Universidad Nacional de General Sarmiento

Programación 1

Trabajo práctico "La invasión de los Zombies Grinch"

integrantes :

Cavila Andres — andrescaliva25@gmail.com

Galeano Adriana — adrianagaleano077@gmail.com

Medina Solange — solemedina193@gmail.com

Correos electronicos:

Docentes :

Verónica Moyano

Cesar Niveyro

Introducción:

En el pueblo de Whoville, el Grinch se convirtió en zombi y tiene la capacidad de copiarse a sí mismo, en la época de Navidad sigue atacando al pueblo para robar los regalos. El científico del pueblo creó algunos nuevos tipos de plantas con la capacidad de atacar y defenderse de los Zombies Grinch. Por ejemplo está Rose Blade una rosa que tira bolas de fuego y Wall-Nut una nuez con una gran resistencia y defensa, esta última sirve para frenar el ataque de los Zombies Grinchs.

El objetivo es desarrollar un video juego para prepararse y poder proteger los regalos del ataque de los Zombies Grinch en esta Navidad.

Descripción:

A continuación, se dará una explicación general de cada clase implementada, describiendo las variables de instancia y dando una breve descripción de cada método.

Clase: Casillero

La clase `Casillero` representa una casilla de un tablero dentro del juego. Cada casillero tiene una posición, un tamaño y un color que indica su estado (verde brillante o verde oscuro). La clase permite dibujar la casilla en pantalla y ver sus propiedades.

Variables de instancia

Nombre	Tipo	Descripción

`x`	`int`	Coordenada X del centro del casillero en la ventana del juego.
`y`	`int`	Coordenada Y del centro del casillero en la ventana del juego.
`ancho`	`int`	Ancho del casillero.
`alto`	`int`	Alto del casillero.
`esVerde`	`boolean`	Indica si el casillero se dibuja en verde brillante (`true`) o verde oscuro (`false`).

Constructores

```
public Casillero(int x, int y, int ancho, int alto, boolean esVerde)
```

- Inicializa un objeto `Casillero` con su posición, tamaño y color.

- Parámetros:

    - `x`, `y`: posición del casillero.

    - `ancho`, `alto`: dimensiones del casillero.

- `esVerde`: estado de color del casillero.

:

Métodos:

Método	Tipo de retorno	Descripción
`public void dibujar(Entorno entorno)`	`void`	Dibuja el casillero en la pantalla usando la librería `Entorno`. Si `esVerde` es `true`, se dibuja verde; si es `false`, verde oscuro.
`public int getX()`	`int`	Devuelve la coordenada X del casillero.
`public int getY()`	`int`	Devuelve la coordenada Y del casillero.
`public int getAncho()`	`int`	Devuelve el ancho del casillero.
`public int getAlto()`	`int`	Devuelve el alto del casillero.

Clase: Regalos

La clase `Regalos` representa un objeto regalo dentro del juego. Cada regalo tiene una posición en la pantalla y un tamaño fijo. La clase permite dibujar el regalo en la ventana del juego y consultar sus coordenadas.

Variables de instancia

Nombre	Tipo	Descripción
`x`	`double`	Coordenada X del centro del regalo en la ventana del juego.
`y`	`double`	Coordenada Y del centro del regalo en la ventana del juego.

Constructores

```
public Regalos(double x, double y)
```

- Inicializa un objeto `Regalos` con su posición en la pantalla.

- Parámetros:

    - `x`: coordenada horizontal del regalo.

    - `y`: coordenada vertical del regalo.

Métodos

Método	Tipo de retorno	Descripción
`public void dibujar(Entorno e)`	`void`	Dibuja el regalo como un cuadrado azul de tamaño fijo en la ventana del juego utilizando la librería `Entorno`.
`public double getX()`	`double`	Devuelve la coordenada X del regalo.
`public double getY()`	`double`	Devuelve la coordenada Y del regalo.

Clase: Planta

La clase `Planta` representa una planta dentro del juego, que puede ser de tipo normal o especial (`roseBlade`). Cada planta tiene posición, vida, tamaño y puede disparar proyectiles si es del tipo `roseBlade`.

Variables de instancia

Nombre	Tipo	Descripción
`x`	`int`	Coordenada horizontal (fila) de la planta.
`y`	`int`	Coordenada vertical (columna) de la planta.
`vida`	`int`	Cantidad de vida que le queda a la planta.
`diametro`	`int`	Tamaño visual de la planta al dibujarla.
`seleccionada`	`boolean`	Indica si la planta está actualmente seleccionada por el jugador.

`roseBlade`	`boolean`	Indica si la planta es de tipo especial que puede disparar proyectiles.
`intervalosDisparos`	`int`	Cantidad de ticks entre disparos para plantas tipo `roseBlade`.
`contadorDisparos`	`int`	Contador para controlar cuándo la planta puede disparar de nuevo.
`juego`	`Juego`	Referencia al objeto `Juego` para interactuar con otras mecánicas (por ejemplo, disparar proyectiles).

Constructores

```
public Planta(int x, int y, boolean roseBlade, Juego juego)
```

- Inicializa una planta en la posición `(x, y)`.
- Parámetros:
  - `x`, `y`: coordenadas de la planta.
  - `roseBlade`: indica si es del tipo especial que dispara.
  - `juego`: referencia al juego para ejecutar disparos.

- Define el tamaño y el intervalo de disparos si corresponde.

