



Trabajo Práctico de Programación 1

Integrantes :

Nombre	Email	N° Legajo
Nicolas Fernando Vargas	nicofvargass@gmail.com	38940533
Martin Edgardo Quiroga	martinedgardoquiroya2@gmail.com	40948782
Juan Pablo Galeano	JuampiGaleano37@gmail.com	43022862

Introducción:

El siguiente trabajo consiste en crear un juego “Al rescate de los gnomos” con el lenguaje Java, se trata de un Héroe que tiene que salvar a los gnomos, antes de que los enemigos los maten. Los gnomos tendrán un spawn en la primer isla superior en la casa, y las tortugas caerán desde el cielo de forma aleatoria y caminarán dando vueltas en la isla que cae. El jugador deberá salvar a esos gnomos en las dos filas de islas inferiores y eliminará a las tortugas usando su habilidad especial llamada Bola de Fuego.

Descripción:

Tenemos las siguientes clases creadas del juego:

Jugador:

Tipo	Nombre	Descripción
Clase	Jugador	Crea el objeto Jugador
Atributos	<code>private double x;</code> <code>private double y;</code> <code>private double ancho;</code> <code>private double alto;</code>	Atributos que representan en un plano 2d la ubicación (X e Y) y su ancho y alto, estos atributos son usados en los métodos de colisión.
	<code>private double velocidad;</code> <code>private double velocidadCaida;</code> <code>private double gravedad;</code>	Atributos que son usados para el comportamiento de movimiento en el plano 2d.
	<code>private boolean enElAire;</code> <code>private boolean miraDerecha;</code> <code>private boolean miralq;</code>	Atributos que permite tener un mayor control en el método de <code>aplicarGravedad()</code> , <code>miraDerecha</code> y <code>miralq</code> son usados para establecer la direccional crear el objeto <code>bolaFuego</code> y, además, para que el método <code>dibujar()</code> muestre correctamente entre dos imágenes.
	<code>private String rutaCaminaDerecha;</code> <code>private String rutaCaminalq;</code> <code>private String rutaAtaqueDerecha;</code> <code>private String rutaAtaquelq;</code>	Atributos de tipo String que representan las rutas de imágenes.
	<code>private Image caminaDerecha;</code> <code>private Image caminalq;</code>	Atributos de tipo Image que representan las

	<pre>private Image ataqueDerecha; private Image ataquelzq;</pre>	imágenes que serán utilizadas para mostrar en pantalla.
Métodos	<pre>public double getX(); public double getY() public double getAncho() public double getAlto() public boolean getMiraDerecha() public boolean getMiralzq()</pre>	Métodos que cumplen la función de getters para acceder a los atributos privados de clase.
	<pre>public void dibujar(Entorno entorno)</pre>	Método encargado de dibujar en pantalla
	<pre>public void moverDerecha(Entorno entorno, Isla[] islas) public void moverlzquierda(Isla[] islas) public void saltar(Isla[] islas) public void aplicarGravedad(Isla[] islas)</pre>	Métodos encargados de establecer y actualizar el movimiento/posición en la pantalla.
	<pre>public boolean hayColisionlzq() public boolean hayColisionDer(Entorno entorno) public boolean hayColisionVentanaAbajo(Entorno entorno) public boolean colisionaDerecha(Isla[] islas) public boolean colisionalzquierda(Isla[] islas) public boolean colisionaAbajo(Isla[] islas) public boolean colisionaArriba(Isla[] islas) public boolean colisionaDerechaTortu(Tortuga[] tortugas) public boolean colisionalzquierdaTortu(Tortuga[] tortugas) public boolean colisionaArribaTortu(Tortuga[] tortugas) public boolean colisionaAbajoTortu(Tortuga[] tortugas)</pre>	Métodos que devuelven un boolean si colisionan con ventana, objetos de tipo Isla o objetos de tipo Tortuga. Además actualiza la posición de jugador para no atravesar las mismas.

Esta clase presentó distintos cambios en su estructura y métodos a lo largo de su desarrollo, el principal problema presentado fue el concepto de gravedad al no estar bien desarrollada la velocidad máxima de caída y las condiciones para detener la misma, atravesaba los objetos Isla, además de que el método inicialmente estaba incompleto y faltaban las verificaciones correspondientes (ColisionaAbajo(Isla[] islas) y ColisionaArriba(Isla[] islas)), a su vez, buscando solución a esto hemos encontrado un error en los métodos de colisión notando que solo funcionaban con un margen de 10 (`if(bordeInferiorPersonaje>=bordeSuperiorTortuga && bordeInferiorPersonaje<=bordeSuperiorTortuga +10)`), donde inicialmente estaba seteado con un margen que dependía del atributo velocidad, así que no solo hemos resuelto la gravedad sino que en el proceso logramos solucionar todos los métodos de colisión.

BolaFuego:

Tipo	Nombre	Descripcion
Atributos	private double x; private double y; private double ancho; private double alto;	Atributos que representan en un plano 2d la ubicación (X e Y) y su ancho y alto, estos atributos son usados en los métodos de colisión.
	private double velocidad; private int direccion;	Atributos que establecen la velocidad de movimiento y dirección es usado en el constructor para asignar si va a izquierda o derecha.
	private String rutaCaminaDerecha; private String rutaCaminalzq;	Atributos de tipo String que contienen la ruta de las imágenes.
	private Image caminaDerecha; private Image caminalzq;	Atributo de tipo Image que contienen las imágenes.
Métodos	public void dibujar(Entorno entorno)	Método encargado de dibujar en pantalla al objeto bolaFuego
	public void mover()	Método encargado de mover la bola de fuego.
	public boolean colisionaDerecha(Isla[] islas) public boolean colisionaIzquierda(Isla[] islas) public boolean colisionaAbajo(Isla[] islas) public boolean colisionaArriba(Isla[] islas)	Métodos encargados de devolver true si colisiona con alguna isla o false si no colisiona con ninguna.
	public boolean hayColisionIzq() public boolean hayColisionDer(Entorno entorno)	Métodos encargados de devolver true si colisiona con algún borde de la ventana o false caso contrario
	public double getX() public double getY() public double getAncho() public double getAlto()	Métodos usados para acceder a atributos privados fuera de clase

En esta clase se presentó el problema de que el movimiento al crearse el objeto bolaFuego variaba en el aire si el jugador se movía también (ya que

dependía de un getter de Jugador), la solución fue establecer la dirección en el constructor así no puede alterarse en el transcurso del juego.

Isla:

Variables de instancia de Class isla.

Tipo	Nombre	Descripcion
Atributo	Double x; Double y; Int ancho; Int alto;	Atributos que representan en un plano 2d la ubicación (X e Y) y su ancho y alto, estos atributos son usados en los métodos de colisión.
	Boolean tortugaEnIsla;	Atributo que verifica si tortuga está en la isla,
	Boolean estaEnIsla;	Atributo para confirmar que está en isla
Método	public void establecerTortuga(boolean estado)	verifica si tortuga esta en isla, dependiendo el boolean
	public boolean hayTortuga()	si la tortuga está presente en la isla.
	public void dibujar(Entorno entorno)	Método encargado de dibujar en pantalla al objeto isla.
	public boolean contienePunto(double px, double py)	Se usa para determinar un punto en el medio del rectángulo.
	public double getX() public double getY() public double getAncho() public double getAlto()	Métodos usados para acceder a atributos privados fuera de clase

Uno de los problemas encontrados fueron con isla, nosotros habíamos creado islas, con una variable de posiciones en "Y" y que vaya iterando, cambiando con otro for y dependiendo la posición de "Y" le íbamos restando una isla.

El profesor nos explicó otra manera para que quede mejor. Fue lo que implementamos como isla, que es la que quedó.

