Universidad Nacional de General Sarmiento

Programación I –

Segundo Semestre 2024

Trabajo Práctico:

Al rescate de los gnomos

Docentes:
Leonor Gutiérrez
Ángel Juárez

Alumnos:
Mariana Brussa
Email: marianabrussaa@gmail.com
Legajo:35.121.900

Javier Cobos
Email: javier_kratos@outlook.com
Legajo:42.689.828

Natalia Evangelista
Email: evangelistanv@gmail.com
Legajo:30.724.603

Introducción

En el presente trabajo nos hemos dispuesto a realizar un juego, el cual se basa en un mundo mágico donde habitan unos gnomos de manera pacífica sobre islas flotantes. El conflicto reside en que estos seres son atacados por unas tortugas venenosas y además no saben defenderse por sí mismos. Es por ello, que contratan los servicios de un caballero fuerte e intrépido con algunos poderes especiales, con el propósito de que los salve.

Entonces podemos decir que el juego consiste en que el caballero Pep se desplace por todas las islas flotantes y rescate a los gnomos antes de que sean eliminados por las tortugas. Él cuenta con el poder del sol en sus manos, por ende, su arma más poderosa es lanzar bolas de fuego con las cuales puede eliminar a las tortugas, sin embargo, si es atacado (o sea existe colisión Pep-Tortuga) el caballero muere y se termina el juego. También debemos tener en cuenta que si la cantidad de 25 Gnomos son eliminados por las tortugas el jugador también pierde el juego. Además, debemos aclarar que los gnomos solo pueden ser rescatados en las islas inferiores, si Pep rescata a 10 gnomos antes de los 60 segundos sin caer o ser atacado por una tortuga el jugador gana la partida.

Descripción

Clase Islas:

La clase se encarga de modelar el objeto isla de manera visual dentro del entorno del juego. Sus variables de instancia son:

- private double x
- private double y
- private double ancho
- private double alto
- private double limiteIzq
- private double limiteDer
- private double limiteSup
- private double limiteInf

Constructor:

Es el encargado de inicializar la posición X e Y las dimensiones de alto y ancho y por último calcular todos los límites.

Métodos

DibujarIsla: Utilizando la clase entorno dibuja la imagen de un rectángulo con los parámetros pasados en el constructor.

Clase Tortuga Asesina:

La clase se encarga de modelar visualmente al objeto tortuga dentro del entorno, además de detectar las colisiones con las islas y una vez sucedido esto las mismas se mueven de izquierda a derecha.

Sus variables de instancia son:

- private double x
- private double y
- private double alto
- private double ancho
- private double velocidad
- private double limiteIzq
- private double limiteDer
- private double limiteSup
- private double limiteInf

Constructor:

Inicializa las dimensiones ancho, alto, establece la velocidad, y genera coordenadas aleatorias para que la tortuga no caiga en la isla donde se encuentra la casa de los gnomos. Además, se establece una pequeña distancia para que no caigan todas o algunas en una misma isla.

Métodos

DibujarTortuga: Dibuja la tortuga en el entorno gráfico utilizando su imagen y sus coordenadas.

Movimiento: Controla el movimiento de la tortuga. Se mueve hacia abajo hasta que colisiona con una isla, momento en el que comienza a moverse sobre la isla con el método moverSobreIsla.

ColisionTortugaIsla: Verifica si la tortuga colisiona con alguna isla, basándose en la superposición de límites de la tortuga con la isla.

ActualizarLimites: Actualiza los límites de la tortuga después de que su posición cambie.

MoverSobreIsla: Maneja el movimiento de la tortuga sobre una isla, invirtiendo la dirección si alcanza los bordes.

Clase Gnomos:

La clase gnomos se encarga de modelar visualmente el objeto dentro del entorno y detecta las colisiones, ya sea con las islas o tortugas y su comportamiento (si está cayendo o se desplaza de izquierda a derecha si en todo caso ha colisionado con una isla). Además, determina si ellos salen de la pantalla.

Sus variables de instancia son:

- private double x;
- private double y;
- private double alto;
- private double ancho;
- private double velocidadX
- private double velocidadY
- private boolean cayendo
- private boolean haSalidoDePrimeraIsla
- private Random random

Constructor:

Inicializa las coordenadas, dimensiones y estado del gnomo. También carga la imagen

Métodos

DibujarGnomos: Dibuja el gnomo en el entorno.

movimientoGnomos: Controla el movimiento del gnomo. Si está cayendo, lo mueve hacia abajo; si no está cayendo, lo mueve horizontalmente y verifica si ha salido de la primera isla.

ColisionaConIsla: Verifica si el gnomo está colisionando con una isla.

Caer: Permite al gnomo caer si está en estado de caída.

ColisionConTortuga: Verifica si el gnomo colisiona con alguna tortuga.

SalioDePantalla: Determina si el gnomo ha salido del área visible de la pantalla.

Clase Jugador Pep:

La clase jugador Pep modela el objeto visualmente dentro del entorno y además proporciona el control de su movimiento (izquierda-derecha), su salto, su estado (puede estar saltando o estar cayendo), gestiona las colisiones (con islas, tortugas o gnomos).

Sus variables de instancia son:

- private double x
- private double y
- private double alto
- private double ancho
- private double velocidadY
- private boolean saltando
- private double saltoY
- private char direccionPep
- private int direImagen

Constructor:

Inicializa las coordenadas, dimensiones y estado del jugador, y carga las imágenes.

Métodos

DibujarPep: Dibuja el jugador en el entorno.

Caminar: Maneja el movimiento horizontal del jugador según la tecla presionada.

IniciarSalto: Inicia el salto, estableciendo la velocidad inicial hacia arriba si no está ya saltando.

Saltar: Actualiza la posición vertical del jugador si está en salto y aplica gravedad.

