumenta su velocidad en 2. Al retroceder, si su velocidad es mayor que 0, entonces primero para y su actitud es defensiva, y si su velocidad es 0 entonces disminuirá a valores negativos. Al ser atacado su vida actual disminuye y puede llegar incluso a morir.
- Crear los atributos y métodos extra que considere necesarios.
- Usted deberá crear las dos clases Soldado.java y VideoJuego5.java. Puede reutilizar lo desarrollado en Laboratorios anteriores.
- Del Soldado nos importa el nombre, puntos de vida, fila y columna (posición en el tablero).
- El juego se desarrollará en el mismo tablero de los laboratorios anteriores. Para el tablero utilizar la estructura de datos más adecuada.
- Tendrá 2 Ejércitos. Inicializar el tablero con n soldados aleatorios entre 1 y 10 para cada Ejército. Cada soldado tendrá un nombre autogenerado: Soldado0X1, Soldado1X1, etc., un valor de puntos de vida autogenerado aleatoriamente [1..5], la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado).

- Además de los datos del Soldado con mayor vida de cada ejército, el promedio de puntos de vida de todos los soldados creados por ejército, los datos de todos los soldados porejército 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 2 diferentes algoritmos de ordenamiento (indicar conclusiones respecto a este ordenamiento de HashMaps).
- Finalmente, que muestre qué ejército ganará la batalla (indicar la métrica usada para decidir al ganador de la batalla).
- Crear el diagrama de clases UML.
- El juego es humano contra humano y consistirá en mover un soldado por cada turno de cada jugador. Se puede mover en cualquier dirección, Ud. deberá darle la coordenada del soldado a mover y la dirección de movimiento, el programa deberá verificar que hay un soldado del ejército que corresponda en dicha posición y que el movimiento es válido (no puede haber 2 soldados del mismo ejército en el cuadrado y no se puede ordenar moverse a una posición fuera del tablero), pidiendo ingresar nuevos datos si no es así.
- Cuando un soldado se mueve a una posición donde hay un soldado rival, se produce una batalla y gana el soldado que tenga mayor nivel de vida actual. Gana el juego quien deje al otro ejército vacío. Después de cada movida se deberá mostrar el tablero con su estado actual. Hacer un programa iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo Ubuntu GNU Linux 23 lunar 64 bits Kernell 6.2.v
- Visual Studio Code.
- VIM 9.0.
- OpenJDK 64-Bits 19.0.7.
- Git 2.39.2.
- Cuenta en GitHub con el correo institucional.
- Programación Orientada a Objetos.
- Actividades del Laboratorio 10.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/VictorMA18/fp2-23b.git`
- URL para el laboratorio 10 en el Repositorio GitHub.
- `https://github.com/VictorMA18/fp2-23b/tree/main/Fase02/Lab10`

4. Actividades del Laboratorio 10

4.1. Ejercicio Soldado

- En el primer commit agregando la clase soldado que esta siendo reutilizada para podef reslizar el siguiente ejercicio.
- El codigo y el commit seria el siguiente:

Listing 1: Commit

```
$ git commit -m "Agregando la clase soldado para podef reslizar el siguiente ejercicio "
```