Métodos

Método	Tipo de retorno	Descripción
`public void dibujar(Entorno e)`	`void`	Dibuja la planta como un círculo amarillo o azul (si está seleccionada). Si es `roseBlade`, dibuja un círculo rojo más pequeño encima.
`public void actualizarEstado()`	`void`	Controla el disparo automático de la planta si es `roseBlade`. Cada `intervalosDisparos` ticks dispara un proyectil a la derecha usando el método `juego.disparar()`.
`public boolean contienePunto(int px, int py)`	`boolean`	Verifica si un punto `(px, py)` está dentro del área de la planta (útil para selección con clic).
`public void moverPlanta(int dx, int dy)`	`void`	Mueve la planta sumando desplazamientos a sus coordenadas X e Y.
`public void recicbirDanio(int danio)`	`void`	Reduce la vida de la planta según el daño recibido.

`public boolean plantaMuerta()`	`boolean`	Devuelve `true` si la vida de la planta es 0 o menor.
`public int getX()`	`int`	Retorna la coordenada X de la planta.
`public int getY()`	`int`	Retorna la coordenada Y de la planta.
`public int getDiametro()`	`int`	Retorna el tamaño de la planta.
`public boolean isSeleccionada()`	`boolean`	Indica si la planta está seleccionada.
`public boolean isRoseBlade()`	`boolean`	Indica si la planta es del tipo especial.
`public void setSeleccionada(boolean seleccionada)`	`void`	Permite cambiar el estado de selección de la planta.
`public void moverArriba(int desplazamiento)`	`void`	Mueve la planta hacia arriba.
`public void moverAbajo(int desplazamiento)`	`void`	Mueve la planta hacia abajo.
`public void moverIzquierda(int desplazamiento)`	`void`	Mueve la planta hacia la izquierda.
`public void moverDerecha(int desplazamiento)`	`void`	Mueve la planta hacia la derecha.

`public void setX(int x)`	`void`	Cambia la coordenada X de la planta.

Clase: Grinch

La clase `Grinch` representa a un zombie del juego que se desplaza hacia la izquierda, puede recibir daño y morir. También puede interactuar con los objetos `Regalos` para detectar colisiones, se dibuja como un conjunto de círculos de colores rojo y negro.

Variables de instancia

Nombre	Tipo	Descripción
`x`	`double`	Coordenada horizontal del grinch (fila).
`y`	`double`	Coordenada vertical del grinch (columna).
`velocidad`	`double`	Velocidad a la que se mueve hacia la izquierda.
`vida`	`double`	Cantidad de vida que le queda al grinch.
`tamanio`	`int`	Tamaño visual del grinch (radio aproximado para colisiones).

Constructores

```
public Grinch(double x, double y)
```

- Inicializa un grinch en la posición `(x, y)`.

- Configura valores por defecto:

    - `velocidad = 1`

    - `vida = 10`

    - `tamanio = 20`

Métodos

Método	Tipo de retorno	Descripción
`public void dibujarGrinch(Entorno e)`	`void`	Dibuja el grinch en el entorno usando varios círculos concéntricos de colores rojo y negro para darle apariencia.
`public void moverIzquierda()`	`void`	Mueve el grinch hacia la izquierda según su velocidad, siempre que no salga del borde izquierdo (`x > 0`).

`public boolean grinchMuerto()`	`boolean`	Devuelve `true` si la vida del grinch es 0 o menor.
`public boolean chocaConRegalo(Regalos r)`	`boolean`	Comprueba si el grinch colisiona con un objeto `Regalos`. Calcula la distancia entre los centros y la compara con la suma de los radios (semilados) de ambos objetos.
`public void recibirDanio(int danio)`	`void`	Reduce la vida del grinch según el daño recibido.
`public double getX()`	`double`	Retorna la coordenada X del grinch.
`public void setX(int x)`	`void`	Modifica la coordenada X del grinch.
`public double getY()`	`double`	Retorna la coordenada Y del grinch.
`public void setY(int y)`	`void`	Modifica la coordenada Y del grinch.
`public int getTamanio()`	`int`	Retorna el tamaño visual del grinch.

Clase: *BarraSuperior*

La clase **BarraSuperior** representa una franja ubicada en la parte superior de la pantalla del juego.

Su función principal es dibujar una barra horizontal de color gris que sirve para mostrar la información, ya sea el tiempo transcurrido y cuantos zombies voy eliminando.

Variables de instancia

Nombre	Tipo	Descripción
alturaBanda	*int*	Altura en píxeles de la barra superior. Determina el grosor de la franja que se dibuja en la parte superior de la ventana.

Constructores

La clase no define un constructor explícito, por lo tanto utiliza lo predeterminado
Este inicializa la variable *alturaBanda* con el valor **70**.

Métodos

Método	Tipo de retorno	Descripción
public void dibujar(Entorno entorno)	*void*	Dibuja la barra superior en la pantalla utilizando la librería *Entorno*. El rectángulo se centra horizontalmente en la ventana, ocupa todo el ancho de la pantalla y se dibuja en color gris.
public int getAlturaBanda()	*int*	Devuelve la altura de la barra superior, permitiendo que otras clases conozcan el espacio que ocupa en la parte superior de la pantalla

Clase: *Disparo*

La clase **Disparo** representa un disparado dentro del juego.
Cada disparo tiene una posición, un tamaño, una velocidad, un color y un valor de daño.

Variables de instancia

Nombre	Tipo	Descripción
x	*double*	Coordenada X del centro del disparo en la ventana del juego.
y	*double*	Coordenada Y del centro del disparo en la ventana del juego.
radio	*int*	Radio del disparo (tamaño del círculo que lo representa).
velocidad	*double*	Velocidad con la que el disparo se desplaza hacia la derecha.
activado	*boolean*	Indica si el disparo está activo (true) o desactivado (false).
color	*Color*	Color con el que se dibuja el disparo.
danio	*int*	Cantidad de daño que el disparo provoca al impactar.

