PROGRAMACIÓN I

INFORME DE TRABAJO PRÁCTICO DE PROGRAMACIÓN: LA INVASIÓN DE LOS ZOMBIES GRINCH

Universidad Nacional de General Sarmiento

- Fabrizio Villagrán
  - DNI: 47019094
  - fabriziovillagran4@gmail.com

- Luca Espinola
  - DNI: 45140769
  - lucaesp121103@gmail.com

- Adriel Cuevas
  - DNI: 44532322
  - Cuevasadriel64@gmail.com

ÍNDICE

INFORME

INTRODUCCIÓN

El trabajo práctico consiste en el desarrollo de un videojuego de tower defense cuya jugabilidad se basa en defender una columna de regalos de un ejército de zombies mediante plantas estacionarias que pueden ser manualmente controladas por el jugador. Para ello se harán uso de los contenidos vistos a lo largo de la cursada.

DESARROLLO

Parte obligatoria

Variables de Campo:

```java
import entorno.Entorno;
import java.awt.Image;
import entorno.Herramientas;

public class Campo {
    private double x, y;     // posición del centro de la casilla
    private double ancho, alto;
    private boolean ocupada;
    private boolean bloqueadoRegalo; //bloquean las casillas de los regalos
    Image c;

    public Campo(double x, double y, double ancho, double alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.ocupada = false;
        c = Herramientas.cargarImagen("tablero.png");
```

En la clase campo se le asignaron las siguientes variables. Un X e Y, un ancho y un alto los los cuales sirven para delimitar las casillas, luego implementamos booleans ocupada, que da true cuando dentro tiene una planta/ tanque y “bloqueadoRegalo” la cual sirve para bloquear la columna 0 la cual contiene los regalos, esto da true solo para las plantas, ya que los zombies son los únicos que pueden ingresar.

Variables Regalos

```java
package juego;

import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class Regalos {

    double x, y;
    private int vida;
    private double ancho, largo;
    private double bordeIzq, bordeDer, bordeSup,bordeInf;
    double escala;
    Image gift;

    public Regalos (double x, double y, double ancho, double largo) {
        this.largo = largo;
        this.ancho = ancho;
        this.x=x;
        this.y=y;
        this.escala = 0.1;
        this.setVida(1);
        gift = Herramientas.cargarImagen("gift.jpg");
```

Esta clase cuenta con la variable vida, la cual está seteada en 1, ya que, apenas un zombie colisiona, el juego se termina. Además, cuenta con un ancho y un largo que sirve para ubicarlo dentro de la casilla y la escala para ajustar el tamaño de la imagen. Por último, cuenta con sus respectivos bordes para las colisiones.

Variables Planta y Tanque:

```java
import entorno.Entorno;

import java.awt.Image;

import entorno.Herramientas;

public class Planta {
    double x, y, escala;
    private double bordeIzq, bordeDer, bordeSup,bordeInf;
    Image imgP;
    private int vida;
    boolean seleccionada;
    private boolean enTablero = false;
    boolean colocada;
    private int contadorDisparo = 0;

    public Planta(double x, double y) {
        this.x = x;
        this.y = y;
        this.escala = 0.15;
        imgP = Herramientas.cargarImagen("pd.png");
        this.seleccionada = false;
        this.vida = 1;
        this.colocada = false;
```

En esta clase implementamos las siguientes variables, al igual que en las demás variables tiene definido un X e Y, y también tiene una escala que sirve para ajustar su imagen. Además, tiene una variable de bordes (bordeIzq, bordeDer, etc) los cuales sirven para hacer las colisiones con los zombies.

Luego, cuenta con una variable int vida, la cual setea la cantidad de golpes por tick que aguanta de los zombies. Además, cuenta con dos boolean “colocada” el cual se pone en true cuando la planta está colocada en alguna casilla, y “seleccionada” que da true cuando la planta es seleccionada con el mouse izquierdo.
Solo planta cuenta con “contadorDisparo” ya que solo planta dispara, tanque no. Otra diferencia es que el tanque tiene más vida, en lo demás son las mismas variables.