Caer: Permite que el jugador caiga cuando no está sobre una isla.

PuedoSaltar: Verifica si el jugador puede saltar, asegurándose de que si colisionó con una isla puede saltar. Además, comprueba si la posición vertical del jugador (y) menos la mitad de su altura (alto / 2) es menor o igual a 0. Esto implica que, si el jugador está muy cerca del borde superior del entorno, no debería poder saltar.

ColisionIsla: Comprueba si el borde inferior del jugador es mayor o igual que el borde superior de la isla, y al mismo tiempo, que la posición del jugador está por encima del borde superior de la isla. Además, comprueba que el borde derecho del jugador está a la derecha del borde izquierdo de la isla, y que el borde izquierdo del jugador está a la izquierda del borde derecho de la isla.

ColisionaConTortuga: Comprueba la colisión con una tortuga.

colisionaConGnomos: Verifica la colisión con un gnomo.

Clase Disparo:

La clase disparo modela el objeto disparo dentro del entorno, se encarga de dibujarlo, de detectar las colisiones con las tortugas, determina su estado (disparo activo o inactivo) y movimiento (velocidad y dirección del disparo).

Sus variables de instancia son:

- private double x
- private double y
- private double alto
- private double ancho
- private double velocidad
- private boolean estado
- private char dirección

Constructor:

Inicializa las coordenadas, dimensiones, estado del disparo (false) , la dirección y velocidad.

Métodos

Disparar: El método disparar se encarga de actualizar la posición del disparo en función de la dirección en la que se encuentra Pep.

DibujarDisparo: Dibuja el disparo en el entorno.

FueraDePantalla: Verifica si el disparo ha salido de los límites de la pantalla

ColisionaConTortuga: Comprueba si el disparo colisiona con alguna tortuga. Si hay colisión, elimina la tortuga de la lista (null).

Clase Juego:

La clase juego contiene todas las variables de instancias, los arreglos y demás. En el constructor se inicializan las anteriores mencionadas con sus valores iniciales para luego ser llamadas dentro del tick. En el tick comenzamos el ciclo del juego de manera fluida

variables de instancia:

- private Disparo disparos
- private Islas[] isla
- private Gnomos[] gnomosEnJuego

- private JugadorPep pep
- private TortugaAsesina[] tortugasEnJuego
- private int tiempoParaSiguienteGnomo
- private int gnomosRescatados
- private int gnomosPerdidos
- private int tortugasEliminadas
- private int tiempo
- private boolean enMenu
- private boolean comienzoDelJuego
- private boolean juegoEnPausa
- private boolean ganoJuego
- private boolean perdioJuego
- private Image fondo
- private Image gano
- private Image perdio
- private Image menu
- private Image casa
- private Clip sonidoMenu
- private Clip sonidoFondo
- private Clip sonidoPerdio
- private Clip sonidoGano

Constructor:

Aca se inicializan las variables con sus posiciones iniciales y todo lo que sea necesario para que el ciclo del juego tenga su fluidez.

Métodos:

juego(): Es la clase en donde se crean e inicializan los objetos y los valores que se van a utilizar para representarlos en la interfaz.

tick(): Comienzan a generarse todos los objetos para que el ciclo del juego funcione correctamente. Es el más importante donde todo tiene que tener un correcto orden y relación.

Problemas presentados en la construccion del juego:

El primer problema que tuvimos fue con el movimiento de las tortugas una vez que colisionaban con la isla. El problema se resolvió creando el método moverSobreIsla. Luego se nos presentó el problema para que los gnomos cayeran y cambiarán de dirección cuando caían y salían de la primera isla. El problema se resolvió usando random en el método de movimiento cada vez que caía y colisionaba con una isla su dirección era aleatoria. Después nos encontramos con que el pep disparaba al momento del salto lo cual no era correcto. En la clase principal usamos un booleano llamado estaSobreUnaIsla que se ponía en true si había colisión con una isla, esa misma bandera nos sirvió para resolver el último problema de que pep solo podía disparar si estaba sobre la isla.

Conclusiones:

Para finalizar, hemos intentado implementar todo lo aprendido en clase, desde confeccionar las diversas clases con sus variables de instancia y métodos necesarios para cumplir con cada funcionalidad del juego. Ha sido un trabajo muy duro, pero con la satisfacción de la tarea cumplida y aprendizaje adquirido.

Implementacion:

Código fuente.

```
package juego;


import java.awt.Color;
import java.awt.Image;


import entorno.Entorno;
import entorno.Herramientas;
```

```
public class Disparo {
	private Image imagen;
	private double x;
	private double y;
	private double alto;
	private double ancho;
	private double velocidad;
	private boolean estado;
	private char direccion;

	public Disparo(double x, double y, double direccionPep) {
		this.x = x;
		this.y = y;
		this.alto = 20;
		this.ancho = 20;
		this.velocidad = 0.5;
		this.estado = false;
		this.direccion = 0;
		this.imagen =
Herramientas.cargarImagen("Imagenes/disparo.png").getScaledInstance(45, 25,
Image.SCALE_SMOOTH);
	}

	public void disparar(char direccionPep, Entorno e) {
		if (direccionPep == e.TECLA_DERECHA) {
			this.x += this.velocidad * Math.cos(direccionPep) + 4;
		} else if (direccionPep == e.TECLA_IZQUIERDA) {
			this.x -= this.velocidad * Math.cos(direccionPep) + 4;
```

```
            }
        }

        public void dibujarDisparo(Entorno e) {
            e.dibujarRectangulo(x, y, ancho, alto, 0, new Color(0, 0, 0, 0));
            e.dibujarImagen(imagen, x, y, 0);
        }

        public boolean fueraDePantalla() {
            int anchoPantalla = 800;
            int altoPantalla = 600;

            // verifico si el disparo está fuera de la pantalla
            if (this.x < 0 || this.x > anchoPantalla || this.y < 0 || this.y > altoPantalla) {
                return true;
            }
            return false;
        }

        public boolean colisionaConTortuga(TortugaAsesina[] tortugas) {
            // Recorre todas las tortugas en el array
            for (int i = 0; i < tortugas.length; i++) {
                TortugaAsesina tortuga = tortugas[i];
                if (tortuga != null) {
                    if (this.x >= tortuga.getX() - tortuga.getAnchoTortuga() / 2
                            && this.x <= tortuga.getX() +
tortuga.getAnchoTortuga() / 2
```

```
                                        && this.y >= tortuga.getY() -
tortuga.getAltoTortuga() / 2
                                        && this.y <= tortuga.getY() +
tortuga.getAltoTortuga() / 2) {
                                    tortugas[i] = null; // Elimina la tortuga asignando null
                                    return true;
                                }
                            }
                        }
                        return false;

                }

                public double getX() {
                        return x;
                }

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

                public double getY() {
                        return y;
                }

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