Constructores

public Disparo(**double** x, **double** y, **int** radio, **double** velocidad, Color color, **int** danio)

Inicializa un objeto **Disparo** con su posición, tamaño, velocidad, color y daño.

Parámetros:

- *x, y:* posición del disparo en la pantalla.
- *radio:* tamaño del disparo.
- *velocidad:* rapidez del movimiento horizontal.
- *color:* color del proyectil.
- *danio:* valor de daño que causa el disparo.

Métodos

Método	Tipo de retorno	Descripción
public void moverDerecha()	*void*	Mueve el disparo hacia la derecha sumando su velocidad a la coordenada *x*.
public void dibujar(Entorno e)	*void*	Dibuja el disparo en pantalla como un círculo amarillo si está activado.

public void desactivar()	*void*	Desactiva el disparo (deja de mostrarse en pantalla).
public double getX()	*double*	Devuelve la coordenada X actual del disparo.
public double getY()	*double*	Devuelve la coordenada Y actual del disparo.
public int getRadio()	*int*	Devuelve el radio del disparo.
public boolean isActivado()	*boolean*	Indica si el disparo está activado.
public Color getColor()	*Color*	Devuelve el color actual del disparo.
public int getDanio()	*int*	Devuelve el daño que el disparo inflige.

Clase: *Juego*

La clase **Juego** controla la lógica principal del juego, la creación del tablero, las plantas, los zombies (Grinch), los disparos, los regalos y la interfaz gráfica a través de **Entorno**. Se encarga de actualizar el estado del juego en cada instante mediante el método `tick()`.

Variables de instancia principales

Nombre	Tipo	Descripción

entorno	*Entorno*	Objeto que maneja la ventana, gráficos y eventos del juego.
fila	*int*	Número de filas del tablero (5).
columna	*int*	Número de columnas del tablero (10).
tablero	*Casillero[][]*	Matriz de casilleros que representa el tablero de juego.
zombieGrinch	*Grinch[]*	Arreglo de zombies (Grinch) activos en el juego.
plantas	*Planta[][]*	Matriz que contiene las plantas colocadas en el tablero.
regalos	*Regalos[]*	Arreglo de regalos.
disparos	*Disparo[]*	Arreglo que guarda los disparos realizados por las plantas.
barraSuperior	*BarraSuperior*	Objeto que muestra información del juego .
arrastrando	*boolean*	Indica si el jugador está arrastrando una carta de planta.
plantaPrevizualizada	*Planta*	Planta que se muestra en vista previa mientras se arrastra.

cartaRoseX, cartaRoseY, cartaAncho, cartaAlto	*int*	Posición y tamaño de la carta de planta en la interfaz.
juegoTerminado, juegoGanado	*boolean*	Indican si el juego terminó y si fue ganado.
tiempoInicio	*long*	Marca el momento en que inicia el juego (ms).
tiempo	*int*	Contador de tiempo para generación de zombies.
random	*Random*	Generador de números aleatorios para lógica del juego.
totalEnemigos, enemigosGenerados, maxZombiesSimultaneos, zombiesEliminados, zombiesRestantes	*int*	Contadores de enemigos y control de la dificultad.
minimioTiempoRegeneracion , maximoTiemporRegernacion	*int*	Tiempo mínimo y máximo para generar un nuevo zombie.

Constructores

public Juego()

Inicializa el juego, creando:

- El tablero de casilleros con colores alternados.

- Los regalos por fila.

- Las matrices de plantas y disparos.

- Arreglo de zombies.

- Barra superior y temporizadores.

- Posición inicial de prueba de una planta.

- Finalmente, inicia la ejecución del juego con `entorno.iniciar()`.

Métodos principales

Método	**Tipo de retorno**	**Descripción**
public void tick()	*void*	Método central que se ejecuta cada frame. Actualiza el juego: dibuja tablero, plantas, zombies, disparos , maneja arrastre y colocación de plantas, controla colisiones y verifica victoria o derrota.
public void disparar(double x, double y)	*void*	Crea un nuevo disparo en las coordenadas X, Y si hay espacio en el arreglo de disparos.

public int[] int[] cooodenadasCasillero(int x, int y)	Retorna la fila y columna del casillero que contiene las coordenadas X, Y. Devuelve null si no hay casillero.
public void void deseleccionarTodas()	Desmarca todas las plantas del tablero como no seleccionadas.
public static void void main(String[] args)	Inicia el juego creando un objeto **Juego**.