Duende:

Tipo	Nombre	Descripción
Atributos	Private Double x;	Atributos que representan en un plano 2d la ubicación (X e Y)
	Private Double y;	Atributos que representan en un plano 2d la ubicación (X e Y)
	Private Double Ancho;	y su ancho y alto, estos atributos son usados en los métodos de colisión.
	Private Double Alto;	y su ancho y alto, estos atributos son usados en los métodos de colisión.
	Private Double velocidad;	Velocidad de movimiento horizontalmente.
	Private Double gravedad;	fuerza de gravedad
	Private Double velocidadCaída;	Velocidad de caída que se incrementa con la gravedad
	Private int direccion;	dirección en la que irá el duende
	Private boolean enElAire;	booleano que define si el duende se encuentra en el aire o no
	Private boolean estaEnLaIsla;	booleano si define si el duende se encuentra en la isla o no
	Private Isla islaActual;	isla donde el duende está actualmente
	Private int maximoDuendes	cantidad máxima de duendes
	Private Map<Isla, Integer> direccionesPorIsla;	mapa para seleccionar direcciones por isla
	Private string rutaDerecha	ruta de imagen para el duende que mira a la derecha
	Private string rutaIzq	ruta de imagen para el duende que mira izquierda
	Private Image imagenDerecha	imagen del duende viendo a la derecha
	Private Image imagenIzq	imagen del duende viendo a la izquierda
	getX() , getY()	método para obtener la posición x e y del duende

Tipo	Nombre	Descripción
Métodos	Private Double x;	Atributos que representan en un plano 2d la ubicación (X e Y)
	public boolean duendeEnELAire(Isla[] islas)	devuelve EnELAire en true si no hay colisión con una isla debajo
	public static void crearDuendesConDelay (List<Duende> duendes, int maximoDuendes, Casa casa)	crea duendes con un delay de 3 segundos para que no aparezcan al mismo tiempo hasta alcanzar un número máximo de duendes. Usa un hilo independiente thread para evitar bloquear la función del juego
	public static int duendesVivos (List<Duende>duendes)	cuenta la cantidad de duendes vivos(!null)
	public boolean colisionaAbajo(Isla[] islas)	Verifica si el duende colisiona con alguna isla debajo. Si colisiona, ajusta su posición y actualiza islaActual a la isla correspondiente.
	public void aplicarGravedad(Isla[] islas)	aplica gravedad al duende, si colisiona con una isla establece enELAire en false
	public void patronDeMovimiento(Isla[] islas)	establece la dirección en la que irá el duende por isla de forma aleatoria

en esta clase se presentaron problemas con la implementación del método aplicarGravedad() y patronDeMovimiento(). Con aplicarGravedad() fue que funcionaba para el duende pero no mantenía la dirección a la que iba mientras caía generando un efecto de hundirse abajo y su solución fue implementar la x de duende junto con su dirección predeterminada en patronDeMovimiento(). El problema con el método patrón Movimiento() fue más complejo debido a que hubo complicaciones en conseguir el efecto de que sea una sola dirección por isla su solución fue almacenar una dirección por cada isla en la que se encuentra el duende.

Hud:

Tipo	Nombre	Descripcion
Atributos	private String rutaFondo;	ruta de imagen fondo
	private Image fondo;	imagen del fondo del juego
	private int cronometro; private int duendesSalvados; private int duendesMuertos; private int enemigosEliminados private int inicio Juego;	contador de tiempo, duendes salvados , duendes muertos , enemigos eliminados y tiempo que transcurrio desde que inicio el juego
Métodos	public void iniciarCronometro(Entorno entorno) public void actualizarCronometro(Entorno entorno)	Marca el momento inicial del juego al almacenar el tiempo actual usando entorno.tiempo()
	public void dibujarFondo(Entorno entorno)	dibuja la imagen del fondo del juego
	public void setDuendesSalvado() public void setDuendesMuerto() public void setEnemigoEliminado()	aumenta el contador en uno de su respectivo atributo asignado
	public int getCronometro() public int getDuendesSalvados() public int getDuendesMuertos() public int get enemigosEliminados()	devuelve el valor de su respectivo atributo asignado
	public string cronometroStr() public string duendesSalvadosStr() public string duendesMuertosStr() public string enemigosEliminadosStr()	devuelve su string asignado y su respectivo contador
	public void dibujarCronometro(Entorno entorno) public void dibujarDuendesSalvados(Entorno entorno) public void dibujarDuendesMuertos(Entorno entorno) public void dibujarEnemigosEliminados(Entorno entorno)	dibuja el string de su contador asignado

Tortuga:

Tipo	Nombre	Descripción
------	--------	-------------

Atributos	Double x; Double Ancho; Double y; Double Alto;	Atributos que representan en un plano 2d la ubicación (X e Y) y su ancho y alto, estos atributos son usados en los métodos de colisión.
Atributos	Boolean estaEnIsla;	Este atributo indica si tortuga está en isla.
Atributos	Double velocidadMovimiento;	Con este atributo manejamos la caída de tortuga.
Atributos	Boolean moviendoDerecha;	vemos si se está corriendo para la derecha o para la izquierda, con true o false.
Atributos	Boolean enElAire	verifica si la tortuga está en el aire, devolviendo un true, de lo contrario devuelve false
Atributos	double gravedad	Atributos que son usados para el comportamiento de movimiento en el plano 2d.
Atributos	String rutaCaminaDerecha; String rutaCaminaIzq;	Atributos de tipo String que representan las rutas de imágenes.
Atributos	Image caminaDerecha; Image CaminaIzq;	Atributo de tipo Image que contienen las imágenes.
Métodos	public void aplicarGravedad(Isla[] islas)	este metodo lo que hace es darle la gravedad a la tortuga
Métodos	public void actualizarPosicion(Isla[] islas)	Aca verificamos si tortuga no esta en isla, ajusta y para que caiga, verifica que tortuga colisiona con algun arreglo de isla. Si tortuga esta en isla, permite que se mueva
Métodos	private boolean colisionConIsla(Isla isla)	verifica si hay una intersección entre tortuga e isla, y el método devuelve, true o false.
Métodos	public void dibujar(Entorno entorno)	Método encargado de dibujar en pantalla al objeto tortuga.
Métodos	private void moverEnIsla()	Permite que se mueva de un lado a otro, dentro de los límites de isla de izquierda a derecha.

Métodos	public void dibujarCaminaDerecha(Entorno entorno) public void dibujarCaminaizq(Entorno entorno)	Dibuja tortuga dependiendo de qué lado , con una imagen.
Métodos	public boolean colisionaDerechaBolaFuego(bolaFuego bolaFuego) public boolean colisionalzquierdaBolaFuego(bolaFuego bolaFuego) public boolean colisionaAbajoBolaFuego(bolaFuego bolaFuego)	Detecta si hay una colisión entre tortuga y bolaFuego, del lado derecho o izquierdo, devuelve true, en caso contrario false
Métodos	public double getX() public double getY() public double getAncho() public double getAlto()	Métodos usados para acceder a atributos privados fuera de clase

Otro de los problemas fue que Tortuga, fue creada con otra gravedad que en el juego , ya que no pudimos usar la del todo el juego , por un error que no pudimos descubrir, el problema fue que cuando aplicamos la misma gravedad la tortuga no se podía mover en la isla, quedaba estática , se intentó de todas las maneras y no pudimos resolverlo.

