Informe de Laboratorio 20

Tema: Laboratorio 20

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
20	Laboratorio 20	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - B	Del 08 Diciembre 2024	Al 15 Diciembre 2024

1. Tarea

- Crear diagrama de clases UML y programa.
  • Crear los miembros de cada clase de la forma más adecuada: como miembros de clase o de instancia.
  • Crear la clase Mapa, que esté constituida por el tablero antes visto, que posicione soldados en ciertas posiciones aleatorias (entre 1 y 10 soldados por cada ejército, sólo 1 ejército por reino). Se deben generar ejércitos de 2 reinos. No se admite guerra civil. El Mapa tiene como atributo el tipo de territorio que es (bosque, campo abierto, montaña, desierto, playa). La cantidad de soldados, así como todos sus atributos se deben generar aleatoriamente.
  • Dibujar el Mapa con las restricciones que sólo 1 soldado como máximo en cada cuadrado.
  • El mapa tiene un solo tipo de territorio.
  • Considerar que el territorio influye en los resultados de las batallas, así cada reino tiene bonus según el territorio: Inglaterra-¿bosque, Francia-¿campo abierto, Castilla-Aragón-¿montaña, Moros-¿desierto, Sacro Imperio Romano- Germánico-¿bosque, playa, campo abierto. En dichos casos, se aumenta el Basándose en la clase Soldado crear las clases Espadachín, Arquero, Caballero y Lancero. Las cuatro clases heredan de la superclase Soldado pero aumentan atributos y métodos, o sobrescriben métodos heredados.
  • Los espadachines tienen como atributo particular "longitud de espadaz como acción çrear un muro de escudos"que es un tipo de defensa en particular.
  • Los caballeros pueden alternar sus armas entre espada y lanza, además de desmontar (sólo se realiza cuando está montando e implica defender y cambiar de arma a espada), montar (sólo se realiza cuando está desmontado e implica montar, cambiar de arma a lanza y envestir). El caballero también puede envestir, ya sea montando o desmontando, cuando es desmontado equivale

a atacar 2 veces pero cuando está montando implica a atacar 3 veces.
• Los arqueros tienen un número de flechas disponibles las cuales pueden dispararse y se gastan cuando se hace eso.
• Los lanceros tienen como atributo particular, "longitud de lanzaz como acción "schiltrom"(como una falange que es un tipo de defensa en particular y que aumenta su nivel de defensa en 1).
• Tendrá 2 Ejércitos que pueden ser constituidos sólo por espadachines, caballeros, arqueros y lanceros. No se acepta guerra civil. Crear una estructura de datos conveniente para el tablero. Los soldados del primer ejército se almacenarán en un arreglo estándar y los soldados del segundo ejército se almacenarán en un ArrayList. Cada soldado tendrá un nombre autogenerado: Espadachin0X1, Arquero1X1, Caballero2X2, etc., un valor de nivel de vida autogenerado aleatoriamente, la fila y columna también autogenerados aleatoriamente (no puede haber 2 soldados en el mismo cuadrado) y valores autogenerados para el resto de atributos.
• Todos los caballeros tendrán los siguientes valores: ataque 13, defensa 7, nivel de vida [10..12] (el nivel de vida actual empieza con el valor del nivel de vida).
• Todos los arqueros tendrán los siguientes valores: ataque 7, defensa 3, nivel de vida [3..5] (el nivel de vida actual empieza con el valor del nivel de vida).
• Todos los espadachines tendrán los siguientes valores: ataque 10, defensa 8, nivel de vida [8..10] (el nivel de vida actual empieza con el valor del nivel de vida).
• Todos los lanceros tendrán los siguientes valores: ataque 5, defensa 10, nivel de vida [5..8] (el nivel de vida actual empieza con el valor del nivel de vida).
• Mostrar el tablero, distinguiendo los ejércitos y los tipos de soldados creados. Además, se debe mostrar todos los datos de todos los soldados creados para ambos ejércitos. Además de los datos del soldado con mayor vida de cada ejército, el promedio de nivel de vida de todos los soldados creados por ejército, los datos de todos los soldados por ejército en el orden que fueron creados y un ranking de poder de todos los soldados creados por ejército (del que tiene más nivel de vida al que tiene menos) usando algún algoritmo de ordenamiento.
• Finalmente, que muestre el resumen los 2 ejércitos, indicando el reino, cantidad de unidades, distribución del ejército según las unidades, nivel de vida total del ejército y qué ejército ganó la batalla (usar la métrica de suma de niveles de vida y porcentajes de probabilidad de victoria basado en ella). Este porcentaje también debe mostrarse.
• Hacerlo programa iterativo.