IMPLEMENTACIONES:

```
package juego;

import java.awt.Color;

import java.util.Random;


import entorno.Entorno;

import entorno.InterfaceJuego;


public class Juego extends InterfaceJuego {

	// El objeto Entorno que controla el tiempo y otros

	private Entorno entorno;


	// Variables y métodos propios de cada grupo

	//Tamaño del tablero

	private int fila=5;  // usar dentro los metodos

	private int columna=10; // usar dentro los metodos

	private Casillero[][] tablero;

	private Grinch[] zombieGrinch;//Zombies grinch

	private Planta[][] plantas;//Matriz de plantas
```

```
private Regalos[] regalos;

private Disparo[] disparos;

private int minimioTiempoRegeneracion=60;

private int maximoTiemporRegernacion=180;

private int totalEnemigos=60;

private int enemigosGenerados=0;

private int maxZombiesSimultaneos=10;

private int zombiesEliminados=0;

private int zombiesRestantes=totalEnemigos;

private int tiempo;

private Random random=new Random(); // usar dentro los metodos

private BarraSuperior barraSuperior;


//Carta de plantas para el HUD

private boolean arrastrando=false;

private Planta plantaPrevizualizada=null;

private CartaPlanta carta;

private int tiempoTranscurrido=0;

//private int cartaRoseX=80;  //Clase carta

//private int cartaRoseY=30; //Clase carta

//private int cartaAncho=60; //Clase carta

//private int cartaAlto=40; //Clase carta

int tiempoDisp = 500;//Clase carta




private boolean juegoTerminado=false;
```

```
	private boolean juegoGanado=false;



	public Juego() {

		// Inicializa el objeto entorno

		this.entorno = new Entorno(this, "La inavasion de los zombies Grinch - Grupo 6 - Caliva - Galeano -Medina", 800, 600);

		// Inicializar lo que haga falta para el juego

		this.zombieGrinch=new Grinch[15];  //arreglo con 15 zombies

		//zombieGrinch[0] = new Grinch(900);   //En la posición 0 creo un zombie nuevo en la posición X = 900

		//this.grinch=new Grinch(800);      //gricn en la pos 800

		this.plantas=new Planta[fila][columna];  //Matriz de plantas

		this.regalos = new Regalos[fila];     //Crea un arrego de regaloscon tantas posiciones como filas hay en el tablero

		this.disparos=new Disparo[50];        //arreglo para los disparos

		this.carta=new CartaPlanta(80,30,60,40,500);

		this.barraSuperior = new BarraSuperior();

		this.zombiesEliminados=0;

		this.zombiesRestantes=totalEnemigos;

this.tiempo=random.nextInt(maximoTiemporRegernacion-minimioTiempoRegeneracion+1)+minimioTiempoRegeneracion;
```

```
                    // Define el desplazamiento (70pixeles)
                    int yInicialDesplazamiento = 70;

                    // Nuevo Alto del área de juego (600 - 70 = 530)
                    int altoAreaJuego = entorno.alto() - yInicialDesplazamiento;

                    // Nuevo altoCasillero (530 / 5 = 106)
                    int anchoCasillero = entorno.ancho() / columna;
                    int altoCasillero = altoAreaJuego / fila;   // 106 de alto cada casillero  (106*5 = 530)

                    // Posición X (Columna 0, centrada):
                    int xInicialRegalo = anchoCasillero / 2;


                    // 5 regalos, uno por fila
                    for (int i = 0; i < fila; i++) {
                        int  yInicialRegalo  = yInicialDesplazamiento + (i * altoCasillero) + (altoCasillero / 2);     // centro y del regalo , el 70 mueve todo hacia abajo
                      this.regalos[i] = new Regalos(xInicialRegalo, yInicialRegalo);
                    }

  //Creacion del tablero
          this.tablero = new Casillero[fila][columna];


                    // Recorremos todas las filas.
                    for (int i = 0; i < fila; i++) {
```

```
// Dentro de cada fila, recorremos todas las columnas.
                    for (int j = 0; j < columna; j++) {
                        int x=j*anchoCasillero+anchoCasillero/2;

    // AHORA SUMA EL DESPLAZAMIENTO
                        int y= yInicialDesplazamiento + i*altoCasillero + altoCasillero /2;

                        // La suma de la fila (i) y la columna (j) determina el color de la casilla.
                        // Si la suma es par, es una casilla verde claro.
                        if ((i + j) % 2 == 0) {
                            tablero[i][j] = new Casillero(x, y, anchoCasillero, altoCasillero,true);
                        } else {
    // Si la suma es impar, es una casilla verde oscuro.
                            tablero[i][j] = new Casillero(x, y, anchoCasillero, altoCasillero, false);
                        }

                    }
                }

                int testFila=2;
                int testColumna=1;
                int px=tablero[testFila][testColumna].getX();
                int py=tablero[testFila][testColumna].getY();
                this.plantas[testFila][testColumna]=new Planta(px,py,true,this);
                this.plantas[testFila][testColumna].setSeleccionada(true);
```

```
		// Inicia el juego!
		this.entorno.iniciar();

	}

	/**
	 * Durante el juego, el método tick() será ejecutado en cada instante y
	 * por lo tanto es el método más importante de esta clase. Aquí se debe
	 * actualizar el estado interno del juego para simular el paso del tiempo
	 * (ver el enunciado del TP para mayor detalle).
	 */
	public void tick()
	{
		tiempoTranscurrido++;
		carta.actualizarTicks();//actualiza la carga de disponibildad planta


		this.barraSuperior.dibujar(entorno, zombiesEliminados, zombiesRestantes, tiempoTranscurrido/60);
		this.carta.Dibujar(entorno);
		//entorno.dibujarRectangulo(cartaRoseX,cartaRoseY,cartaAncho,cartaAlto,0,Color.PINK);

		//entorno.cambiarFont("Arial", 18, Color.black);
		//entorno.escribirTexto("Rose Blade" + tiempoDisp, (double)cartaRoseX, (double)cartaRoseY);