Listing 2: Las lineas de codigos del metodo creado:

```
// Laboratorio Nro 10 - Ejercicio Soldado
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
import java.util.*;
public class Soldado { //CREAMOS LA CLASE SOLDODADO PARA PODER USAR UN ARREGLO
    BIDIMENSIONAL DONDE NECESITAMOS LA VIDA , EL NOMBRE DEL SOLDADO Y TAMBIEN SU
    POSICION COMO LA FILA Y LA COLUMNA

   private String name;
   private int lifeactual;
   private int row;
   private String column;
   private int attacklevel;
   private int defenselevel;
   private int lifelevel;
   private int speed;
   private String attitude;
   private boolean lives;

   Random rdm = new Random();

   //Anadiendo metodo que nos permita que un arreglo tenga datos nulos si este esta
       vacio
   public Soldado(){
      this.name = "";
      this.row = 0;
      this.column = "";
      this.attacklevel = 0;
      this.defenselevel = 0;
      this.lifelevel = 0
      this.lifeactual = 0;
      this.speed = 0;
      this.attitude = "";
      this.lives = false;
   }

   //Constructor
   public Soldado(String name, int health, int row, String column){
      this.name = name;
      this.lifeactual = health;
```

```
        this.lifelevel = health;
        this.lifeactual = health;
        this.row = row;
        this.column = column;
        this.lives = true;

        //YA QUE ESTOS DATOS SERIAN ALEATORIOS YA QUE SE ESTARIA CREANDO EL SOLDADO
            TENDRIAMOS DATOS QUE SERIAN COMO ATTACKLEVEL DEFENSELEVEL EL CUAL TENDRIAN
            QUE SER ALEATORIOS
        this.attacklevel = rdm.nextInt(5) + 1;
        this.defenselevel = rdm.nextInt(5) + 1;

    }

    //Constructor para los diferentes niveles como de vidad defensa ataque velocidad
    public Soldado(String name , int attacklevel, int defenselevel, int lifelevel, int
         speed, boolean lives, int row, String column) {
        this.name = name;
        this.attacklevel = attacklevel;
        this.defenselevel = defenselevel;
        this.lifelevel = lifelevel;
        this.speed = speed;
        this.lives = lives;
        this.row = row;
        this.column = column;
    }

    //Metodos necesarios como avanzar defender huir al seratacado al retroceder
    public void advance(){
        this.speed = getSpeed() + 1;
        System.out.println("El soldado " + this.name + "avanzo");
    }
    public void defense(){
        this.speed = 0;
        this.attitude = "DEFENSIVA";
        System.out.println("El soldado " + this.name + "esta defendiendo");
    }
    public void flee(){
        this.speed = getSpeed() + 2;
        this.attitude = "HUYE";
        System.out.println("El soldado " + this.name + "esta huyendo");
    }
    public void back(){
        System.out.println("El soldado " + this.name + "esta retrocediendo");
        if(this.speed == 0){
            this.speed = rdm.nextInt(5) - 5;
        }else{
            if(this.speed > 0){
                this.speed = 0;
                this.attitude = "DEFENSIVA";
            }
        }
    }
    public void attack(Soldado soldier){
        if(this.getLifeActual() > soldier.getLifeActual()){
            int life = this.getLifeActual() - soldier.getLifeActual();
```

```
            this.setLifeActual(life);
            this.setLifeLevel(life);
            soldier.lives = false;
            System.out.println(this.name + " asesino al soldado " + soldier.name);
         }else if(soldier.getLifeActual() > this.getLifeActual()){
            int life = soldier.getLifeActual() - this.getLifeActual();
            this.lives = false;
            soldier.setLifeActual(life);
            soldier.setLifeLevel(life);
            System.out.println(soldier.name + " asesino al soldado " + this.name);
         }else{
            this.lives = false;
            soldier.lives = false;
            System.out.println("los 2 soldados se asesinaron");
         }
      }
      public void morir(){
         this.lives = false;
         this.attitude = "SOLDADO MUERTO";
      }

      // Metodos mutadores
      public void setName(String n){
         name = n;
      }
      public void setLifeActual(int p){
         lifeactual = p;
      }
      public void setRow(int b){
         row = b;
      }
      public void setColumn(String c){
         column = c;
      }
      public void setAttackLevel(int attacklevel) {
         this.attacklevel = attacklevel;
      }
      public void setDefenseLevel(int defenselevel) {
         this.defenselevel = defenselevel;
      }
      public void setLifeLevel(int lifelevel){
         this.lifelevel = lifelevel;
      }
      public void setSpeed(int speed) {
         this.speed = speed;
      }
      public void setAttitude(String attitude) {
         this.attitude = attitude;
      }
      public void setLives(boolean lives) {
         this.lives = lives;
      }

      // Metodos accesores
      public String getName(){
         return name;
```

```
    }
    public int getLifeActual(){
       return lifeactual;
    }
    public int getRow(){
       return row;
    }
    public String getColumn(){
       return column;
    }
    public int getAttackLevel() {
       return attacklevel;
    }
    public int getDefenseLevel() {
       return defenselevel;
    }
    public int getLifeLevel(){
       return lifelevel;
    }
    public int getSpeed() {
       return speed;
    }
    public String getAttitude() {
       return attitude;
    }
    public boolean getLives() {
       return lives;
    }

    // Completar con otros metodos necesarios
    public String toString(){ //CREAMOS ESTE METODO PARA IMPRIMIR LOS DATOS DEl OBJETO
       String join = "\nNombre: " + getName() + "\nVida: " + getLifeActual() + "\nFila: "
           + getRow() + "\nColumna: " + getColumn() + "\nNivel de ataque: " +
           getAttackLevel() + "\nNivel de Defensa: " + getDefenseLevel() + "\nNivel de
           vida: " + getLifeLevel() + "\nVelocidad: " + getSpeed() + "\nActitud: " +
           getAttitude() + "\nEstado: " + getLives(); //Agregamos un espaciador para
           poder separar
       return join;
    }

}
```