2. Equipos, materiales y temas utilizados

- Sistema Operativo 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 20.

3. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- `https://github.com/VictorMA18/fp2-23b.git`
- URL para el laboratorio 20 en el Repositorio GitHub.
- `https://github.com/VictorMA18/fp2-23b/tree/main/Fase03/Lab20`

4. Actividades del Laboratorio 20

4.1. Ejercicio Soldado

- En el primer, segundo , tercero , cuarto commit en resumen creamos las clases con los atributos respectivos con sus funciones para cada clase,
- El codigo y el commit seria el siguiente:

Listing 1: Commit

```
$ git commit -m "Agregando la clase Soldado y Espadachin para poder hacer el juego
    bueno solo en la clase espadachin usamos la herencia que nos deja la clase Soldado
    y tambien creamos la funcion muroEscudo() la cual devuelve como mensaje el uso de
    esta habilidad defensiva y los getters y setters"
$ git commit -m "en la clase soldado creamos el metodo atacar el cual permite al
    caballero realizar esta funcion cuando este use su modo montar o desmontar y para
    que este embista y tambien creamos los getters de Caballero el cual es solo
    devuelve el tipo de arma que este usando"
$ git commit -m "Agregando la clase Arquero la cual va tener la funcion de disparar y
    usar la cantidad de flechas dependiendo del numero de estas va a poder atacar y
    tambien agregamos los getters y setters respectivos"
$ git commit -m "Agregando la clase Lancero la cual le agregamos su atributo principal
    que lancelth y su funcion schiltrom y tambien sus getters and setters"
```

Listing 2: Las lineas de codigos de la clase Espadachin creada:

```
public class Espadachin extends Soldado{
   private int swordlth;
   public Espadachin(String name , int attacklevel, int defenselevel, int lifelevel,
       int speed, String attitude ,boolean lives, int row, String column, int swordlth){
      super(name, attacklevel, defenselevel, lifelevel, speed, attitude, lives, row,
          column);
      this.swordlth = swordlth;
   }
   public void muroEscudo(){
      System.out.println("Usted uso la habilidad muro de Escudos");
   }
   public int getSwordlth(){
      return swordlth;
   }
   public void setSwordlth(int n){
      this.swordlth = n;
   }
}
```

Listing 3: Las lineas de codigos de la clase Caballero creada:

```
public class Caballero extends Soldado{
   private boolean montar;
   private String arma;
   public Caballero(String name , int attacklevel, int defenselevel, int lifelevel, int
       speed, String attitude ,boolean lives, int row, String column,boolean montar){
      super(name, attacklevel, defenselevel, lifelevel, speed, attitude, lives, row,
          column);
      this.montar = montar;
   }
   public void montar(){
      if(!this.montar){
         this.arma = "Lanza";
         this.embestir();
      }
   }
   public void desmontar(){
      if (this.montar) {
         this.arma = "Espada";
      }
   }
   public void embestir(){
      if(!montar){
         this.atacar();
         this.atacar();
      }else{
         this.atacar();
         this.atacar();
         this.atacar();
      }
   }
   public String getArma(){
      return arma;
   }
}
```