```
public Image getImagen() {
        return imagen;
}

public void setImagen(Image imagen) {
        this.imagen = imagen;
}

public double getAlto() {
        return alto;
}

public void setAlto(double alto) {
        this.alto = alto;
}

public double getAncho() {
        return ancho;
}

public void setAncho(double ancho) {
        this.ancho = ancho;
}

public double getVelocidad() {
        return velocidad;
}
```

```
	public void setVelocidad(double velocidad) {
		this.velocidad = velocidad;
	}

	public boolean isEstado() {
		return estado;
	}

	public void setEstado(boolean estado) {
		this.estado = estado;
	}

	public char getDireccion() {
		return direccion;
	}

	public void setDireccion(char direccion) {
		this.direccion = direccion;
	}

}

package juego;

import java.awt.Color;
import java.awt.Image;
import java.util.Random;
```

```
import entorno.Entorno;
import entorno.Herramientas;

public class Gnomos {
	private Image imagen;
	private double x;
	private double y;
	private double alto;
	private double ancho;
	private double velocidadX; // Velocidad horizontal
	private double velocidadY; // Velocidad vertical (caída)
	private boolean cayendo; // Estado de caída
	private boolean haSalidoDePrimeraIsla; // Control de salida desde la primera isla
	private Random random;

	// Constructor
	public Gnomos(double x, double y, double ancho, double alto) {
		this.x = x; // Posición inicial en x
		this.y = y; // Posición inicial en y
		this.alto = alto;
		this.ancho = ancho;
		this.velocidadX = 0; // Inicialmente, el gnomo no se mueve horizontalmente
		this.velocidadY = 0.5; // Velocidad de caída inicial
		this.cayendo = true; // El gnomo comienza cayendo
		this.haSalidoDePrimeraIsla = false; // Inicialmente no ha salido de la
primera isla
		this.random = new Random();
```

```
            this.imagen = Herramientas.cargarImagen("Imagenes/gnomo.png").getScaledInstance(30, 35, Image.SCALE_SMOOTH);
        }

        // Método para dibujar los gnomos en la pantalla
        public void dibujarGnomos(Entorno entorno) {
            entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));
            entorno.dibujarImagen(imagen, this.x, this.y, 0);
        }

        // Movimiento de los gnomos controlando colisiones con las islas
        public void movimientoGnomos(Islas[] islas) {
            if (cayendo) {
                // Si el gnomo está cayendo, se mueve hacia abajo
                this.y += velocidadY;

                // Verificar colisión con cada isla
                for (Islas isla : islas) {
                    if (colisionaConIsla(isla)) {
                        cayendo = false;
                        velocidadY = 0; // Detiene la caída
                        velocidadX = (random.nextBoolean()) ? 0.1 : -0.1; // Movimiento horizontal aleatorio

                        break; // Detener al colisionar con una isla
                    }
                }
            }
```

```
            if (!cayendo) {
                // Movimiento horizontal si no está cayendo
                this.x += velocidadX;

                if (!haSalidoDePrimeraIsla) {
                    // Verificar si el gnomo ha salido de la primera isla
                    Islas primeraIsla = islas[0];
                    if (this.x <= primeraIsla.getLimiteIzq() - 10 || this.x >= primeraIsla.getLimiteDer() + 10) {
                        haSalidoDePrimeraIsla = true; // El gnomo ha salido de la primera isla
                    }
                }

                // después de salir de la primera isla
                if (haSalidoDePrimeraIsla) {
                    for (Islas isla : islas) {
                        if (colisionaConIsla(isla)) {
                            // Si llega al borde de cualquier isla, comienza a caer
                            if (this.x < isla.getLimiteIzq() - 10 || // le doy un margen para que el gnomo caiga bien al

                                // borde de la isla
                                    this.x > isla.getLimiteDer() + 10) {
                                // si el gnomo ha salido de los límites de la isla
                                cayendo = true; // Activar el estado de caída
                                velocidadY = 0.5;
```

```
                                                    break; // No seguir verificando otras
islas
                                                }
                                            }
                                        }
                                    }
                                }

                            }

                            public boolean colisionaConIsla(Islas isla) {
                                // Verifica si el gnomo está justo en el borde superior de la isla.
                                return (this.y + this.ancho / 2 == isla.getLimiteSup()) && (this.x + this.alto /
2 >= isla.getLimiteIzq())
                                        && (this.x - this.alto / 2 <= isla.getLimiteDer());
                            }

                            public void caer(Islas isla) {
                                if (cayendo && colisionaConIsla(isla)) {
                                    this.y += velocidadY; // Movimiento hacia abajo
                                }
                            }