Variables Zombie

```java
import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class Zombies {

    boolean detenido;
    private double vida;
    private int danio;
    double x , y , escala;
    double velocidad;
    private double bordeIzq, bordeDer, bordeSup,bordeInf;
    Image imgz;

    public Zombies ( double x, double y) {
        this.detenido = false;
        this.velocidad = 2.0;
        this.x = x;
        this.y = y;
        this.danio = 2;
        this.vida  = 4;
        this.escala = 0.15;
        imgz = Herramientas.cargarImagen("zombies1.png");
```

En esta clase implementamos la variable “detenido” la cual la implementamos para solucionar un problema el cual era que el zombi hacia colisión con tanque pero seguía su camino, por lo cual implementamos ese boolean que sirve para que cuando choque con un tanque, hasta que este no sea null, no puede continuar.
Además, cuenta con “vida” y “danio”, a vida le dimos un valor de 4 por lo que aguantan 4 proyectiles antes de ser null, y danio es la cantidad de danio que le hacen a las plantas/tanques por tick. También cuenta con la variable velocidad la cual se encarga de cuánto avanza el zombie y por otro lado, cuenta con sus respectivos bordes para las colisiones.

Variable proyectiles

```java
import entorno.Entorno;
import entorno.Herramientas;
import java.awt.Image;

public class Proyectiles {
    private double x, y;
    private double velocidad;
    private double danio;
    private double bordeIzq, bordeDer, bordeSup,bordeInf;
    private boolean impacto;
    double escala;
    Image prImg;

    public Proyectiles (double velocidad, double danio, boolean impacto, double x, double y) {
        this.escala = 0.5;
        this.x = x;
        this.y = y;
        this.velocidad = 7.0;
        this.impacto = false;
        this.danio = 1;
        prImg = Herramientas.cargarImagen("bolaFuego.png");
    }
```

En esta clase se implementaron posición (x,y), una velocidad (double), un daño (double), los bordes para las colisiones (derecho, izquierdo, inferior, superior), un booleano impacto que detecta cuando un proyectil colisionó con un zombie y un double escala para la imagen.
La velocidad, al cambiarla, varía en cuantos pixeles suma a la posición actual de la bala por tick, generando un efecto de movimiento continuo. El daño, por otro lado, funciona restando vida del zombie al que impacta cuando se detecta una colisión con este.

Inicializar tablero:

Para comenzar, en el trabajo se nos pedía crear una cuadrícula donde se desarrolla posteriormente el juego, había distintas maneras de encarar este ejercicio, pero nosotros tomamos la decisión de crear una matriz.

Lo que hace este código es recorrer el arreglo “Campo [] []” el cual contiene a filas y columnas. Mediante un “for” se recorre el arreglo para cada una, para luego darles formas con los márgenes, el ancho y el alto de esta misma casilla.

```
private void inicializarTablero() {
    tablero = new Campo[filas][columnas];
    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            double x = margenX + j * anchoCasilla;
            double y = margenY + i * altoCasilla;
            tablero[i][j] = new Campo(x, y, anchoCasilla - 2, altoCasilla - 2);
        }
    }
}
```

Inicializar Regalos Por Fila:

El siguiente método lo que hace es generar los regalos que el jugador debe defender, se emplea un for para asegurar que los mismos aparezcan en la primera casilla de cada fila del tablero por lo mismo en campo c = tablero [i] [0] indica que recorre las filas “i” y el 0 es porque debe colocarlas en la primer columna. luego se llama al método “ocupar” para que ni la planta, ni el tanque puedan colocarse en esa casilla.

```
private void inicializarRegalosPorFila() {
    regalosPorFila = new Regalos[filas];
    for (int i = 0; i < filas; i++) {
        Campo c = tablero[i][0]; // primera columna de cada fila
        double xCentro = c.getX() + c.getAncho() / 220;
        double yCentro = c.getY() + c.getAlto() / 220;
        regalosPorFila[i] = new Regalos(xCentro, yCentro, 50.0, 50.0);
        c.ocupar();
    }
}
```