		/*if(tiempoDisp>0) {
```

```
                tiempoDisp--;
                }*/

    //Dibujo del tablero
                for (int i = 0; i < fila; i++) {
//      // Dentro de cada fila, recorremos todas las columnas.
                    for (int j = 0; j < columna; j++) {
                        if(tablero[i][j]!=null){
                            tablero[i][j].dibujar(entorno);
                        }
                    }

                }

                //Manjeo de arrastre de la carta de planta
                arrastrePlanta();



                if(enemigosGenerados<totalEnemigos){
                    tiempo--;
                    //Cuenta cuantos zombies activos hay en el tablero
                    int activos=contadorActivos(zombieGrinch);
                    if(tiempo<=0 && activos<maxZombiesSimultaneos) {

                        int zombiesAntes=activos;
```

```
                                        Grinch.generacionZombiesAleatorios(zombieGrinch, tablero, fila, columna,
entorno, random);

                                        int zombiesDespues=contadorActivos(zombieGrinch);

                                        if(zombiesAntes>zombiesDespues) {

                                                enemigosGenerados++;

                                                zombiesRestantes=Math.max(0, totalEnemigos);


tiempo=random.nextInt(maximoTiemporRegernacion-minimioTiempoRegeneracion+1)+minimioTiempoRege
neracion;

                                        }

                                        /*

                                        * for(int i=0;i<zombieGrinch.length;i++) { if (zombieGrinch[i] == null) { int

                                        * tipo = random.nextInt(3); int filaAleatoria = random.nextInt(fila); int yFila

                                        * = tablero[filaAleatoria][columna - 1].getY(); if (tipo == 0) {

                                        * zombieGrinch[i] = new Grinch(entorno.ancho() + 50, yFila+10, -1,

                                        * filaAleatoria); } else if (tipo == 1) { zombieGrinch[i] = new

                                        * Grinch(tablero[filaAleatoria][4].getX(), -50, filaAleatoria, -1); } else {

                                            * zombieGrinch[i] = new Grinch(tablero[filaAleatoria][4].getX(),
entorno.alto()

                                        * + 50, filaAleatoria, fila); } zombieGrinch[i].setGenerado(true);

                                        * enemigosGenerados++; zombiesRestantes = Math.max(0, totalEnemigos -

                                                * enemigosGenerados); tiempo =
random.nextInt(maximoTiemporRegernacion -

                                        * minimioTiempoRegeneracion + 1) + minimioTiempoRegeneracion; break;
} }

                                        */

                                }

                        }


                        //Creacion de los zombies grinch dentro del tablero con sus respectivos moviementos
```

```
for(int i=0;i<zombieGrinch.length;i++) {
    if(zombieGrinch[i]!=null) {
        zombieGrinch[i].moverIzquierda();
        zombieGrinch[i].dibujarGrinch(entorno);

        boolean eliminado=false;
        //Si el grinch se sale de la pantalla
        if(zombieGrinch[i].getX()<0) {
            zombieGrinch[i]=null;
            eliminado=true;
        } else if(zombieGrinch[i].grinchMuerto()) {
            zombieGrinch[i]=null;
            zombiesEliminados++;
            eliminado=true;
        }

        if(eliminado) {
            continue;
        }
        Planta planta=obtenerPlantaEnPosicion(zombieGrinch[i].getX(),zombieGrinch[i].getY());
        if(planta!=null) {
            quitarPlanta(planta); //Elimina planta
            zombieGrinch[i].ralentizacion(60);//

        }
```

```
                    //Si el grinch entra en contacto con el regalo
                    for(int j=0;j<regalos.length;j++) {
                        if(regalos[j]!=null&&zombieGrinch[i].chocaConRegalo(regalos[j]))
{
                            juegoTerminado=true;
                            juegoGanado=false;
                            break;
                        }
                    }
                }

            }


            //Control y dibujo de los disparos, movimiento, dibujo y colision de los grinch
            for(int i=0;i<disparos.length;i++){
                Disparo d=disparos[i];
                if(d==null) continue;
                if(!d.isActivado()){
                    disparos[i]=null;
                    continue;
                }

                d.moverDerecha();
                d.dibujar(entorno);
        //Fuera de pantalla: eliminar zombie
                if(d.getX()>entorno.ancho()){
```

```
                    disparos[i]=null;
                    continue;
                }
        //Varifica colision con grinch
                boolean impacto=false;
                for(int j=0;j<zombieGrinch.length;j++){
                    Grinch g=zombieGrinch[j];
                    if(g==null) continue;

                    double dx=d.getX()-g.getX();
                    double dy=d.getY()-g.getY();
                    double distancia=Math.sqrt(dx*dx+dy*dy);
                    double alcance=d.getRadio()+g.getTamanio()/2.0;

                    if(distancia<alcance){
                        //Hay impacto
                        g.recibirDanio(d.getDanio());
                        disparos[i]=null;
                        impacto=true;
                        // Verificar si el grinch murio y aumerntar el contador de zombies
eliminados
                        if(g.grinchMuerto()) {
                            zombieGrinch[j]=null;
                            zombiesEliminados++;