                            public boolean colisionConTortuga(TortugaAsesina[] tortugas) {
                                for (TortugaAsesina tortuga : tortugas) {
                                    if (tortuga != null) { // Verifico que no haya tortugas nulas en el array
                                        // Verificar si gnomo y la tortuga colisionan
                                        if (this.x < tortuga.getLimiteDer() && this.x + this.ancho >
tortuga.getLimiteIzq()
```

```
                                        && this.y < tortuga.getLimiteInf() && this.y
+ this.alto > tortuga.getLimiteSup()) {
                                    return true; // Hay colisión
                                }
                            }
                        }
                        return false; // No hay colisión
                }

                public boolean salioDePantalla() {
                        double gnomoY = getY();

                        if (gnomoY > 600) {
                                return true; // El gnomo ha salio de la pantalla
                        }
                        return false; // El gnomo está dentro de la pantalla
                }

                // Getters y setters
                public double getX() {
                        return x;
                }

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

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

```
}

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

public Image getImagen() {
        return imagen;
}

public void setImagen(Image imagen) {
        this.imagen = imagen;
}

public double getAlto() {
        return alto;
}

public void setAlto(double alto) {
        this.alto = alto;
}

public double getAncho() {
        return ancho;
}

public void setAncho(double ancho) {
        this.ancho = ancho;
```

```
	}

	public void setCayendo(boolean cayendo) {
		this.cayendo = cayendo;
	}
}

package juego;

import java.awt.Color;
import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class Islas {
	private Image imagen;
	private double x;
	private double y;
	private double ancho;
	private double alto;
	private double limiteIzq;
	private double limiteDer;
	private double limiteSup;
	private double limiteInf;

	public Islas(double x, double y, double ancho, double alto) {
		this.x = x;
```

```
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.limiteIzq = x - (ancho / 2); // Calcula el límite izquierdo
        this.limiteDer = x + (ancho / 2); // Calcula el límite derecho
        this.limiteSup = y - (alto / 2); // Calcula el límite superior
        this.limiteInf = y + (alto / 2); // Calcula el límite inferior
        this.imagen = Herramientas.cargarImagen("Imagenes/isla.png").getScaledInstance(95, 30, Image.SCALE_SMOOTH);
    }

    public void dibujarIsla(Entorno entorno) {
        entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));

        entorno.dibujarImagen(imagen, x, y - 10, 0);
    }

    public Image getImagen() {
        return imagen;
    }

    public void setImagen(Image imagen) {
        this.imagen = imagen;
    }

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

```
}

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

public double getY() {
        return y;
}

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

public double getAlto() {
        return alto;
}

public void setAlto(double alto) {
        this.alto = alto;
}

public double getAncho() {
        return ancho;
}

public void setAncho(double ancho) {
        this.ancho = ancho;
```

```
}

public double getLimiteIzq() {
        return limiteIzq;
}

public void setLimiteIzq(double limiteIzq) {
        this.limiteIzq = limiteIzq;
}

public double getLimiteDer() {
        return limiteDer;
}

public void setLimiteDer(double limiteDer) {
        this.limiteDer = limiteDer;
}

public double getLimiteSup() {
        return limiteSup;
}

public void setLimiteSup(double limiteSup) {
        this.limiteSup = limiteSup;
}

public double getLimiteInf() {
        return limiteInf;
```

```
	}

	public void setLimiteInf(double limiteInf) {
		this.limiteInf = limiteInf;
	}
}

package juego;

import java.awt.Color;
import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class JugadorPep {
	private Image[] imagen;
	private double x;
	private double y;
	private double alto;
	private double ancho;
	private double velocidadY;
	private boolean saltando;
	private double saltoY = 0.5;
	private char direccionPep;
	private int direImagen;

	public JugadorPep(double x, double y, double ancho, double alto) {
```

```
        this.imagen = new Image[2];
        this.x = x;
        this.y = y;
        this.alto = 60;
        this.ancho = 20.0;
        this.velocidadY = 0;
        this.saltando = false;
        this.direccionPep = '0';
        imagen[0] = Herramientas.cargarImagen("Imagenes/pepDer.png").getScaledInstance(25, 60, Image.SCALE_SMOOTH);
        imagen[1] = Herramientas.cargarImagen("Imagenes/pepIzq.png").getScaledInstance(25, 60, Image.SCALE_SMOOTH);

        // le sume un int al alto y ancho para que la imagen sea mas grande
        this.direImagen = 0; // Imagen inicial
    }

    public void DibujarPep(Entorno entorno) {
        entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0, new Color(0, 0, 0, 0));
        entorno.dibujarImagen(imagen[direImagen], this.x, this.y, 0);
    }

    public void caminar(Entorno entorno, char teclaPresionada) {

        if (teclaPresionada == entorno.TECLA_DERECHA) {
            x++;
            direccionPep = entorno.TECLA_DERECHA;
```

```
                direImagen = 0;
            } else if (teclaPresionada == entorno.TECLA_IZQUIERDA) {
                x--;
                direccionPep = entorno.TECLA_IZQUIERDA;
                direImagen = 1;

            }
        }

        public boolean tieneDireccion() {
            return this.direccionPep != '0';
        }

        public void iniciarSalto() {
            if (!saltando) {
                this.saltando = true;
                this.velocidadY = -10; // Velocidad inicial hacia arriba

            }
        }

        public void saltar() {
            if (saltando) {
                y += velocidadY; // Actualiza la posición vertical según la velocidad
actual
                velocidadY += saltoY; // caida con gravedad

                // Detener el salto cuando empieza a caer
                if (velocidadY > 0) {
```

```
                        saltando = false;
                    }
                }
            }

            public void caer() {
                if (!saltando) {
                    y += 3;