Generar Plantas:

Lo que hace este método es generar, tanto las plantas (Rose Blade) como los tanques (Wall Nut) cada ciertos ticks los cuales cuenta con el contador, genera una planta y un tanque en el hud, para que esté disponible al jugador para ser colocado.

```
private void generarPlantas() {
    contadorTicks++;

    if (contadorTicks % 300 == 0) {
        if (!hayPlantaDisponible() && roseI < RoseBlade.length) {
            double x = 50.0;
            double y = 50.0;
            RoseBlade[roseI] = new Planta(y, x);
            roseI++;
```

Hay PLANTA/TANQUE Disponible:

Aquí nos encontramos con nuestro primer obstáculo a superar. Lo que sucedió fue que con el método anterior se generaban la cantidad de plantas que le pasabamos en el array de manera que si teníamos ”RoseBlade = new Planta[15];” o “Tung = new Tanque[15];”, se generaban una detrás de la otra sin importar si ya había una en el x e y, entonces se amontonaban. Por lo tanto implementamos “hayPlantaDisponible” y “hayTanqueDisponible” los cuales se encargan de que esto mismo no suceda utilizando un boolean donde si devuelve true vuelve a contar ticks para generar una planta, sino no lo hace.

```
private boolean hayPlantaDisponible() {
    for (int i = 0; i < roseI; i++) {
        if (RoseBlade[i] != null && !RoseBlade[i].colocada) {
            return true;
        }
    }
    return false;
}

private boolean hayTanqueDisponible() {
    for (int i = 0; i < tungI; i++) {
        if (Tung[i] != null && !Tung[i].colocada) {
            return true;
        }
    }
    return false;
}
```

Arrastrar:

Permite mover con el mouse las distintas plantas con el botón izquierdo del mouse, cabe aclarar que una vez seleccionada una casilla las plantas no pueden volver a ser arrastradas.

```
private void arrastrar(Planta rosa) {
    rosa.x = entorno.mouseX();
    rosa.y = entorno.mouseY();
}

private void arrastrarT(Tanque tung) {
    tung.x = entorno.mouseX();
    tung.y = entorno.mouseY();
}
```

Colocar PLANTA / TANQUE en casilla:

Se establece la posición en el tablero de modo que la planta quede fija en el lugar que le corresponde, además se utiliza el “ocupar” para que en esa casilla no puedan existir más de una planta/tanque

```
private void colocarPlantaEnCasilla(Planta p, int fila, int col) {
    Campo c = tablero[fila][col];
    p.x = c.getX();
    p.y = c.getY();
    c.ocupar();
}

private void colocarTanqueEnCasilla(Tanque t, int fila, int col) {
    Campo c = tablero[fila][col];
    t.x = c.getX();
    t.y = c.getY();
    c.ocupar();
}
```

```java
private void colocarEnCasilla(Tanque t) {
    double mx = entorno.mouseX();
    double my = entorno.mouseY();

    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            Campo c = tablero[i][j];

            double xMin = c.getX() - (anchoCasilla / 2);
            double xMax = c.getX() + (anchoCasilla / 2);
            double yMin = c.getY() - (altoCasilla / 2);
            double yMax = c.getY() + (altoCasilla / 2);

            if (mx >= xMin && mx <= xMax && my >= yMin && my <= yMax) {
                if (!c.estaOcupada()) {
                    t.x = c.getX();
                    t.y = c.getY();
                    t.colocada = true;
                    c.ocupar();
                    System.out.println("Tung colocado en [" + i + "][" + j + "]");
                }
                return;
            }
        }
    }
}
```

Generar zombies:

Este método hace que, cada 200 ticks genera un zombie en el arreglo zombie[] en una fila aleatoria, para ello utilizamos los recursos de java.util.random.En donde filaAleatoria genera un número al azar entre 0 y 4 (cantidad de filas).

```java
private void generarZombies() {
    contadorTicksZ++;
    if (contadorTicksZ % 200 == 0 && zomI < zombie.length) {
        int filaAleatoria = random.nextInt(filas);
        double x = tablero[filaAleatoria][columnas - 1].getX();
        double y = tablero[filaAleatoria][columnas - 1].getY();
        zombie[zomI] = new Zombies(x + 50, y);
        zomI++;
        System.out.println("Zombie generado en fila " + filaAleatoria);
    }
}
```

Método para detectar zombies:

Este booleano evalúa si la fila donde está la planta del arreglo y la fila donde está el zombie del arreglo coinciden, si es así se marca true. Caso contrario marca false.

```
private boolean detectarZ(Planta p) {
    int filaplanta = (int) ((p.y - margenY) / altoCasilla);
    if (filaplanta < 0 || filaplanta >= filas) {
        return false;
    }

    for (int i = 0; i < zomI; i++) {
        Zombies z = zombie[i];
        if (z != null) {

            int filaZombie = (int) ((z.getY() - margenY) / altoCasilla);
            if (filaZombie == filaplanta) {
                return true;
            }
        }
    }
    return false;
}
```

Métodos de colisión:

Estos 4 métodos evalúan si los bordes, calculados en cada clase, coinciden. Es decir, están lo suficientemente cerca (o literalmente tocándose) para marcarlo como una colisión.

```
private boolean colisionaConRegalo(Zombies z, Regalos g) {
    return !(z.getBordeDer() < g.getBordeIzq() ||
             z.getBordeIzq() > g.getBordeDer() ||
             z.getBordeSup() > g.getBordeInf() ||
             z.getBordeInf() < g.getBordeSup());
}

public boolean colisionConPlanta(Zombies z, Planta p) {
    return !(z.getBordeDer() < p.getBordeIzq() ||
             z.getBordeIzq() > p.getBordeDer() ||
             z.getBordeSup() > p.getBordeInf() ||
             z.getBordeInf() < p.getBordeSup());
}

public boolean colisionConTanque(Zombies z, Tanque t) {
    return !(z.getBordeDer() < t.getBordeIzq() ||
             z.getBordeIzq() > t.getBordeDer() ||
             z.getBordeSup() > t.getBordeInf() ||
             z.getBordeInf() < t.getBordeSup());
}

private boolean colisionConProyectil(Proyectiles pr, Zombies z) {
    return !(pr.getBordeDer() < z.getBordeIzq() ||
             pr.getBordeIzq() > z.getBordeDer() ||
             pr.getBordeSup() > z.getBordeInf() ||
             pr.getBordeInf() < z.getBordeSup());
}
```

Métodos para poder seleccionar las plantas:

Estos métodos evalúan si el mouse está dentro de la figura de la planta o del tanque respectivamente. Este método es utilizado para poder seleccionar las plantas que hay disponibles y poder arrastrarlas hasta una casilla libre.

```
private boolean clickSobrePlanta(Planta p) {
    double mx = entorno.mouseX();
    double my = entorno.mouseY();
    double ancho = 50;
    double alto = 50;
    return mx >= p.x - ancho / 2 && mx <= p.x + ancho / 2 && my >= p.y - alto / 2 && my <= p.y + alto / 2;
}

private boolean clickSobreTanque(Tanque t) {
    double mx = entorno.mouseX();
    double my = entorno.mouseY();
    double ancho = 50;
    double alto = 50;
    return mx >= t.x - ancho / 2 && mx <= t.x + ancho / 2 && my >= t.y - alto / 2 && my <= t.y + alto / 2;
}
```

Método para colocar plantas en una casilla libre:

Estos dos métodos detectan si una planta seleccionada es colocada sobre una casilla libre, si es así marca la casilla como ocupada para que no se pueda mover otra planta o tanque a ese lugar y centra la planta o tanque en la casilla ocupada.

```
private void colocarEnCasilla(Planta p) {
    double mx = entorno.mouseX();
    double my = entorno.mouseY();

    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            Campo c = tablero[i][j];

            double xMin = c.getX() - (anchoCasilla / 2);
            double xMax = c.getX() + (anchoCasilla / 2);
            double yMin = c.getY() - (altoCasilla / 2);
            double yMax = c.getY() + (altoCasilla / 2);

            if (mx >= xMin && mx <= xMax && my >= yMin && my <= yMax) {
                if (!c.estaOcupada()) {
                    p.x = c.getX();
                    p.y = c.getY();
                    c.ocupar();
                    p.colocada = true;
                    System.out.println("RoseBlade colocada en [" + i + "][" + j + "]");
                }
                return;
            }
        }
    }
}


private void colocarEnCasilla(Tanque t) {
}
```

Métodos de selección única:

Estos booleanos evalúan si ya hay una planta seleccionada en los métodos de MoverPlanta. Esto lo hicimos como respuesta a un problema que era al arrastrar una planta, si esta pasaba por encima de otra también se seleccionaba superponiendo. Estos métodos hacen que solo se pueda seleccionar una planta a la vez.

```
private boolean seleccionada(Planta p) {
    double cursorX = entorno.mouseX();
    double cursorY = entorno.mouseY();
    return cursorX > p.getBordeIzq() && cursorX < p.getBordeDer() && cursorY > p.getBordeSup() && cursorY < p.getBordeInf();
}

private boolean seleccionada(Tanque t) {
    double cursorX = entorno.mouseX();
    double cursorY = entorno.mouseY();
    return cursorX > t.getBordeIzq() && cursorX < t.getBordeDer() && cursorY > t.getBordeSup() && cursorY < t.getBordeInf();
}
```

Métodos de arrastre:

Estos dos métodos funcionan si una planta del HUD es seleccionada, siguen al mouse hasta que son soltadas en una casilla libre, luego se centran con el método colocarEnCasilla explicado anteriormente.

```
  private void arrastrar(Planta rosa) {
      rosa.x = entorno.mouseX();
      rosa.y = entorno.mouseY();
  }

  private void arrastrarT(Tanque tung) {
      tung.x = entorno.mouseX();
      tung.y = entorno.mouseY();
  }
```

Método de reinicio de juego:

Este método se aplica, ya sea cuando ganamos o cuando perdemos, y reinicia contadores de ticks, los índices, inicializa en false los booleanos de JuegoTerminado, libera todos los

casilleros del tablero, y limpia todos los arreglos volviendo null sus contenidos.

```
private void reiniciarJuego() {

    juegoTerminado = false;
    contadorTicks = 0;
    contadorTicksZ = 0;
    roseI = 0;
    tungI = 0;
    zomI = 0;
    proyI = 0;
    zombiesDerrotados = 0;
    juegoGanado = false;

    for (int i = 0; i < proyectiles.length; i++) {
        proyectiles[i] = null;
    }
    for (int i = 0; i < RoseBlade.length; i++) {
        RoseBlade[i] = null;
    }
    for (int i = 0; i < Tung.length; i++) {
        Tung[i] = null;
    }
    for (int i = 0; i < zombie.length; i++) {
        zombie[i] = null;
    }


    for (int i = 0; i < filas; i++) {
        for (int j = 0; j < columnas; j++) {
            tablero[i][j].liberar();
        }
    }


    inicializarRegalosPorFila();
```

Conclusión

Fue un trabajo extenso, donde se implementó de manera repetida los bucles “for” y más cosas aprendidas durante las clases. Tuvimos complicaciones las cuales no tomaron mucho tiempo resolver ya que manejamos entornos y modelos de programa que no estábamos acostumbrados. Pero por sobre todo fue divertido poder ver los resultados de lo que como grupo fuimos construyendo. Aprendimos a trabajar con un entorno ya creado, además aprendimos cómo manejarnos con el “tick()” algo que no conocíamos hasta este momento. También aprendimos a trabajar en conjunto con nuestros compañeros y desarrollar nuestro conocimiento para resolver problemas más complejos como esquematizar un juego desde cero. Por último, adjuntamos una imagen de como quedó y estamos felices con este resultado.