Lo cual al final pudimos hacer que se aplique gravedad de esta manera:

```
public void aplicarGravedad(Isla[] islas) {
    if(enElAire==true) {
        velocidadCaida+=gravedad;
        if (velocidadCaida>10) {
            velocidadCaida=10;
        }
        this.y+=velocidadCaida;
    }
    if(colisionaAbajo(islas)) {
        enElAire=false;
        velocidadCaida=0;
    }
}
```

Menú:

Tipo	Nombre	Descripción
Atributos	<code>private Entorno entorno;</code>	representamos el entorno del juego
	<code>private int botonIniciarX, botonIniciarY, botonSalirX, botonSalirY</code>	coordenadas para centrar los botones iniciar y Salir.
	<code>private int anchoBoton ;</code> <code>private int altoBoton ;</code>	Con este atributo definimos el ancho del botón y alto
	<code>private String rutaFondo ;</code> <code>private String rutaGanaste;</code> <code>private String rutaPerdiste;</code>	Con este String almacenamos la ruta de la imagen
	<code>private Image fondo;</code> <code>private Image ganaste;</code> <code>private Image perdiste;</code>	la utilizamos para cargar la imagen, que se utiliza en su respectiva ruta
	<code>private int ancho, alto;</code>	con este atributo representa el área de la ventana
Método	<code>public void dibujar()</code>	Dibuja la imagen de fondo, el título del juego, el botón “iniciar partida” y botón “salir de la partida”
	<code>public void dibujarVictoria()</code>	Dibuja la pantalla de victoria con un mensaje que indica al jugador que ha ganado.
	<code>public void dibujarDerrota()</code>	Dibuja la pantalla de derrota con un mensaje que indica al jugador que ha perdido.
	<code>public boolean botonIniciarPresionado()</code>	detecta si el botón “iniciar partida” fue presionado
	<code>public boolean botonSalirPresionado()</code>	detecta si el botón “salir de la partida” fue presionado

Juego:

Tipo	Nombre	Descripción
Atributos	private Jugador jugador private Duende duende List<Duende> duendes private Tortuga tortuga Tortuga[] tortugas bolaFuego bolaFuego Hud ui Casa casa	Atributos que representan los Objetos del juego.
	Isla[] islas double[] posicionesX boolean[] posicionesXUsadas Isla[] islasDelSpawn	Atributos principales en el escenario del juego.
	int maximoDuendes int condicionVictoria int condicionDerrota boolean gano boolean perdio	Atributos que son utilizados como condicion de victoria/derrota.
	Entorno entorno Menu menu EstadoJuego estadoJuego	Atributos principales para el funcionamiento del juego.
	int ultimo Random random	Atributos que son utilizados para control de eventos y seleccion aleatoria de posicion para spawn.
	public double getPosicionAleatoria() public double[] posicionesDisponibles() public void verificarIslaDisponible()	Metodos encargados de elegir un spawn aleatorio de las islas que no contengan una tortuga.
Metodos	public static Isla[] crearIslas()	Metodo encargado de crear la totalidad de islas.
	public void tick()	Metodo encargado de actualizar constantemente la logica del Juego.
	private void reiniciarJuego()	Metodo encargado de reiniciar todos los atributos

		y objetos necesarios a su estado inicial.
--	--	---

En esta clase (la sección tick) fue terminada sin dificultades ya que las clases al tener sus métodos correspondientes fue cuestión de agregar el comportamiento en el tick con una estructura básica de for e if. Fuera del tick sí hemos encontrado una dificultad ya que hemos utilizado métodos que consideramos no eran de clase (métodos relacionados al comportamiento de Tortuga e Isla) sino de la clase Juego, principalmente lo que nos presento dificultad fue lograr que no spawnen tortugas en una isla donde ya hay una tortuga. La solución que hemos encontrado fue crear 3 arrays que se corresponden entre sí en orden (posicionesX[], posicionesXUsadas[]) y el último un array de Islas (islasDelSpawn[]). El comportamiento de los métodos que utilizan estos arrays es el siguiente:

- **verificarIslaDisponible()** : Este método recorre el array islasDelSpawn[] y accede a un atributo de isla llamado tortugaEnIsla mediante el método hayTortuga() cuyo return es un booleano. Si el booleano es true, utiliza el índice para establecer como true el valor en el array posicionesXUsadas[], y viceversa. Este array es utilizado en el siguiente método.

- **posicionesDisponibles(boolean[] posicionesXusadas, double[] posicionesX)** : Este metodo tiene un return de tipo double[], inicialmente tiene un for que itera sobre posicionesXUsadas y suma 1 a un contador por cada false encontrado, este contador se utiliza para asignar la cantidad de posiciones que va a tener el nuevo array. Finalmente se utiliza un último for que itera sobre posicionesXUsadas y por cada false accede a posicionesX utilizando el índice y estos valores se cargan en el nuevo array que retorna este método.

- **getPosicionAleatoria()** : Este método llama a los métodos anteriores en su estructura para verificar que posición está disponible y utiliza el nuevo array retornado para obtener una posición aleatoria que luego es pasada por parámetro al constructor de Tortuga.

Casa:

Tipo	Nombre	Descripción
Atributos	<code>private double x</code> <code>private double y</code> <code>private double ancho</code> <code>private double alto</code>	Atributos que representan en un plano 2d la ubicación (X e Y) y su ancho y alto, estos atributos son usados en los métodos de colisión.
Atributos	<code>private String rutaImagen</code>	Con este String almacenamos la ruta de la imagen
Atributos	<code>private Image imagenCasa</code>	la utilizamos para cargar la imagen, que se utiliza en el fondo
Metodos	<code>public double getY()</code> <code>public double getX()</code>	Getters que son usados para el spawn de duende.

Conclusión:

Finalmente la elaboración del Juego “Al rescate de los Gnomos” nos ayudó a poner en práctica los conocimientos adquiridos durante la cursada poniendo a prueba nuestra lógica para enfrentar problemas en un escenario de desarrollo real y el trabajo en equipo.

Implementación:

Jugador:

```
package Juego;

import entorno.Entorno;
import entorno.Herramientas;

import java.awt.*;

public class Jugador { 9 usages  ± nicofvargas +2 *
    //atributos de clase
    private double x; 26 usages
    private double y; 22 usages
    private double ancho; 23 usages
    private double alto; 20 usages
    private double velocidad; 5 usages
    private double gravedad; 2 usages
    private double velocidadCaida; 7 usages
    private boolean enElAire; 5 usages
    private boolean miraDerecha; 3 usages
    private boolean miraIzq; 4 usages
    private String rutaCaminaDerecha="Images/Mago/magocaminaderecha.png"; 1 usage
    private String rutaCaminaIzq="Images/Mago/magocaminaizq.png"; 1 usage
    private Image caminaDerecha; 2 usages
    private Image caminaIzq; 2 usages
```

```

//Constructor
public Jugador(Entorno entorno) { 1 usage  ± nicofvargas +2
    this.x = 65;
    this.y = 456;
    this.ancho = 20;
    this.alto = 40;
    this.velocidad = 5;
    this.gravedad = 0.3;
    this.velocidadCaida = 0;
    this.enElAire=false;
    this.caminaDerecha= Herramientas.cargarImagen(rutaCaminaDerecha).getScaledInstance((int)this.ancho,(int)this.alto,Image.SCALE_SMOOTH);
    this.caminaIzq= Herramientas.cargarImagen(rutaCaminaIzq).getScaledInstance((int)this.ancho,(int)this.alto,Image.SCALE_SMOOTH);
}

//getters
public double getX() { 7 usages  ± nicofvargas
    return this.x;
}
public double getY() { 8 usages  ± nicofvargas
    return this.y;
}
public boolean getMiraDerecha() { 2 usages  .
    return miraDerecha;
}
public boolean getMiraIzq() { no usages  ± nic
    return miraIzq;
}
public double getAncho() { 6 usages  new *
    return this.ancho;
}
public double getAlto() { 6 usages  new *
    return this.alto;
}

//metodos para mostrar en pantalla
public void dibujar(Entorno entorno) { 1 usage  ± martindev3 +1
    if(this.miraIzq) {
        entorno.dibujarImagen(caminaIzq,this.x,this.y, angulo: 0);
    }
    else {
        entorno.dibujarImagen(caminaDerecha,this.x,this.y, angulo: 0);
    }
}