                    // Si no está saltando, caer de manera gradual
                }
            }

            public boolean puedoSaltar(Islas[] isla) {
                if (y - alto / 2 <= 0) {
                    return false;
                }
                for (Islas i : isla) {
                    if (colisionIsla(i)) {
                        return true;
                    }
                }
                return false;
            }

            public boolean colisionIsla(Islas i) {
                return (this.y + (this.alto / 2) >= i.getY() - (i.getAlto() / 2) && this.y <=
i.getY() - (i.getAlto() / 2))
                        && this.x + (this.ancho / 2) > i.getX() - (i.getAncho() / 2)
```

```
                && this.x - (this.ancho / 2) < i.getX() + (i.getAncho() / 2);
    }

    public boolean colisionaConTortuga(TortugaAsesina t) {
        return (this.x + (this.ancho / 2) > t.getLimiteIzq() && this.x - (this.ancho / 2)
< t.getLimiteDer()
                && this.y + (this.alto / 2) > t.getLimiteSup() && this.y -
(this.alto / 2) < t.getLimiteInf());
    }

    public boolean colisionaConGnomos(Gnomos g) {
        return (this.y + (this.alto / 2) >= g.getY() - (g.getAlto() / 2) && this.y <=
g.getY() - (g.getAlto() / 2))
                && this.x + (this.ancho / 2) > g.getX() - (g.getAncho() / 2)
                && this.x - (this.ancho / 2) < g.getX() + (g.getAncho() / 2);
    }

    public boolean getSaltando() {
        return saltando;
    }

    public void setSaltando(boolean saltando) {
        this.saltando = saltando;
    }

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

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

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

public void setPosicion(double y) {
        this.y = y;
}

public char getDireccionPep() {
        return direccionPep;
}

public void setDireccionPep(char direccionPep) {
        this.direccionPep = direccionPep;
}

public double getAlto() {
        return alto;
}

public void setAlto(double alto) {
        this.alto = alto;
}
```

```
	public double getAncho() {
		return ancho;
	}

	public void setAncho(double ancho) {
		this.ancho = ancho;
	}

}
```

```
package juego;

import java.awt.Color;
import java.awt.Image;
import java.util.Random;

import entorno.Entorno;
import entorno.Herramientas;

public class TortugaAsesina {
	private Image imagen;
	private double x;
	private double y;
	private double alto;
	private double ancho;
	private double velocidad;
	private double limiteIzq;
	private double limiteDer;
```

```
        private double limiteSup;
        private double limiteInf;


        public TortugaAsesina(double ancho, double alto, double velocidad, TortugaAsesina[] tortugasExistentes,
                        int tortugasCreadas) {
                this.alto = alto;
                this.ancho = ancho;
                Random tortugasRandom = new Random();
                boolean posicionValida = false;
                double distanciaMinima = 80; // Distancia mínima entre tortugas
                this.velocidad = 0.3;
                // Generar coordenadas aleatorias hasta que no estén cerca de otra tortuga y no
                // caigan en la isla
                while (!posicionValida) {
                        this.x = tortugasRandom.nextInt(800) + 1; // Genera coordenadas aleatorias en X
                        this.y = tortugasRandom.nextInt(100) + 1; // Genera coordenadas aleatorias en Y
                        posicionValida = true;

                        // para que no caigan en la primer isla
                        if (this.x >= 300 && this.x <= 500) {
                                posicionValida = false;
                                continue;
                        }

                        // Verificar que no esté muy cerca de las tortugas existentes
                        for (int i = 0; i < tortugasCreadas; i++) {
```

```
                        TortugaAsesina otraTortuga = tortugasExistentes[i];
                        double distancia = Math
                                .sqrt(Math.pow(this.x - otraTortuga.getX(), 2)
+ Math.pow(this.y - otraTortuga.getY(), 2));
                        if (distancia < distanciaMinima) {
                            posicionValida = false; // Si la distancia es menor que
la mínima, buscar nueva posición
                            break;
                        }
                    }
                    this.imagen                                                    =
Herramientas.cargarImagen("Imagenes/tortuga.png").getScaledInstance(40, 40,
                            Image.SCALE_SMOOTH);

                }

                this.velocidad = tortugasRandom.nextDouble() * 0.3 + 0.3;

            }

            public void DibujarTotuga(Entorno entorno) {
                entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0.0, new
Color(0, 0, 0, 0));
                entorno.dibujarImagen(imagen, x, y, 0.0);
            }

            public void movimiento(Islas[] isla) {
                // Si la tortuga no colisiona con ninguna isla, sigue moviéndose hacia abajo
                if (!colisionTortugaIsla(isla)) {
                    this.y += this.velocidad; // Mover hacia abajo
```

```
                        actualizarLimites();
                        // Verificar si la tortuga llega al borde inferior del entorno
                        if (this.y > 580) { // el límite inferior del entorno es 580
                                this.y = 580; // Mantenerla dentro del límite
                        }
                } else {
                        // Si la tortuga colisiona con una isla, moverse sobre ella de izquierda
a
                        // derecha
                        moverSobreIsla(isla);
                }
        }

        public boolean colisionTortugaIsla(Islas[] isla) {
                // Verificar si los bordes de la tortuga y la isla se superponen
                for (Islas is : isla) {
                        if ((is.getLimiteDer() > this.limiteIzq) && (is.getLimiteIzq() <
this.limiteDer)
                                        && (is.getLimiteInf() > this.limiteSup) &&
(is.getLimiteSup() < this.limiteInf)) {
                                return true; // Hay colisión
                        }
                }
                return false; // No hay colisión
        }

        public void actualizarLimites() {
                this.limiteIzq = this.x - (this.ancho / 2);
                this.limiteDer = this.x + (this.ancho / 2);
```