                        }
                        break;
                    }
```

```
                    }
                    if(impacto) continue;
                }

//Dibujar regalos

for (int i = 0; i < regalos.length; i++) {
  if (regalos[i] != null) {
    regalos[i].dibujar(entorno);
  }
}
// dibujar plantas
for(int i=0;i<fila;i++){
            for(int j=0;j<columna;j++){
                    Planta p = plantas[i][j];
                    if(p!=null){
                            p.dibujar(entorno);
                            p.actualizarEstado();
                    }
            }
}
if(plantaPrevizualizada!=null){
            plantaPrevizualizada.dibujar(entorno);
}

//Movimiento de la planta seleccionada
```

```
for(int i=0;i<fila;i++) {
        for(int j=0;j<columna;j++) {
            Planta planta=plantas[i][j];
            if(planta != null && planta.isSeleccionada()) {
               int velocidad = 5;  // pixeles por tick

if(entorno.estaPresionada(entorno.TECLA_ARRIBA)&&planta.getY()-velocidad-planta.getDiametro()>barraSuperior.getAlturaBanda()) { // && !planta.tocaCasillaocupada(tablero)) {
//                  planta.moverArriba(velocidad);
                  planta.moverArriba(velocidad, tablero,plantas,fila,columna);
               }

if(entorno.estaPresionada(entorno.TECLA_ABAJO)&&planta.getY()+velocidad+planta.getDiametro()<entorno.alto()) {
                  planta.moverAbajo(velocidad,tablero,plantas,fila,columna);
               }

if(entorno.estaPresionada(entorno.TECLA_IZQUIERDA)&&planta.getX()-velocidad-planta.getDiametro()>0) {
                  planta.moverIzquierda(velocidad,tablero,plantas,fila,columna);
               }
                                    if(entorno.estaPresionada(entorno.TECLA_DERECHA) && planta.getX()+velocidad+planta.getDiametro() <entorno.ancho()) {
                  planta.moverDerecha(velocidad,tablero,plantas,fila,columna);
               }
               planta.dibujar(entorno);
            }
        }
    }
```

```
        //Verificacion de victoria o derrota
         if(zombiesEliminados>=totalEnemigos) {
                    juegoTerminado=true;
                    juegoGanado=true;
        }

        entorno.cambiarFont("Arial", 20, Color.RED);
        if(juegoTerminado) {
                if(juegoGanado) {
                        entorno.escribirTexto("¡Has ganado!", entorno.ancho()/2, entorno.alto()/2);
                }else {
                        entorno.escribirTexto("¡Has perdido!", entorno.ancho()/2, entorno.alto()/2);
                }
        }

}

        //Metodo auxiliar para el control de los disparos
        public void disparar(double x, double y){
                for(int i=0;i<disparos.length;i++){
                        if(disparos[i]==null){
                                disparos[i]=new Disparo(x, y, 5, 5, Color.YELLOW, 1);
```

```
                    break;
                }
            }
        }

        //Metodo auxiliar para obtener las coordenadas del casillero en base a las coordenadas X e Y
        public int[] cooodenadasCasillero(int x, int y){
            for(int i=0;i<fila;i++){
                for(int j=0;j<columna;j++){
                    Casillero c=tablero[i][j];
                    if(Math.abs(x-c.getX())<=c.getAncho()/2 &&
Math.abs(y-c.getY())<=c.getAlto()/2){
                        return new int[] {i,j};
                    }
                }
            }
            return null;
        }

        //Metodo auxiliar para deseleccionar todas las plantas del tablero
        public void deseleccionarTodas(){
            for(int i=0;i<fila;i++){
                for(int j=0;j<columna;j++){
                    if(plantas[i][j]!=null){
                        plantas[i][j].setSeleccionada(false);
                    }
                }
```

```
                }
        }

        private Planta obtenerPlantaEnPosicion(double x,double y) {
                for(int i=0;i<fila;i++) {
                        for(int j=0;j<columna;j++) {
                                Planta p= plantas[i][j];
                                if(p!=null&&Math.abs(p.getX()-x)<p.getDiametro()/2 &&
Math.abs(p.getY()-y)<p.getDiametro()/2) {
                                        return p;
                                }
                        }
                }
                return null;
        }

        public void quitarPlanta(Planta p) {
                for(int i=0;i<fila;i++) {
                        for(int j=0;j<columna;j++) {
                                if(plantas[i][j]==p) {
                                        plantas[i][j]=null;
                                        return;
                                }
                        }
                }
        }
```

```
public int contadorActivos(Grinch[] a) {
		int contador=0;
		for(int i=0;i<a.length;i++) {
			if(a[i]!=null) {
				contador++;
			}
		}
		return contador;
	}

	public void arrastrePlanta() {
		//Manjeo de arrastre de la carta de planta
			if(!arrastrando && entorno.estaPresionado(entorno.BOTON_IZQUIERDO)){
				int mouseX=entorno.mouseX();
				int mouseY=entorno.mouseY();
				if(carta.contienePunto(mouseX, mouseY)&&carta.puedeUsar()){
					arrastrando=true;
					plantaPrevizualizada=new Planta(mouseX,mouseY,true,this);
				} else{
						//click fuera de la carta
					deseleccionarTodas();
					for(int i=0;i<fila;i++){
						for(int j=0;j<columna;j++){
							Planta p=plantas[i][j];
							if(p!=null&&p.contienePunto(mouseX, mouseY)){
							//sellecciona unicamente esa planta para el
tablero
```

```
                                                    p.setSeleccionada(true);
                                                }
                                            }