```



```

//metodos para mover
public void moverDerecha(Entorno entorno, Isla[] islas) { 1 usage  ± nicofvargas
    this.x += velocidad;
    this.miraIzq=false;
    this.miraDerecha=true;
    if (hayColisionDer(entorno)) { //esto es para que no se pase del borde de ventana
        if(colisionaDerecha(islas)) { //esto comprueba si hay colision con isla vuelve para atras
            this.x -= velocidad;
        }
    }
}

public void moverIzquierda(Isla[] islas) { 1 usage  ± nicofvargas
    this.x -= velocidad;
    this.miraDerecha=false;
    this.miraIzq=true;
    if(hayColisionIzq()) {
        if(colisionaIzquierda(islas)) {
            this.x+=velocidad;
        }
    }
}

public void saltar(Isla[] islas) { // 1 us
    if (!enElAire) {
        velocidadCaida = -8;
        enElAire = true;
    }
}

//gravedad
public void aplicarGravedad(Isla[] islas) { 1 usage  ± nicofvargas
    if(enElAire=true) {
        velocidadCaida+=gravedad;
        if (velocidadCaida>10) {
            velocidadCaida=10;
        }
        this.y+=velocidadCaida;
    }
    if(colisionaAbajo(islas) || colisionaArriba(islas)) {
        enElAire=false;
        velocidadCaida=0;
    }
}

```

```
//colision con bordes de ventana
public boolean hayColisionIzq() { 1 usage  ⚙ nicofvargas *
    if(this.x - this.ancho / 2 <= 0) {
        this.x=this.ancho/2;
        return true;
    }
    return false;
}

public boolean hayColisionDer(Entorno entorno) { 1 usage  ⚙
    if (this.x + this.ancho / 2 >= entorno.ancho()) {
        this.x=entorno.ancho()-this.ancho/2;
        return true;
    }
    return false;
}

public boolean hayColisionVentanaAbajo(Entorno entorno) { 1
    return this.y + this.ancho/2 >= entorno.alto();
}
```

//colision con objetos

```
public boolean colisionaDerecha(Isla[] islas) { 2 usages  ± nicofvargas
    for (Isla isla : islas) {
        if (isla == null) {
            continue;
        }

        double bordeDerechoPersonaje = this.x + (this.ancho / 2);
        double bordeIzquierdoIsla = isla.getX() - (isla.getAncho() / 2);

        if (bordeDerechoPersonaje >= bordeIzquierdoIsla && bordeDerechoPersonaje <= bordeIzquierdoIsla + 10) {
            if (this.y + (this.alto / 2) > isla.getY() - (isla.getAlto() / 2) && this.y - (this.alto / 2) < isla.getY() + (isla.getAlto() / 2)) {
                this.x = bordeIzquierdoIsla - (this.ancho / 2);
                return true;
            }
        }
    }
    return false;
}
```

```
public boolean colisionaIzquierda(Isla[] islas) { 2 usages  ± nicofvargas
    for (Isla isla : islas) {
        if (isla == null) {
            continue;
        }

        double bordeIzquierdoPersonaje = this.x - (this.ancho / 2);
        double bordeDerechoIsla = isla.getX() + (isla.getAncho() / 2);

        if (bordeIzquierdoPersonaje <= bordeDerechoIsla && bordeIzquierdoPersonaje >= bordeDerechoIsla - 10) {
            if (this.y + (this.alto / 2) > isla.getY() - (isla.getAlto() / 2) && this.y - (this.alto / 2) < isla.getY() + (isla.getAlto() / 2)) {
                this.x = bordeDerechoIsla + (this.ancho / 2);
                return true;
            }
        }
    }
    return false;
}
```

```
public boolean colisionaArriba(Isla[] islas) { 2 usages  ± nicofvargas
    for(Isla isla : islas) {
        if(isla==null) {
            continue;
        }
        double bordeSuperiorPersonaje = this.y - (this.alto / 2);
        double bordeInferiorIsla = isla.getY() + (isla.getAlto() / 2);

        if(bordeSuperiorPersonaje <= bordeInferiorIsla && bordeSuperiorPersonaje>= bordeInferiorIsla-10) {
            if(this.x+(this.ancho/2) > isla.getX()-(isla.getAncho()/2) && this.x-(this.ancho/2) < isla.getX()+(isla.getAncho()/2)) {
                this.y=bordeInferiorIsla+(this.alto/2);
                return true;
            }
        }
    }
    return false;
}
```

```

public boolean colisionaAbajo(Isla[] islas) { 1 usage  ± nicofvargas
    for(Isla isla : islas) {
        if(isla==null) {
            continue;
        }
        double bordeInferiorPersonaje = this.y + (this.alto / 2);
        double bordeSuperiorIsla = isla.getY() - (isla.getAlto() / 2);

        if(bordeInferiorPersonaje>=bordeSuperiorIsla && bordeInferiorPersonaje<=bordeSuperiorIsla +10) {
            if(this.x+(this.ancha/2) > isla.getX()-(isla.getAncho()/2) && this.x-(this.ancha/2) < isla.getX()+(isla.getAncho()/2)) {
                this.y=bordeSuperiorIsla-(this.alto/2);
                return true;
            }
        }
    }
    return false;
}

```

```

//metodos de colision con tortuga
public boolean colisionaDerechaTortu(Tortuga[] tortugas) { 1 usage  ± nicofvargas
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }

        double bordeDerechoPersonaje = this.x + (this.ancha / 2);
        double bordeIzquierdoTortu = tortuga.getX() - (tortuga.getAncho() / 2);

        if (bordeDerechoPersonaje >= bordeIzquierdoTortu && bordeDerechoPersonaje <= bordeIzquierdoTortu + 10) {
            if (this.y + (this.alto / 2) > tortuga.getY() - (tortuga.getAlto() / 2) && this.y - (this.alto / 2) < tortuga.getY() + (tortuga.getAlto() / 2)) {
                this.x = bordeIzquierdoTortu - (this.ancha / 2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaIzquierdaTortu(Tortuga[] tortugas) { 1 usage  ± nicofvargas
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }

        double bordeIzquierdoPersonaje = this.x - (this.ancha / 2);
        double bordeDerechoTortu = tortuga.getX() + (tortuga.getAncho() / 2);

        if (bordeIzquierdoPersonaje <= bordeDerechoTortu && bordeIzquierdoPersonaje >= bordeDerechoTortu - 10) {
            if (this.y + (this.alto / 2) > tortuga.getY() - (tortuga.getAlto() / 2) && this.y - (this.alto / 2) < tortuga.getY() + (tortuga.getAlto() / 2)) {
                this.x = bordeDerechoTortu + (this.ancha / 2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaArribaTortu(Tortuga[] tortugas) { 1 usage  ± nicofvargas
    for(Tortuga tortuga : tortugas) {
        if(tortuga==null) {
            continue;
        }
        double bordeSuperiorPersonaje = this.y - (this.alto / 2);
        double bordeInferiorTortu = tortuga.getY() + (tortuga.getAlto() / 2);

        if(bordeSuperiorPersonaje <= bordeInferiorTortu && bordeSuperiorPersonaje>= bordeInferiorTortu-10) {
            if(this.x+(this.ancho/2) > tortuga.getX()-(tortuga.getAncho()/2) && this.x-(this.ancho/2) < tortuga.getX()+(tortuga.getAncho()/2)) {
                this.y=bordeInferiorTortu+(this.alto/2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaAbajoTortu(Tortuga[] tortugas) { 1 usage  ± nicofvargas
    for(Tortuga tortuga : tortugas) {
        if(tortuga==null) {
            continue;
        }
        double bordeInferiorPersonaje = this.y + (this.alto / 2);
        double bordeSuperiorTortuga = tortuga.getY() - (tortuga.getAlto() / 2);

        if(bordeInferiorPersonaje>=bordeSuperiorTortuga && bordeInferiorPersonaje<=bordeSuperiorTortuga +10) {
            if(this.x+(this.ancho/2) > tortuga.getX()-(tortuga.getAncho()/2) && this.x-(this.ancho/2) < tortuga.getX()+(tortuga.getAncho()/2)) {
                this.y=bordeSuperiorTortuga-(this.alto/2);
                return true;
            }
        }
    }
    return false;
}