```
            this.limiteSup = this.y - (this.alto / 2);
            this.limiteInf = this.y + (this.alto / 2);
        }

        public void moverSobreIsla(Islas[] isla) {
            for (Islas is : isla) {
                // Verificar si la tortuga está sobre la isla
                if (this.limiteDer > is.getLimiteIzq() && this.limiteIzq < is.getLimiteDer()
                        && (Math.abs(this.limiteInf - is.getLimiteSup()) < 5)) { // La tortuga está sobre la isla
                    // Mover la tortuga de izquierda a derecha
                    this.x += this.velocidad;

                    // Actualizar los límites de la tortuga
                    actualizarLimites();

                    // Si la tortuga llega al borde derecho de la isla, invertir la dirección
                    if (this.limiteDer > is.getLimiteDer()) {
                        this.x = is.getLimiteDer() - (this.ancho / 2); // Ajustar posición al borde
                        this.velocidad = -this.velocidad; // Invertir la dirección (ahora se mueve a la izquierda)
                    }

                    // Si la tortuga llega al borde izquierdo de la isla, invertir la dirección
                    if (this.limiteIzq < is.getLimiteIzq()) {
                        this.x = is.getLimiteIzq() + (this.ancho / 2); // Ajustar posición al borde
```

```
                                    this.velocidad = -this.velocidad; // Invertir la dirección (ahora se mueve a la derecha)
                                }

                                break; // Romper el bucle ya que la tortuga está moviéndose sobre una isla
                            }
                        }
                    }

                    public double getX() {
                        return x;
                    }

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

                    public double getY() {
                        return y;
                    }

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

                    public Image getImagenTotuga() {
                        return imagen;
                    }
```

```
public void setImagenTortuga(Image imagen) {
        this.imagen = imagen;
}

public double getAltoTortuga() {
        return alto;
}

public void setAltoTortuga(double alto) {
        this.alto = alto;
}

public double getAnchoTortuga() {
        return ancho;
}

public void setAnchoTortuga(double ancho) {
        this.ancho = ancho;
}

public double getLimiteIzq() {
        return limiteIzq;
}

public void setLimiteIzq(double limiteIzq) {
        this.limiteIzq = limiteIzq;
}
```

```
	public double getLimiteDer() {
		return limiteDer;
	}

	public void setLimiteDer(double limiteDer) {
		this.limiteDer = limiteDer;
	}

	public double getLimiteSup() {
		return limiteSup;
	}

	public void setLimiteSup(double limiteSup) {
		this.limiteSup = limiteSup;
	}

	public double getLimiteInf() {
		return limiteInf;
	}

	public void setLimiteInf(double limiteInf) {
		this.limiteInf = limiteInf;
	}

}

package juego;
```

```
import java.awt.Color;
import java.awt.Image;

import javax.sound.sampled.Clip;

import entorno.Entorno;
import entorno.Herramientas;
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
	// ...
	private Disparo disparos;
	private Islas[] isla;
	private Gnomos[] gnomosEnJuego;
	private JugadorPep pep;
	private TortugaAsesina[] tortugasEnJuego;
	private int tiempoParaSiguienteGnomo; // Contador de tiempo para el desfase
	private int gnomosRescatados;
	private int gnomosPerdidos;
	private int tortugasEliminadas;
	private int tiempo;
	private boolean enMenu;
	private boolean comienzoDelJuego;
```

```
	private boolean juegoEnPausa;
	private boolean ganoJuego;
	private boolean perdioJuego;
	private Image fondo;
	private Image gano;
	private Image perdio;
	private Image menu;
	private Image casa;
	private Clip sonidoMenu;
	private Clip sonidoFondo;
	private Clip sonidoPerdio;
	private Clip sonidoGano;

	Juego() {
		// Inicializa el objeto entorno
		this.entorno = new Entorno(this, "Al rescate de los Gnomos- Grupo 6- p1",
800, 600);
		this.fondo = Herramientas.cargarImagen("Imagenes/fondo.png");
		this.gano = Herramientas.cargarImagen("Imagenes/winner.gif");
		this.perdio = Herramientas.cargarImagen("Imagenes/game-over.gif");
		this.menu = Herramientas.cargarImagen("Imagenes/menu.gif");
		this.casa = Herramientas.cargarImagen("Imagenes/casa.png");

		this.enMenu = true;
		this.comienzoDelJuego = false;
		this.juegoEnPausa = true;
		this.ganoJuego = false;
		this.perdioJuego = false;
```

```
this.sonidoMenu = Herramientas.cargarSonido("Sonidos/menu.wav");
this.sonidoFondo = Herramientas.cargarSonido("Sonidos/juego.wav");
this.sonidoPerdio = Herramientas.cargarSonido("Sonidos/perdio.wav");
this.sonidoGano = Herramientas.cargarSonido("Sonidos/gano.wav");

// Inicializar lo que haga falta para el juego
// ...
pep = new JugadorPep(180, 455, 20, 60);
gnomosEnJuego = new Gnomos[8];
tortugasEnJuego = new TortugaAsesina[10];

gnomosRescatados = 0;
gnomosPerdidos = 0;
tiempoParaSiguienteGnomo = 0;
tortugasEliminadas = 0;
tiempo = 0;

isla = new Islas[15]; // número total de islas
int[] rectanguloN = { 1, 2, 3, 4, 5 }; // Cantidad de islas por fila
int[] xInicio = { 400, 300, 160, 90, 1 }; // Posiciones iniciales en x para cada
fila
int[] yFila = { 100, 200, 300, 400, 500 }; // Posiciones en y para cada fila
int anchoRectangulo = 100;
int distancia = 200;
int k = 0; // posición en el arreglo de islas

for (int i = 0; i < rectanguloN.length; i++) { // Recorremos las filas
        int x = xInicio[i];
        int y = yFila[i];
```