                                        }
                                    }
                                }

                                //mientras mantenga presionado, actualiza la vista previa del mouse
                                if(arrastrando && entorno.estaPresionado(entorno.BOTON_IZQUIERDO)){
                                    int mouseX=entorno.mouseX();
                                    int mouseY=entorno.mouseY();
                                    if(plantaPrevizualizada!=null){
plantaPrevizualizada.moverPlanta(mouseX-plantaPrevizualizada.getX(), mouseY-plantaPrevizualizada.getY());
                                    }
                                }

                                //soltar la planta en el tablero y deteccion de casilletro libre u ocupado
                                if(arrastrando && !entorno.estaPresionado(entorno.BOTON_IZQUIERDO)){
                                    int mouseX=entorno.mouseX();
                                    int mouseY=entorno.mouseY();
                                    int[] ij= cooodenadasCasillero(mouseX,mouseY);
                                    if(ij!=null){
                                        int fi=ij[0];
                                        int co=ij[1];
                                        //Si hay un casillero libre,  se coloca la planta
                                        if(plantas[fi][co]==null){
```

```
                                                int px=tablero[fi][co].getX();
                                                int py=tablero[fi][co].getY();
                                                plantas[fi][co]=new     Planta(px,py,true,this);//Roseblade
por defecto
                                                carta.reiniciarCarga();
                                        }
                                }
                                plantaPrevizualizada=null;
                                arrastrando=false;
                        }
                }


                @SuppressWarnings("unused")
                public static void main(String[] args)
                {
                        Juego juego = new Juego();
                }
}
package juego;
import java.awt.Color;
import entorno.Entorno;


public class Casillero {
                private int x;
                private int y;
```

```
	private int ancho;
	private int alto;
	private boolean esVerde;
	boolean ocupada;

	public Casillero(int x, int y, int ancho, int alto, boolean esVerde){
		this.x=x;
		this.y=y;
		this.ancho=ancho;
		this.alto=alto;
		this.esVerde=esVerde;

	}
	public void dibujar(Entorno entorno){
		if(esVerde){
			entorno.dibujarRectangulo(x, y, ancho, alto, 0, Color.GREEN);
		}else{
			entorno.dibujarRectangulo(x, y, ancho, alto, 0, Color.GREEN.darker());
		}
	}

//	public boolean estaDisponible() {
//
//	}

	public int getX(){
		return x;
```

```
	}

	public int getY(){
		return y;
	}

	public int getAncho(){
		return ancho;
	}

	public int getAlto(){
		return alto;
	}
	public boolean isOcupada() {
		return ocupada;
	}
	public void setOcupada(boolean ocupada) {
		this.ocupada = ocupada;
	}

}

package juego;
import java.awt.Color;
import entorno.Entorno;
```

```
public class Casillero {

	private int x;

	private int y;

	private int ancho;

	private int alto;

	private boolean esVerde;

	boolean ocupada;


	public Casillero(int x, int y, int ancho, int alto, boolean esVerde){

		this.x=x;

		this.y=y;

		this.ancho=ancho;

		this.alto=alto;

		this.esVerde=esVerde;


	}

	public void dibujar(Entorno entorno){

		if(esVerde){

			entorno.dibujarRectangulo(x, y, ancho, alto, 0, Color.GREEN);

		}else{

			entorno.dibujarRectangulo(x, y, ancho, alto, 0, Color.GREEN.darker());

		}

	}


//	public boolean estaDisponible() {

//

//	}
```

```
	public int getX(){
		return x;
	}

	public int getY(){
		return y;
	}

	public int getAncho(){
		return ancho;
	}

	public int getAlto(){
		return alto;
	}
	public boolean isOcupada() {
		return ocupada;
	}
	public void setOcupada(boolean ocupada) {
		this.ocupada = ocupada;
	}

}
import java.awt.Color;
import entorno.Entorno;
```

```
public class BarraSuperior {

	private int alturaBanda=90; //Altura


	public void dibujar(Entorno entorno, int zombiesEliminados, int zombiesRestantes, int tiempo) {

		//Dibuja el rectangulo

entorno.dibujarRectangulo(entorno.ancho()/2,alturaBanda/2,entorno.ancho(),alturaBanda,0, Color.gray);

		entorno.cambiarFont("Arial", 18, Color.black);

		entorno.escribirTexto("Zombies eliminados: "+zombiesEliminados+"| Tiempo: "+tiempo+"s", 10, 40);

	}


	public int getAlturaBanda() {

		return alturaBanda;

	}


	public void setAlturaBanda(int alturaBanda) {

		this.alturaBanda = alturaBanda;

	}


}

import java.awt.Color;

import java.util.Random;


import entorno.Entorno;


public class Grinch {
```

```
private double x;  //Fila

private double y;  //Columna

private double velocidad;

private double vida;

private int tamanio;

private int tickLento=0;

private int filaObjetivo;

private boolean generado=false;



public Grinch(double x,double y, int filaObjetivo, int filaInicial) {

        // centro de las filas

        this.x=x;

        this.y=y;

        this.velocidad=1;

        this.vida=2;

        this.tamanio= 20;

        this.filaObjetivo=filaObjetivo;

}



public void dibujarGrinch(Entorno e) {

        e.dibujarCirculo(this.x, this.y, 30, Color.RED);

        e.dibujarCirculo(this.x, this.y, 25, Color.black);

        e.dibujarCirculo(this.x, this.y, 18, Color.red);

        e.dibujarCirculo(this.x, this.y, 12, Color.black);

        e.dibujarCirculo(this.x, this.y, 8, Color.red);
```

```
}



public void moverIzquierda() {   //Da el moviemiento de un zombie grinch