4.2. Ejercicio Soldado

- En el segundo commit creamos el metodo fillRegister() se utiliza para simular la creación de un ejército de soldados en un tablero bidimensional. Comienza inicializando una matriz bidimensional llamada army que representará el ejército. Luego, se generan aleatoriamente entre 1 y 10 soldados y se los asigna a casillas en el tablero. El código evita que múltiples soldados ocupen la misma casilla. Cada soldado recibe un nombre único, puntos de salud, posición (fila y columna), y velocidad. La información de cada soldado se imprime en la consola, lo que permite rastrear su registro. Al final, la función devuelve la matriz army, que contiene la disposición aleatoria de los soldados en el tablero. Este código podría formar parte de un juego o simulación militar donde se requiere gestionar y rastrear un ejército de soldados.

- El codigo , el commit y la ejecucion seria el siguiente:

Listing 3: Commit

```
$ git commit -m "Este metodo fillRegister() se utiliza para simular la creacion de un
    ejercito de soldados en un tablero bidimensional Comienza inicializando una matriz
    bidimensional llamada army que representara el ejercito. Luego, se generan
    aleatoriamente entre 1 y 10 soldados y se los asigna a casillas en el tablero. El
    codigo evita que multiples soldados ocupen la misma casilla. Cada soldado recibe un
    nombre unico, puntos de salud, posicion (fila y columna), y velocidad. La
    informacion de cada soldado se imprime en la consola, lo que permite rastrear su
    registro. Al final, la funcion devuelve la matriz army, que contiene la disposicion
    aleatoria de los soldados en el tablero. Este codigo podria formar parte de un
    juego o simulacion militar donde se requiere gestionar y rastrear un ejercito de
    soldados."
```

Listing 4: Las lineas de codigos del metodo creado:

```
// Laboratorio Nro 10 - Ejercicio Videojuego1
// Autor: Mamani Anahua Victor Narciso
// Colaboro:
// Tiempo:
import java.util.*;
class Videojuego1 {
   public static ArrayList<ArrayList<Soldado>> fillRegister(int num){
      Random rdm = new Random();
      ArrayList<ArrayList<Soldado>> army = new ArrayList<ArrayList<Soldado>>();
      int numbersoldiers = rdm.nextInt(10) + 1; //NUMERO DE SOLDADOS ALEATORIOS ENTRE 1
          A 10 SOLDADOS
      for(int i = 0; i < 10; i++){ //ITERACION
         army.add(new ArrayList<Soldado>()); //LLENAMOS NUESTROS ARRAYLIST BIDIMENSIONAL
             CON CADA FILA PARA QUE CUMPLAN CON ESTRUCTURA DEL TABLERO
         for(int j = 0; j < 10 ; j++){//ITERACION
            army.get(i).add(null); // LLENAMOS CADA FILA DEL ARRAYLIST CON UN OBJETO
                SOLDADO CON TAL QUE ESTE SEA NULL PARA QUE SEPA QUE ESTE TIENE UNA
                CASILLA PERO NO HAY NADIE TODAVIA SE PUEDE LLENAR
         }
      }
      System.out.println("El Ejercito " + num + " tiene " + numbersoldiers + " soldados
          : " );
      System.out.println("");
      for(int i = 0; i < numbersoldiers; i++){ //LLENAMOS CASILLAS CON CADA SOLDADO
          CREADO ALEATORIAMENTE
         String name = "Soldado" + i + "X" + num;
         //System.out.println(name); PRUEBA QUE SE HIZO PARA VER LOS NOMBRES
         int health = rdm.nextInt(5) + 1;
         int row = rdm.nextInt(10) + 1;
         int speed = rdm.nextInt(5) + 1;
         String column = String.valueOf((char)(rdm.nextInt(10) + 65)); //REUTILIZAMOS
             CODIGO DEL ANTERIOR ARCHIVO VIDEOJUEGO2.JAVA YA QUE TENDRIAN LA MISMA
             FUNCIONALIDAD
         //System.out.println(army.get(row - 1).get((int)column.charAt(0) - 65)); PRUEBA
             QUE SE HIZO PARA COMPROBAR SI EL OBJETO SE ESTABA DANDO O NO CAPAZ NI
             EXISTIA
         if(army.get(row - 1).get((int)column.charAt(0) - 65) == null){
            System.out.println("Registrando al " + (i + 1) + " soldado del Ejercito " +
                num + "");
            army.get(row - 1).set((int)column.charAt(0) - 65, new Soldado(name, health,
```

```
                row, column));
            army.get(row - 1).get((int)column.charAt(0) - 65).setSpeed(speed);
            System.out.println(army.get(row - 1).get((int)column.charAt(0) -
                65).toString());
            System.out.println("---------------------------------");
         }else{
            i -= 1; //NOS AYUDARIA CON LOS SOLDADOS QUE SE REPITEN EN EL MISMO CASILLERO
                CON TAL QUE NO DEBERIA CONTAR
         }
      }
      System.out.println("*********************************");
      return army;
   }
   public static void main(String[] args) {
      ArrayList<ArrayList<Soldado>> army1 = fillRegister(1);
      ArrayList<ArrayList<Soldado>> army2 = fillRegister(2);

   }
}
```