```
                int numRec = rectanguloN[i];

                for (int j = 0; j < numRec; j++) { // Recorremos las islas por fila
                    this.isla[k] = new Islas(x, y, anchoRectangulo, 30);
                    x += distancia; // Mover la siguiente isla a la derecha
                    k++; // Incrementar el índice para la siguiente isla
                }
            }

            // 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() {
            // Procesamiento de un instante de tiempo
            if (enMenu && !comienzoDelJuego && juegoEnPausa) {// imagen menu
del juego
                entorno.dibujarImagen(menu, 400, 300, 0);
                entorno.cambiarFont("arial", 18, Color.BLACK);
                entorno.escribirTexto("Bienvenidos al juego", 8, 20);
                entorno.escribirTexto("reglas del juego", 250, 20);
                entorno.escribirTexto("comandos", 475, 20);
```

```
                entorno.escribirTexto("lo dejo por si quieren salir ", 680, 20);// para
que el tiempo sea en segundos divido

                                                // por 1000


                sonidoMenu.start();
            }
            tiempo=entorno.tiempo()/1000;
            if(tiempo>=60){
                perdioJuego=true;
            }
            if (perdioJuego == true) {
                entorno.dibujarImagen(perdio, entorno.ancho() / 2, entorno.alto() / 2,
0);
                sonidoFondo.stop();
                sonidoPerdio.start();

                return;

            }
            if (ganoJuego == true) {
                entorno.dibujarImagen(gano, entorno.ancho() / 2, entorno.alto() / 2,
0);
                sonidoFondo.stop();
                sonidoGano.start();

                return;
            }
            if (entorno.sePresiono(entorno.TECLA_ENTER)) {// comienza el juego
                enMenu = false;
```

```
                comienzoDelJuego = true;
                juegoEnPausa = false;
                sonidoMenu.stop();


        }
        if (comienzoDelJuego = true && !enMenu && !ganoJuego) {
                entorno.dibujarImagen(fondo, entorno.ancho() / 2, entorno.alto() / 2,
0);
                sonidoFondo.start();
        }
        if (!juegoEnPausa) {
                entorno.dibujarImagen(casa, 390, 40, 0);
                entorno.cambiarFont("arial", 18, Color.BLACK);
                entorno.escribirTexto("Enemigos eliminados :" +
tortugasEliminadas, 8, 20);
                entorno.escribirTexto("Gnomos perdidos :" + gnomosPerdidos, 475,
40);
                entorno.escribirTexto("Gnomos salvados :" + gnomosRescatados,
475, 20);
                entorno.escribirTexto("Tiempo: " + tiempo, 700, 20);// para que el
tiempo sea en segundos divido por 1000



                // Dibujar las islas
                for (Islas i : isla) {
                        if (i != null) {
                                i.dibujarIsla(entorno);
                        }
                }
                // Dibujo y movimiento de Pep
```

```
                if (pep != null) {
                        if (!pep.getSaltando() && pep.puedoSaltar(isla)) { // moverse si no está saltando
                                if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
                                        pep.caminar(entorno, entorno.TECLA_IZQUIERDA);
                                } else if (entorno.estaPresionada(entorno.TECLA_DERECHA)) {
                                        pep.caminar(entorno, entorno.TECLA_DERECHA);
                                }
                        }
                        // Salto de pep
                        if (entorno.sePresiono(entorno.TECLA_ARRIBA) && pep.puedoSaltar(isla)) {
                                pep.iniciarSalto(); // Iniciar el salto
                                Herramientas.play("Sonidos/salto.wav");
                        }
                        pep.saltar();

                        // Verifico colisiones con las islas para frenar o crear la caída
                        boolean estaSobreUnaIsla = false;
                        for (Islas i : isla) {
                                if (i != null && pep.colisionIsla(i)) {
                                        estaSobreUnaIsla = true;
                                        break;// si esta sigo el flujo del juego
                                }
                        }
                        if (!estaSobreUnaIsla) {
```

```
                    pep.caer(); // si no colisiona con las islas puede caer


                }
                pep.DibujarPep(entorno);


                if (entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)
&& disparos == null && estaSobreUnaIsla == true) {
                    Herramientas.play("Sonidos/disparo.wav");
                    // Disparar si no hay disparo activo


                    if (pep.tieneDireccion()) {
                        char direccionPep = pep.getDireccionPep();
                        disparos = new Disparo(pep.getX(),
pep.getY() + 20, direccionPep);
                        disparos.setDireccion(direccionPep);//
Obtener dirección de Pep
                    } else {
                        char direccionPorDefecto =
entorno.TECLA_DERECHA; // sino no tiene una direccion.Por defecto

                            // sale por la derecha.
                        disparos = new Disparo(pep.getX(),
pep.getY() + 20, direccionPorDefecto);
                        disparos.setDireccion(direccionPorDefecto);
                    }
                }


                if (pep.getY() > 300) {
                    // Variable para contar los gnomos rescatados
```

```
                    // Recorremos el arreglo de gnomos
                    for (int i = 0; i < gnomosEnJuego.length; i++) {

                        if (gnomosEnJuego[i] != null) { // Verificamos que el gnomo aún exista (no sea null)
                            // Verificamos la colisión con Pep
                            if (pep.colisionaConGnomos(gnomosEnJuego[i])) {
                                gnomosRescatados++;
                                if (gnomosRescatados == 10 && tiempo < 60) {
                                    ganoJuego = true;
                                }
                                // Si colisiona, el gnomo se rescata y se elimina (lo hacemos null)
                                gnomosEnJuego[i] = null;

                                Herramientas.play("Sonidos/gnomosalvado.wav");
                                // Incrementamos el contador de gnomos rescatados