Listing 4: Las lineas de codigos de la clase Arquero creada:

```
public class Arquero extends Soldado{
   private int flechas;
   public Arquero(String name , int attacklevel, int defenselevel, int lifelevel, int
       speed, String attitude ,boolean lives, int row, String column, int flechas){
      super(name, attacklevel, defenselevel, lifelevel, speed, attitude, lives, row,
          column);
      this.flechas = flechas;
   }
   public void disparar(){
      if(this.flechas == 0){
         System.out.println("El arquero ya tiene flechas para poder disparar");
      }else{
         this.flechas = flechas - 1;
         this.atacar();
      }
   }
   public void setFlechas(int n){
```

```
        this.flechas = n;
    }
    public int getFlechas(){
        return flechas;
    }
}
```

Listing 5: Las lineas de codigos de la clase Lancero creada:

```
public class Lancero extends Soldado{
    private int lancelth;
    public Lancero(String name , int attacklevel, int defenselevel, int lifelevel, int
        speed, String attitude ,boolean lives, int row, String column, int lancelth){
        this.lancelth = lancelth;
    }
    public void schiltrom(){
        this.setDefenseLevel(this.getDefenseLevel() + 1);
        System.out.println("El lancero uso el schiltrom su nivel de defensa subio 1
            punto");
    }
    public void setLancelth(int n){
        this.lancelth = n;
    }
    public int getLancelth(){
        return lancelth;
    }
}
```

4.2. Ejercicio Mapa

- En el quinto commit creamos la clase Mapa y la clase Juegoprincipal la cual va tener sus atributos private String territory; private ArrayList-ArrayList-Soldado– board; private ArrayList-ArrayList-Soldado– army1; private ArrayList-ArrayList-Soldado– army2; private String[] types-territory; private String[] kingdoms; los cuales nos van ayudar para la creacion de ejercitos y asi poder hacer la relacion de herencia , composicion y tambien ya vamos creando en la funcion inciarjuego() el juego el cual en nuestro archivo Juegoprincipal lo vamos a implementar con solo esta funcion el cual por ahora solo esta trayendo el menu de inicio el cual es si quieres jugar o no,
- El codigo y el commit seria el siguiente:

Listing 6: Commit

```
$ git commit -m "Probando el menu de incio para jugar el juego"
```

Listing 7: Las lineas de codigos de la clase Mapa creada:

```
import java.util.*;

public class Mapa {
    Scanner sc = new Scanner(System.in);
    private String territory;
    private ArrayList<ArrayList<Soldado>> board;
```

```
        private ArrayList<ArrayList<Soldado>> army1;
        private ArrayList<ArrayList<Soldado>> army2;
        private String[] typesterritory = {"bosque", "campo abierto", "montana", "desierto",
            "playa"};
        private String[] kingdoms = {"Inglaterra", "Francia", "Sacro", "Castilla", "Aragon",
            "Moros"};
        public Mapa(){
           this.board = fillboard();
           this.army1 = fillarray(1);
           this.army2 = fillarray(2);
        }
        public void iniciarJuego() {
           do {
              menuBatalla();
              int resbattle = sc.nextInt();
              if(resbattle == 1){
              }else{
                 if(resbattle == 2){
                    break;
                 }else{
                    break;
                 }
              }
           } while (true);
        }
        public static ArrayList<ArrayList<Soldado>> fillboard(){
           ArrayList<ArrayList<Soldado>> army = new ArrayList<ArrayList<Soldado>>();
           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
              }
           }
           return army;
        }
        public static ArrayList<ArrayList<Soldado>> fillarray(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 menuBatalla(){
         System.out.println("-----------------------------------------");
         System.out.println("--             MENU                --");
         System.out.println("-----------------------------------------");
         System.out.println(" SELECCIONE UN NUMERO PARA PODER EMPEZAR O TERMINAR");
         System.out.println(" 1 : JUGAR");
         System.out.println(" 2 : NO JUGAR");
      }
   }
```

Listing 8: Las lineas de codigos de la clase Juegoprincipal creada:

```
public class Juegoprincipal{
   public static void main(String[] args) {
      Mapa mapa = new Mapa();
      mapa.iniciarJuego();
   }
}
```

4.3. Estructura de laboratorio 20

- El contenido que se entrega en este laboratorio20 es el siguiente:

```
\Lab20
```

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