Listing 5: Ejecucion:

```
El Ejercito 1 tiene 4 soldados :

Registrando al 1 soldado del Ejercito 1

Nombre: Soldado0X1
Vida: 1
Fila: 5
Columna: J
Nivel de ataque: 2
Nivel de Defensa: 3
Nivel de vida: 1
Velocidad: 3
Actitud: null
Estado: true
---------------------------------
Registrando al 2 soldado del Ejercito 1

Nombre: Soldado1X1
Vida: 4
Fila: 6
Columna: I
Nivel de ataque: 1
Nivel de Defensa: 1
Nivel de vida: 4
Velocidad: 5
Actitud: null
Estado: true
---------------------------------
Registrando al 3 soldado del Ejercito 1

Nombre: Soldado2X1
Vida: 1
Fila: 4
Columna: J
```

```
Nivel de ataque: 1
Nivel de Defensa: 3
Nivel de vida: 1
Velocidad: 4
Actitud: null
Estado: true
---------------------------------
Registrando al 4 soldado del Ejercito 1

Nombre: Soldado3X1
Vida: 4
Fila: 9
Columna: D
Nivel de ataque: 5
Nivel de Defensa: 2
Nivel de vida: 4
Velocidad: 5
Actitud: null
Estado: true
---------------------------------
*********************************
El Ejercito 2 tiene 4 soldados :

Registrando al 1 soldado del Ejercito 2

Nombre: Soldado0X2
Vida: 5
Fila: 6
Columna: G
Nivel de ataque: 3
Nivel de Defensa: 3
Nivel de vida: 5
Velocidad: 4
Actitud: null
Estado: true
---------------------------------
Registrando al 2 soldado del Ejercito 2

Nombre: Soldado1X2
Vida: 4
Fila: 4
Columna: F
Nivel de ataque: 4
Nivel de Defensa: 3
Nivel de vida: 4
Velocidad: 4
Actitud: null
Estado: true
---------------------------------
Registrando al 3 soldado del Ejercito 2

Nombre: Soldado2X2
Vida: 5
Fila: 3
Columna: D
Nivel de ataque: 3
```

```
Nivel de Defensa: 1
Nivel de vida: 5
Velocidad: 3
Actitud: null
Estado: true
---------------------------------
Registrando al 4 soldado del Ejercito 2

Nombre: Soldado3X2
Vida: 1
Fila: 5
Columna: D
Nivel de ataque: 2
Nivel de Defensa: 4
Nivel de vida: 1
Velocidad: 5
Actitud: null
Estado: true
---------------------------------
**********************************
```

4.3. Estructura de laboratorio 10

- El contenido que se entrega en este laboratorio10 es el siguiente:

```
/Lab10
   "PONER RAMA"
```

5. Rúbricas

5.1. Entregable Informe

Tabla 1: Tipo de Informe

Informe	
Latex	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

5.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumplio con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos lo items.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

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

Tabla 3: Rúbrica para contenido del Informe y demostración

Contenido y demostración		Puntos	Checklist	Estudiante	Profesor
1. GitHub	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
2. Commits	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
3. Código fuente	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
4. Ejecución	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
5. Pregunta	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
6. Fechas	Las fechas de modificación del código fuente estan dentro de los plazos de fecha de entrega establecidos.	2	X	2	
7. Ortografía	El documento no muestra errores ortográficos.	2	X	2	
8. Madurez	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	2	
Total		20		18	

6. Referencias

- https://drive.google.com/drive/u/1/folders/19TzLFO-T77qG7bOWmg5OH7FXAMD2CrJL