```

bolaFuego:

```
package Juego;
```

```
import entorno.Entorno;
```

```
import entorno.Herramientas;
```

```
import java.awt.*;
```

```
public class bolaFuego { 7 usages  ± nicofvargas +1
```

```
    //atributos
```

```
    private double x; 11 usages
```

```
    private double y; 8 usages
```

```
    private double ancho; 10 usages
```

```
    private double alto; 8 usages
```

```
    private double velocidad; 3 usages
```

```
    private int direccion; 4 usages
```

```
    private String rutaCaminaDerecha="Images/BolaFuego/bolafuegoderecha.png";
```

```
    private String rutaCaminaIzq="Images/BolaFuego/bolafuegoizq.png"; 1 usage
```

```
    private Image caminaDerecha; 2 usages
```

```
    private Image caminaIzq; 2 usages
```

```
//constructor
```

```
public bolaFuego(Jugador jugador) { 1 usage  ± nicofvargas +1
```

```
    this.x=jugador.getX();
```

```
    this.y=jugador.getY();
```

```
    this.ancho=30;
```

```
    this.alto=30;
```

```
    this.velocidad=7;
```

```
    this.caminaDerecha= Herramientas.cargarImagen(rutaCaminaDerecha).getScaledInstance((int)this.ancho,(int)this.alto,Image.SCALE_SMOOTH);
```

```
    this.caminaIzq= Herramientas.cargarImagen(rutaCaminaIzq).getScaledInstance((int)this.ancho,(int)this.alto,Image.SCALE_SMOOTH);
```

```
    //este if lo pongo en el constructor para evitar que cambie la direccion de la bola de fuego al lanzarse
```

```
    if(jugador.getMiraDerecha()) {
```

```
        this.direccion=1;
```

```
    }
```

```
    else {
```

```
        this.direccion=-1;
```

```
    }
```

```
}
```

```
public void dibujar(Entorno entorno) { 1 usage ± Darmus88 +1
    if (direccion==1) {
        entorno.dibujarImagen(caminaDerecha,this.x,this.y, angulo: 0);
    }
    else{
        entorno.dibujarImagen(caminaIzq, this.x, this.y, angulo: 0);
    }
}
```

```
public void mover() { 1 usage ± nicofvargas
    this.x+=velocidad*this.direccion;
}
```

```
//colisiones islas
public boolean colisionaDerecha(Isla[] islas) { 1 usage ± nicofvargas
    for (Isla isla : islas) {
        if (isla == null) {
            continue;
        }

        double bordeDerechoPersonaje = this.x + (this.ancho / 2);
        double bordeIzquierdoIsla = isla.getX() - (isla.getAncho() / 2);

        if (bordeDerechoPersonaje >= bordeIzquierdoIsla && bordeDerechoPersonaje <= bordeIzquierdoIsla + velocidad) {
            if (this.y + (this.alto / 2) > isla.getY() - (isla.getAlto() / 2) && this.y - (this.alto / 2) < isla.getY() + (isla.getAlto() / 2)) {
                this.x = bordeIzquierdoIsla - (this.ancho / 2);
                return true;
            }
        }
    }
    return false;
}
```

```

public boolean colisionaIzquierda(Isla[] islas) { 1 usage  1 nicofvargas
    for (Isla isla : islas) {
        if (isla == null) {
            continue;
        }

        double bordeIzquierdoPersonaje = this.x - (this.anchos / 2);
        double bordeDerechoIsla = isla.getX() + (isla.getAncho() / 2);

        if (bordeIzquierdoPersonaje <= bordeDerechoIsla && bordeIzquierdoPersonaje >= bordeDerechoIsla - 10) {
            if (this.y + (this.alto / 2) > isla.getY() - (isla.getAlto() / 2) && this.y - (this.alto / 2) < isla.getY() + (isla.getAlto() / 2)) {
                this.x = bordeDerechoIsla + (this.anchos / 2);
                return true;
            }
        }
    }
    return false;
}

//colision con ventana
public boolean hayColisionIzq() { 1 usage  1 nicofvargas
    return this.x - this.anchos / 2 <= 0; //le resto el ancho dividido dos porque sino se pasa de la ventana ya que X es el medio del rectangulo
}
public boolean hayColisionDer(Entorno entorno) { 1 usage  1 nicofvargas
    return this.x + this.anchos / 2 >= entorno.anchos(); //aca lo mismo pero a la inversa le falta medio rectangulo para llegar a colisionar
}

```

//getters

```

public double getX() { 8 usages  1 nicofvargas
    return this.x;
}
public double getY() { 8 usages  1 nicofvargas
    return this.y;
}
public double getAncho() { 8 usages  1 nicofvargas
    return this.anchos;
}
public double getAlto() { 8 usages  1 nicofvargas
    return this.alto;
}

```


Isla:

```
public Isla(int x, int y, int ancho, int alto) {  
    super();  
    this.x = x;  
    this.y = y;  
    this.ancho = ancho;  
    this.alto = alto;  
    this.tortugaEnIsla = false;  
    this.imagenIsla = Herramientas.cargarImagen(rutaIsla).getScaledInstance(this.ancho, this.alto, Image.SCALE_SMOOTH);  
}
```

```
public boolean hayTortuga() { return tortugaEnIsla; }  
2 usages  👤 martindev3  
public void establecerTortuga(boolean estado) { this.tortugaEnIsla = estado; }  
  
// Método que dibuja la isla en el entorno  
1 usage  👤 martindev3  
public void dibujar(Entorno entorno) { entorno.dibujarImagen(imagenIsla, this.x, this.y, angulo: 0); }
```

```
public double getX() { return this.x; }  
17 usages  👤 nicofvargas  
public double getY() { return this.y; }  
16 usages  👤 nicofvargas  
public double getAncho() { return this.ancho; }  
16 usages  👤 nicofvargas  
public double getAlto() { return this.alto; }
```

```
public boolean contienePunto(double px, double py) {  
    double margen = 10;  
    return (px >= x - ancho / 2 - margen && px <= x + ancho / 2 + margen &&  
            py >= y - alto / 2 - margen && py <= y + alto / 2 + margen);  
}
```

Duende:

```
1 usage  👤 nicofvargas *1  
public static void crearDuendesConDelay(List<Duede> duendes, int maximoDuendes, Casa casa) {  
    new Thread() -> {  
        while (duendesVivos(duendes) < maximoDuendes) {  
            synchronized (duendes) {  
                for (int i = 0; i < duendes.size(); i++) {  
                    if (duendes.get(i) == null) {  
                        duendes.set(i, new Duende(casa));  
                    }  
                }  
                // Si hay espacio para más duendes, agrega uno nuevo  
                if (duendes.size() < maximoDuendes) {  
                    duendes.add(new Duende(casa));  
                }  
            }  
            try {  
                Thread.sleep( millis: 3000);  
            } catch (InterruptedException e) {  
                Thread.currentThread().interrupt();  
            }  
        }  
    }).start();  
}
```

```

public void dibujar(Entorno entorno) {
    if(direccion()) {
        entorno.dibujarImagen(imagenDerecha,this.x,this.y, angulo: 0);
    }
    else {
        entorno.dibujarImagen(imagenIzq,this.x,this.y, angulo: 0);
    }
}
1 usage  👤 martindev3
public boolean direccion() {
    if(direccion==1) {
        return true;
    }
    return false;
}
}

```

```

2 usages  👤 Darmus88
public double getX() {return this.x;}
2 usages  👤 Darmus88
public double getY() { return this.y; }

1 usage  👤 Darmus88
public boolean duendeEnElAire(Isla[] islas) {
    if (!colisionaAbajo(islas)) {
        enElAire = true;
    }
    return false;
}
}

```

```

1 usage  👤 nicofvargas
public static int duendesVivos(List<Duende> duendes) {
    int contador=0;
    for (Duende duende : duendes) {
        if (duende!=null) {
            contador++;
        }
    }
    return contador;
}
}