        if(tickLento>0) {
                tickLento--;
        }
        if(filaObjetivo!=-1&&(int)y!=filaObjetivo) {
                this.y+=(filaObjetivo>y) ? velocidad:-velocidad;
        }else {
                this.x-=velocidad;
        }
}


public boolean grinchMuerto() {//Indica cuando un grinch zombie esta muerto
        if(vida<=0) {
                return true;
        }
        return false;
}


public boolean chocaConRegalo(Regalos r) {
        if(r==null) {
                return false;
        }
        double semiladoRegalo=20.0;
```

```
		double semiradioGrinch=this.tamanio/2.0;
		double dx=this.x-r.getX();
		double dy=this.y-r.getY();
		double distancia=Math.sqrt(dx*dx+dy*dy);

		return distancia<=(semiradioGrinch+semiladoRegalo);
	}

	public void ralentizacion(int ticks) {
		tickLento=ticks;

	}

	public void recibirDanio(int danio) {
		this.vida-=danio;
	}

	public static void generacionZombiesAleatorios(Grinch[] zombieGrinch, Casillero[][] tablero, int fila, int columna, Entorno entorno, Random random) { //Pasar a grinch
		//Busca la primera posicion nula del arreglo para crear un nuevo zombie
		for(int i=0;i<zombieGrinch.length;i++) {
			if(zombieGrinch[i]==null) {

				int filaInicialValida=1; //La primera fila valida pra el juego
				if(fila<filaInicialValida) {
```

```
                        return;
                    }
                    int numeroFilasValidas=fila-filaInicialValida;

                    //Crea un nuevo zombie fuera de pantalla
                    int filaAleatoria=random.nextInt(numeroFilasValidas);
                    int yFila=tablero[filaAleatoria][columna-1].getY();
                    zombieGrinch[i]=new Grinch(entorno.ancho()+50,yFila,-1,filaAleatoria);
                    return;
                }
            }
        }

    public double getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public double getY() {
        return y;
```

```
	}



	public void setY(int y) {
		this.y = y;
	}



	public int getTamanio() {
		return tamanio;
	}



	public boolean isGenerado() {
		return generado;
	}



	public void setGenerado(boolean generado) {
		this.generado = generado;
	}

}
import java.awt.Color;
import entorno.Entorno;

public class Regalos {
```

```
	private double x;
	private double y;

	// Constructor que recibe la posición (x, y)
	public Regalos(double x, double y) {
		this.x = x;
		this.y = y;
	}

	public void dibujar(Entorno e) {
	  int lado = 40; // Tamaño
	  e.dibujarRectangulo(x, y, lado, lado, 0, Color.BLUE);
	}



	public double getX() {
		return x;
	}

	public double getY() {
		return y;
	}
}
import java.awt.Color;

import entorno.Entorno;
```

```
public class CartaPlanta {

	private int cartaRoseX;

	private int cartaRoseY;

	private int cartaAncho;

	private int cartaAlto;

	private int tiempoCarga=150; //5 segs a 60 FPS

   private int tiempoDisp=tiempoCarga;


	public CartaPlanta(int x, int y, int alto, int ancho, int tiempoDisp) {

		this.cartaRoseX=x;

		this.cartaRoseY=y;

		this.cartaAlto=alto;

		this.cartaAncho=ancho;

		this.tiempoDisp=tiempoDisp;

	}


	public void Dibujar(Entorno e) {

		e.dibujarRectangulo(cartaRoseX,cartaRoseY,cartaAncho,cartaAlto,0,Color.PINK);

		e.cambiarFont("Arial", 12, Color.black);

		e.escribirTexto("Rose Blade" + tiempoDisp, cartaRoseX, cartaRoseY);

		int barraAncho=(int)(cartaAncho*(1.0-(double)tiempoDisp/tiempoCarga));

e.dibujarRectangulo(cartaRoseX-cartaAncho/2+barraAncho,cartaRoseY+cartaAlto/2-5,barraAncho,10,0,Color.
GREEN);

	}


	public boolean puedeUsar() {
```

```
		return tiempoDisp>=tiempoCarga;

	}


	public void reiniciarCarga() {

		tiempoDisp=0;//Pone el tiempo a 0

	}


	public void actualizarTicks() {

		if(tiempoDisp<tiempoCarga) {

			tiempoDisp++;//Incrementa ticks en cada fotograma

		}

	}


	public boolean contienePunto(int px, int py) {

		return px>=cartaRoseX-cartaAncho/2 && px<=cartaRoseX+cartaAncho/2 &&
py>=cartaRoseY-cartaAlto/2 && py<=cartaRoseY+cartaAlto/2;

	}


	public int getCartaRoseX() {

		return cartaRoseX;

	}


	public void setCartaRoseX(int cartaRoseX) {

		this.cartaRoseX = cartaRoseX;

	}


	public int getCartaRoseY() {
```

```
		return cartaRoseY;
	}

	public void setCartaRoseY(int cartaRoseY) {
		this.cartaRoseY = cartaRoseY;
	}

	public int getCartaAncho() {
		return cartaAncho;
	}

	public void setCartaAncho(int cartaAncho) {
		this.cartaAncho = cartaAncho;
	}

	public int getCartaAlto() {
		return cartaAlto;
	}

	public void setCartaAlto(int cartaAlto) {
		this.cartaAlto = cartaAlto;
	}

	public int getTiempo() {
		return tiempoDisp;
	}
```

```
public void setTiempo(int tiempo) {

        this.tiempoDisp = tiempo;

}
```

Problemas encontrados:

Como apartado final, explicaremos un poco de las cosas en las que tuvimos problemas:

Al principio tuvimos varios problemas mientras hacíamos el juego. Por ejemplo, con la clase Casillero no sabíamos muy bien cómo dibujar los casilleros en la ventana, así que al principio usábamos **System.out.println** y se imprimían todos los colores y tamaños en la consola muchas veces, en vez de mostrarse en el juego.

También nos costó entender cómo funcionaban algunos métodos y cómo actualizar todo en el entorno gráfico. A veces los objetos se dibujaban en lugares equivocados o no se actualizaban bien. Otro problema que nos dificultaba el avance en el trabajo practico fue confundir los índices de filas y columnas, o no inicializar algún arreglo, lo que hacía que algunas plantas aparecieran fuera del tablero o no funcionaran.

Los disparos al principio tampoco detectaban bien a los enemigos, y los zombies a veces atravesaban todo sin recibir daño. La generación de enemigos y el control del tiempo también nos dio problemas al principio porque no aparecían como queríamos.

.