                            } else {
                                // Si no hay colisión, seguimos dibujando y moviendo al gnomo
                                gnomosEnJuego[i].dibujarGnomos(entorno);

                                gnomosEnJuego[i].movimientoGnomos(isla);
                            }
                        }
                    }
```

```
                    }
                    // Verificar colisión con todas las tortugas
                    for (int i = 0; i < tortugasEnJuego.length; i++) {
                        if (pep != null) {
                            if (pep.getY() > 580 || tortugasEnJuego[i] !=
null
                                    &&
pep.colisionaConTortuga(tortugasEnJuego[i])) {
                                pep = null;
                                perdioJuego = true;
                                // muere el pep manejar que termine el
juego
                            }
                        }
                    }
                }
                // Actualizo el disparo
                if (disparos != null) {
                    disparos.dibujarDisparo(entorno); // Dibujar el disparo
                    disparos.disparar(disparos.getDireccion(), entorno); // Mover
el disparo en la ultima direccion de pep.

                    // Eliminar el disparo si sale de la pantalla o colisiona con
tortuga
                    if (disparos.fueraDePantalla()) {
                        disparos = null; // El disparo desaparece
                    } else if (disparos.colisionaConTortuga(tortugasEnJuego)) {
                        disparos = null;
                        tortugasEliminadas++;
```

```
                    Herramientas.play("Sonidos/tortugamuerte.wav");

                }
            }

            // Dibujar y mover las tortugas
            int tortugasEnPantalla = 0;

            // Contamos cuántas tortugas no son null (es decir, cuántas están activas en
            // pantalla)
            for (int i = 0; i < this.tortugasEnJuego.length; i++) {
                if (tortugasEnJuego[i] != null) {
                    tortugasEnPantalla++;
                }
            }

            // Si hay menos de 7 tortugas, creamos nuevas en las posiciones que están null
            if (tortugasEnPantalla < 7) {
                for (int i = 0; i < this.tortugasEnJuego.length; i++) {
                    if (tortugasEnJuego[i] == null) {
                        tortugasEnJuego[i] = new TortugaAsesina(32, 35, 2, tortugasEnJuego, i); // Creamos nueva tortuga
                        tortugasEnPantalla++; // Incrementamos el conteo de tortugas en pantalla
                        if (tortugasEnPantalla == 7) { // Cuando alcanzamos 7 tortugas, dejamos de crear más
                            break;
                        }
```

```
                    }
                }
            }

            // Dibujar y mover solo las tortugas que están en pantalla (no null)
            for (int i = 0; i < this.tortugasEnJuego.length; i++) {
                if (tortugasEnJuego[i] != null) {
                    tortugasEnJuego[i].DibujarTotuga(entorno);          //
Dibujamos la tortuga
                    tortugasEnJuego[i].movimiento(isla);  // Movemos  la
tortuga
                }
            }

            // dibujar y mover los Gnomos
            tiempoParaSiguienteGnomo++;
            if (tiempoParaSiguienteGnomo % 80 == 0) {
                // gnomos activos en el juego
                int gnomosActivos = 0;
                for (int i = 0; i < gnomosEnJuego.length; i++) {
                    if (gnomosEnJuego[i] != null) {
                        gnomosActivos++;
                    }
                }

                // para que siempre tenga 6 gnomos
                if (gnomosActivos < 6) {
                    for (int i = 0; i < gnomosEnJuego.length; i++) {
                        if (gnomosEnJuego[i] == null) {
```

```
                                        // Crear un nuevo gnomo con un
desfase en su posición x
                                        double desfaseX = i * 3; // Usa el
índice para desfase de posición
                                        gnomosEnJuego[i] = new Gnomos(400
+ desfaseX, 70, 20, 25);
                                        break; // crea un gnomo por ciclo
                                    }
                                }
                            }
                        }

                        for (int i = 0; i < this.gnomosEnJuego.length; i++) {
                            if (gnomosEnJuego[i] != null) {
                                gnomosEnJuego[i].dibujarGnomos(entorno);
                                gnomosEnJuego[i].movimientoGnomos(isla);

                                // Verificar colisiones
                                for (Islas is : isla) {
                                    if (gnomosEnJuego[i] != null &&
gnomosEnJuego[i].colisionaConIsla(is)) {
                                        // Si colisiona con una isla, actualiza su
movimiento
gnomosEnJuego[i].movimientoGnomos(isla);
                                    } else if (gnomosEnJuego[i] != null) {
                                        // Si no colisiona con ninguna isla, cae
                                        gnomosEnJuego[i].caer(is);
                                    }
                                }
```

```
                                        // Verificar si colisiona con tortugas o sale de pantalla
                                        if
(gnomosEnJuego[i].colisionConTortuga(tortugasEnJuego)                                   ||
gnomosEnJuego[i].salioDePantalla()) {
                                                gnomosPerdidos++;
                                                if (gnomosPerdidos == 25) {
                                                        perdioJuego = true;
                                                }
                                                // Elimina el gnomo y crear uno nuevo
                                                gnomosEnJuego[i] = null;

Herramientas.play("Sonidos/gnomomuerte.wav");
                                                // busco null para crear un nuevo gnomo
                                                for (int j = 0; j < gnomosEnJuego.length; j++)
{
                                                        if (gnomosEnJuego[j] == null) {
                                                                double desfaseX = j * 3; // Usa
el índice para desfase de posición
                                                                gnomosEnJuego[j]    =    new
Gnomos(400 + desfaseX, 70, 20, 25);
                                                                break;  //  crea  un  gnomo  por
ciclo
                                                        }
                                                }
                                        }
                                }
                        }
                }

        }
```

```
	@SuppressWarnings("unused")
	public static void main(String[] args) {
		Juego juego = new Juego();
	}

}
```