```

```

3 usages  Darmus88 +1
public boolean colisionaAbajo(Isla[] islas) {
    for (Isla isla : islas) {
        if (isla == null) {
            continue;
        }
        double bordeInferiorDuende = this.y + (this.alto / 2);
        double bordeSuperiorIsla = isla.getY() - (isla.getAlto() / 2);

        if (bordeInferiorDuende >= bordeSuperiorIsla && bordeInferiorDuende <= bordeSuperiorIsla + (velocidad + 8)) {
            if (this.x + (this.ancho / 2) > isla.getX() - (isla.getAncho() / 2) && this.x - (this.ancho / 2) < isla.getX() + (isla.getAncho() / 2)) {
                this.y = bordeSuperiorIsla - (this.alto / 2);
                islaActual = isla; // Actualiza la isla actual
                return true;
            }
        }
    }
    return false;
}

```

```

no usages  Darmus88
public void aplicarGravedad(Isla[] islas) {
    if (enElAire) {
        velocidadCaida += gravedad;
        if (velocidadCaida > 10) {
            velocidadCaida = 10;
        }
        this.y += velocidadCaida;
        this.x += velocidad * direccion;
    }
    if (colisionaAbajo(islas)) {
        enElAire = false;
        velocidadCaida = 0;
    }
}

```

```

no usages  Darmus88
public void patronDeMovimiento(Isla[] islas) {
    if (colisionaAbajo(islas)) {
        if (!direccionesPorIsla.containsKey(islaActual)) {
            direccion = (Math.random() < 0.5) ? -1 : 1;
            direccionesPorIsla.put(islaActual, direccion);
        } else {
            direccion = direccionesPorIsla.get(islaActual);
        }
        this.x += velocidad * direccion;
    }
}

```

```

public boolean colisionaDerechaTortu(Tortuga[] tortugas) {
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }

        double bordeDerechoPersonaje = this.x + (this.ancha / 2);
        double bordeIzquierdoTortu = tortuga.getX() - (tortuga.getAncho() / 2);

        if (bordeDerechoPersonaje >= bordeIzquierdoTortu && bordeDerechoPersonaje <= bordeIzquierdoTortu + 10) {
            if (this.y + (this.alto / 2) > tortuga.getY() - (tortuga.getAlto() / 2) && this.y - (this.alto / 2) < tortuga.getY() + (tortuga.getAlto() / 2)) {
                this.x = bordeIzquierdoTortu - (this.ancha / 2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaIzquierdaTortu(Tortuga[] tortugas) {
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }

        double bordeIzquierdoPersonaje = this.x - (this.ancha / 2);
        double bordeDerechoTortu = tortuga.getX() + (tortuga.getAncho() / 2);

        if (bordeIzquierdoPersonaje <= bordeDerechoTortu && bordeIzquierdoPersonaje >= bordeDerechoTortu - 10) {
            if (this.y + (this.alto / 2) > tortuga.getY() - (tortuga.getAlto() / 2) && this.y - (this.alto / 2) < tortuga.getY() + (tortuga.getAlto() / 2)) {
                this.x = bordeDerechoTortu + (this.ancha / 2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaArribaTortu(Tortuga[] tortugas) {
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }

        double bordeSuperiorPersonaje = this.y - (this.alto / 2);
        double bordeInferiorTortu = tortuga.getY() + (tortuga.getAlto() / 2);

        if (bordeSuperiorPersonaje <= bordeInferiorTortu && bordeSuperiorPersonaje >= bordeInferiorTortu - 10) {
            if (this.x + (this.ancha / 2) > tortuga.getX() - (tortuga.getAncho() / 2) && this.x - (this.ancha / 2) < tortuga.getX() + (tortuga.getAncho() / 2)) {
                this.y = bordeInferiorTortu + (this.alto / 2);
                return true;
            }
        }
    }
    return false;
}

```

```

public boolean colisionaAbajoTortu(Tortuga[] tortugas) {
    for (Tortuga tortuga : tortugas) {
        if (tortuga == null) {
            continue;
        }


        double bordeInferiorPersonaje = this.y + (this.alto / 2);
        double bordeSuperiorTortuga = tortuga.getY() - (tortuga.getAlto() / 2);

        if (bordeInferiorPersonaje >= bordeSuperiorTortuga && bordeInferiorPersonaje <= bordeSuperiorTortuga + 10) {
            if (this.x + (this.ancha / 2) > tortuga.getX() - (tortuga.getAncho() / 2) && this.x - (this.ancha / 2) < tortuga.getX() + (tortuga.getAncho() / 2)) {
                this.y = bordeSuperiorTortuga - (this.alto / 2);
                return true;
            }
        }
    }
    return false;
}

```

1 usage  nicofvargas


```
public boolean colisionaIzquierdaJugador(Jugador jugador) {  
    double bordeIzquierdoDuende = this.x - (this.ancha / 2);  
    double bordeDerechoJugador = jugador.getX() + (jugador.getAncho() / 2);  
  
    if (bordeIzquierdoDuende <= bordeDerechoJugador && bordeIzquierdoDuende >= bordeDerechoJugador - 10) {  
        if (this.y + (this.alto / 2) > jugador.getY() - (jugador.getAlto() / 2) && this.y - (this.alto / 2) < jugador.getY() + (jugador.getAlto() / 2)) {  
            return true;  
        }  
    }  
    return false;  
}
```

1 usage  nicofvargas

```
public boolean colisionaArribaJugador(Jugador jugador) {  
    double bordeSuperiorDuende = this.y - (this.alto / 2);  
    double bordeInferiorJugador = jugador.getY() + (jugador.getAlto() / 2);  
  
    if (bordeSuperiorDuende <= bordeInferiorJugador && bordeSuperiorDuende >= bordeInferiorJugador - 10) {  
        if (this.x + (this.ancha / 2) > jugador.getX() - (jugador.getAncho() / 2) && this.x - (this.ancha / 2) < jugador.getX() + (jugador.getAncho() / 2)) {  
            return true;  
        }  
    }  
    return false;  
}
```

1 usage  nicofvargas

```
public boolean colisionaAbajoJugador(Jugador jugador) {  
    double bordeInferiorDuende = this.y + (this.alto / 2);  
    double bordeSuperiorJugador = jugador.getY() - (jugador.getAlto() / 2);  
  
    if (bordeInferiorDuende >= bordeSuperiorJugador && bordeInferiorDuende <= bordeSuperiorJugador + 10) {  
        if (this.x + (this.ancha / 2) > jugador.getX() - (jugador.getAncho() / 2) && this.x - (this.ancha / 2) < jugador.getX() + (jugador.getAncho() / 2)) {  
            return true;  
        }  
    }  
    return false;  
}
```

1 usage  martindou3

Hud:

```
package Juego;

import entorno.Entorno;
import entorno.Herramientas;
import java.awt.*;
public class Hud { 2 usages  ⚙ nicofvargas +2 *
    //atributos
    private String rutaFondo="Images/Fondo/fondo.jpg"; 1 u
    private Image fondo; 2 usages
    private int cronometro; 4 usages
    private int duendesSalvados; 3 usages
    private int duendesMuertos; 3 usages
    private int enemigosEliminados; 3 usages
    private int inicioJuego; 3 usages

    //constructor
    public Hud() { ⚙ nicofvargas +1
        this.cronometro=0;
        this.inicioJuego=0;
        this.duendesSalvados=0;
        this.duendesMuertos=0;
        this.enemigosEliminados=0;
        this.fondo = Herramientas.cargarImagen(rutaFondo).getScaledInstance( width: 800, height: 600, hints: 500);
    }
```

```

//getters
public int getCronometro() { 1 nicofvarg
    return this.cronometro;
}
public int getDuendesSalvados() { 1 ni
    return duendesSalvados;
}
public int getDuendesMuertos() { 1 nic
    return duendesMuertos;
}
public int getEnemigosEliminados() {
    return enemigosEliminados;
}

//aca los metodos que actualizan cada valor
public void iniciarCronometro(Entorno entorno) { 3 usages 1 nicofvarg
    this.inicioJuego = entorno.tiempo();
}
public void actualizarCronometro(Entorno entorno) { 1 usage 1 nicofv
    this.cronometro = (entorno.tiempo() - this.inicioJuego) / 1000;
}
public void setCronometro(Entorno entorno) { no usages 1 nicofvargas
    this.cronometro=entorno.tiempo()/1000;
}
public void dibujarFondo(Entorno entorno) { 1 usage 1 martindev3
    entorno.dibujarImagen(fondo, x: 400, y: 300, angulo: 0);
}
public void setDuendesSalvado() { 1 usage 1 nicofvargas
    this.duendesSalvados++;
}
public void setDuendesMuerto() { 2 usages 1 nicofvargas
    this.duendesMuertos++;
}
public void setEnemigosEliminado() { 1 usage 1 nicofvargas
    this.enemigosEliminados++;
}

```

//metodos para devolver los valores String ya que escribirTexto solo recibe por parametro un string, y dos double con los valores de X e Y

```
public String cronometroStr() { 1 usage  ⚡ nicofvargas
    return "Tiempo: " + getCronometro();
}
public String duendesSalvadosStr() { 1 usage  ⚡ nicofvargas
    return "Duendes salvados: " + getDuendesSalvados();
}
public String duendesMuertosStr() { 1 usage  ⚡ nicofvargas
    return "Duendes muertos: " + getDuendesMuertos();
}
public String enemigosEliminadosStr() { 1 usage  ⚡ nicofvargas
    return "Enemigos eliminados: " + getEnemigosEliminados();
}
```

//metodos para dibujar

```
public void dibujarCronometro(Entorno entorno) { 1 usage  ⚡ martindev3 +1
    entorno.cambiarFont( font: "arial", tamano: 18,Color.red,Font.BOLD);
    entorno.escribirTexto(cronometroStr(), x: 680, y: 25);
}
public void dibujarDuendesSalvados(Entorno entorno) { 1 usage  ⚡ nicofvargas +1
    entorno.cambiarFont( font: "arial", tamano: 13,Color.darkGray,Font.BOLD);
    entorno.escribirTexto(duendesSalvadosStr(), x: 25, y: 25);
}
public void dibujarDuendesMuertos(Entorno entorno) { 1 usage  ⚡ martindev3 +1
    entorno.escribirTexto(duendesMuertosStr(), x: 230, y: 25);
}
public void dibujarEnemigosEliminados(Entorno entorno) { 1 usage  ⚡ martindev3 +1
    entorno.escribirTexto(enemigosEliminadosStr(), x: 420, y: 25);
}
```


Tortuga:

```
private boolean colisionConIsla(Isla isla) {  
    return (this.x + this.ancho / 2 >= isla.getX() - isla.getAncho() / 2 &&  
            this.x - this.ancho / 2 <= isla.getX() + isla.getAncho() / 2 &&  
            this.y + this.alto / 2 >= isla.getY() - isla.getAlto() / 2 &&  
            this.y - this.alto / 2 <= isla.getY() + isla.getAlto() / 2);  
}
```

```
public void actualizarPosicion(Isla[] islas) {  
    for (Isla isla : islas) {  
        if(colisionConIsla(isla)) {  
            moverEnIsla(isla);  
        }  
    }  
}
```

```
public void aplicarGravedad(Isla[] islas) {  
    if(enElAire==true) {  
        velocidadCaida+=gravedad;  
        if (velocidadCaida>10) {  
            velocidadCaida=10;  
        }  
        this.y+=velocidadCaida;  
    }  
    if(colisionaAbajo(islas)) {  
        enElAire=false;  
        velocidadCaida=0;  
    }  
}
```

```

private void moverEnIsla() {
    if (moviendoDerecha) {
        x += velocidadMovimiento;
        if (x + ancho / 2 > islaActual.getX() + islaActual.getAncho() / 2) {
            moviendoDerecha = false;
        }
    } else {
        x -= velocidadMovimiento;
        if (x - ancho / 2 < islaActual.getX() - islaActual.getAncho() / 2) {
            moviendoDerecha = true;
        }
    }
}
}

```

```

// usage = Darmus88 + 1
public void dibujar(Entorno entorno) {
    if(moviendoDerecha) {
        entorno.dibujarImagen(caminaDerecha,this.x,this.y, angulo: 0);
    }
    else {
        entorno.dibujarImagen(caminaIzq,this.x,this.y, angulo: 0);
    }
}

```

```

no usages = Darmus88
public void dibujarCaminaDerecha(Entorno entorno) { entorno.dibujarImagen(caminaDerecha,this.x,this.y, angulo: 0); }
no usages = Darmus88
public void dibujarCaminaIzq(Entorno entorno) { entorno.dibujarImagen(caminaIzq,this.x,this.y, angulo: 0); }

```

```

public boolean colisionaDerechaBolaFuego(bolaFuego bolaFuego) {
    double bordeDerechoTortuga = this.x + (this.ancho / 2);
    double bordeIzquierdoBolaFuego = bolaFuego.getX() - (bolaFuego.getAncho() / 2);

    if (bordeDerechoTortuga >= bordeIzquierdoBolaFuego && bordeDerechoTortuga <= bordeIzquierdoBolaFuego + 10) {
        if (this.y + (this.alto / 2) > bolaFuego.getY() - (bolaFuego.getAlto() / 2) && this.y - (this.alto / 2) < bolaFuego.getY() + (bolaFuego.getAlto() / 2)) {
            this.x = bordeIzquierdoBolaFuego - (this.ancho / 2);
            return true;
        }
    }
    return false;
}

```

```

public boolean colisionaDerechaBolaFuego(bolaFuego bolaFuego) {
    double bordeDerechoTortuga = this.x + (this.ancho / 2);
    double bordeIzquierdoBolaFuego = bolaFuego.getX() - (bolaFuego.getAncho() / 2);

    if (bordeDerechoTortuga >= bordeIzquierdoBolaFuego && bordeDerechoTortuga <= bordeIzquierdoBolaFuego + 10) {
        if (this.y + (this.alto / 2) > bolaFuego.getY() - (bolaFuego.getAlto() / 2) && this.y - (this.alto / 2) < bolaFuego.getY() + (bolaFuego.getAlto() / 2)) {
            this.x = bordeIzquierdoBolaFuego - (this.ancho / 2);
            return true;
        }
    }
    return false;
}

```

15 usages  martindev3

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

16 usages  martindev3

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

14 usages  martindev3

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


16 usages  martindev3

```
public double getAlto() { return this.alto; }
```

```
public boolean colisionaAbajoBolaFuego(bolaFuego bolaFuego) {  
    double bordeInferiorTortu = this.y + (this.alto / 2);  
    double bordeSuperiorBolaFuego = bolaFuego.getY() - (bolaFuego.getAlto() / 2);  
  
    if (bordeInferiorTortu >= bordeSuperiorBolaFuego && bordeInferiorTortu <= bordeSuperiorBolaFuego + 10) {  
        if (this.x + (this.ancho / 2) > bolaFuego.getX() - (bolaFuego.getAncho() / 2) && this.x - (this.ancho / 2) < bolaFuego.getX() + (bolaFuego.getAncho() / 2)) {  
            this.y = bordeSuperiorBolaFuego - (this.alto / 2);  
            return true;  
        }  
    }  
    return false;  
}
```

```
1 usage  nicofvargas  
public boolean colisionaIzquierdaBolaFuego(bolaFuego bolaFuego) {  
    double bordeIzquierdoTortu = this.x - (this.ancho / 2);  
    double bordeDerechoBolaFuego = bolaFuego.getX() + (bolaFuego.getAncho() / 2);  
  
    if (bordeIzquierdoTortu <= bordeDerechoBolaFuego && bordeIzquierdoTortu >= bordeDerechoBolaFuego - 10) {  
        if (this.y + (this.alto / 2) > bolaFuego.getY() - (bolaFuego.getAlto() / 2) && this.y - (this.alto / 2) < bolaFuego.getY() + (bolaFuego.getAlto() / 2)) {  
            this.x = bordeDerechoBolaFuego + (this.ancho / 2);  
            return true;  
        }  
    }  
    return false;  
}
```

Casa:


```
1 usage  Darmus88 +1  
public void dibujar(Entorno entorno) { entorno.dibujarImagen(imagenCasa,this.x,this.y, angulo: 0); }
```

1 usage  nicofvargas

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

1 usage  nicofvargas

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

```
1 usage  Darmus88 +1  
public Casa(Isla[] islas) {  
    this.x=islas[0].getX();  
    this.y=islas[0].getY()-islas[0].getAlto()-this.alto;  
    this.ancho=100;  
    this.alto=100;  
    this.imagenCasa= Herramientas.cargarImagen(rutaImagen).getScaledInstance((int) this.ancho,(int) this.alto, Image.SCALE_SMOOTH);  
}
```


Menú:

```
public void dibujar() {  
  
    entorno.dibujarImagen(fondo, x: ancho / 2, y: alto / 2, angulo: 0);  
  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 50, Color.BLACK);  
    entorno.escribirTexto( texto: "Al rescate de los Gnomos", x: entorno.ancho() / 2 - 250, y: entorno.alto() / 4);  
  
    entorno.dibujarRectangulo(botonIniciarX, botonIniciarY, anchoBoton, altoBoton, angulo: 0, Color.GREEN);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Iniciar Partida", x: botonIniciarX - 60, y: botonIniciarY + 5);  
  
    entorno.dibujarRectangulo(botonSalirX, botonSalirY, anchoBoton, altoBoton, angulo: 0, Color.red);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Salir de la Partida", x: botonSalirX - 75, y: botonSalirY + 5);  
}
```

```
public boolean botonSalirPresionado() {  
    return entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO) &&  
        entorno.mouseX() >= botonSalirX - anchoBoton / 2 &&  
        entorno.mouseX() <= botonSalirX + anchoBoton / 2 &&  
        entorno.mouseY() >= botonSalirY - altoBoton / 2 &&  
        entorno.mouseY() <= botonSalirY + altoBoton / 2;  
}
```

2 usages  Darmus88

```
public boolean botonIniciarPresionado() {  
    return entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO) &&  
        entorno.mouseX() >= botonIniciarX - anchoBoton / 2 &&  
        entorno.mouseX() <= botonIniciarX + anchoBoton / 2 &&  
        entorno.mouseY() >= botonIniciarY - altoBoton / 2 &&  
        entorno.mouseY() <= botonIniciarY + altoBoton / 2;  
}
```

1 usage  nicofvargas +1

```
public void dibujarDerrota() {  
  
    entorno.dibujarImagen(perdiste, x: ancho / 2, y: alto / 2, angulo: 0);  
  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 50, Color.RED);  
    entorno.escribirTexto( texto: "HAS PERDIDO", x: entorno.anch() / 2-170 , y: entorno.alto() / 4);  
  
    entorno.dibujarRectangulo(botonIniciarX, botonIniciarY, anchoBoton, altoBoton, angulo: 0, Color.GREEN);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Iniciar Partida", x: botonIniciarX - 60, y: botonIniciarY + 5);  
  
    entorno.dibujarRectangulo(botonSalirX, botonSalirY, anchoBoton, altoBoton, angulo: 0,Color.red);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Salir de la Partida", x: botonSalirX - 75, y: botonSalirY + 5);  
}
```

```
public void dibujarVictoria() {  
  
    entorno.dibujarImagen(ganaste, x: ancho / 2, y: alto / 2, angulo: 0);  
  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 50, Color.BLUE);  
    entorno.escribirTexto( texto: "HAS GANADO", x: entorno.anch() / 2-170 , y: entorno.alto() / 4);  
  
    entorno.dibujarRectangulo(botonIniciarX, botonIniciarY, anchoBoton, altoBoton, angulo: 0, Color.GREEN);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Iniciar Partida", x: botonIniciarX - 60, y: botonIniciarY + 5);  
  
    entorno.dibujarRectangulo(botonSalirX, botonSalirY, anchoBoton, altoBoton, angulo: 0,Color.red);  
    entorno.cambiarFont( font: "Engravers MT", tamaño: 20, Color.WHITE);  
    entorno.escribirTexto( texto: "Salir de la Partida", x: botonSalirX - 75, y: botonSalirY + 5);  
}
```

Juego:

```

package Juego;
import entorno.Entorno;
import entorno.InterfaceJuego;
import java.util.Random;
import java.util.ArrayList;
import java.util.List;

public class Juego extends InterfaceJuego {  ⚡ nicofvargas +2 *
    // El objeto Entorno que controla el tiempo y otros
    private Entorno entorno; 33 usages
    private Menu menu; 8 usages
    private Jugador jugador; 27 usages
    private Isla[] islas; 26 usages
    private Duende duende; 1 usage
    private List<Duende> duendes; 10 usages
    private Tortuga tortuga; no usages
    private Tortuga[] tortugas; 22 usages
    private bolaFuego bolaFuego; 16 usages
    private Hud ui; 18 usages
    private Casa casa; 6 usages
    private int maximoDuendes=4; 2 usages
    private int ultimo; 4 usages
    private double[] posicionesX={35,85,165,250,520,650,715,765}; 3 usages
    private boolean[] posicionesXUsadas = new boolean[posicionesX.length];
    private Isla[] islasDelSpawn = new Isla[posicionesX.length]; 10 usages
    private Random random = new Random(); 1 usage
    private int condicionVictoria=10; 1 usage
    private int condicionDerrota=10; 1 usage
    private boolean gano; 5 usages
    private boolean perdio; 4 usages
    private enum EstadoJuego { INICIO, EN_JUEGO, FINALIZADO } 9 usages  ⚡ [
    private EstadoJuego estadoJuego = EstadoJuego.INICIO; 7 usages

```

```

Juego() //constructor 1 usage  ̈́ nicofvargas +2 *
{
    this.entorno = new Entorno( juego: this, titulo: "Al rescate de los Gnomos", ancho: 800, alto: 600);
    this.menu = new Menu(entorno);
    this.entorno.iniciar();
    this.jugador = new Jugador(entorno);
    this.ui = new Hud();
    this.bolaFuego=null;
    islas=crearIslas(entorno);
    this.tortugas = new Tortuga[3];
    islasDelSpawn[0]=islas[10];
    islasDelSpawn[1]=islas[7];
    islasDelSpawn[2]=islas[3];
    islasDelSpawn[3]=islas[1];
    islasDelSpawn[4]=islas[2];
    islasDelSpawn[5]=islas[5];
    islasDelSpawn[6]=islas[9];
    islasDelSpawn[7]=islas[14];
    this.casa = new Casa(islas);
    this.duende = new Duende(casa);
    this.duendes = new ArrayList<>();
    Duende.crearDuendesConDelay(duendes, maximoDuendes, casa);
    this.ultimo= ui.getCronometro();
    this.gano=false;
    this.perdio=false;
    estadoJuego = EstadoJuego.INICIO;
}

//metodos encargados de obtener posicion aleatoria de tortuga
//aca obtengo una posicion aleatoria de X para tortugas(el return de este metodo es pasado por parametro a tortuga)
public double getPosicionAleatoria() { 1 usage  ̈́ nicofvargas
    verificarIslaDisponible();
    double[] posicionesDisponibles = posicionesDisponibles(posicionesXUsadas,posicionesX);
    return posicionesDisponibles[random.nextInt(posicionesDisponibles.length)];
}

```

```

//aca asigno a cada posicion del array el objeto isla para que correspondan por orden
public double[] posicionesDisponibles(boolean[] posicionesXUsadas, double[] posicionesX) {
    //aca obtengo la cantidad de indices que va a tener el nuevo array
    int contador=0;
    for (int i=0; i<posicionesXUsadas.length;i++) {
        if(posicionesXUsadas[i]==false) {
            contador++;
        }
    }
    //aca asigno el tamaño del nuevo array y cargo los valores de X que pueden ser usados
    double[] nuevoArray = new double[contador];
    int indiceArray=0;
    for (int i=0; i<posicionesXUsadas.length; i++) {
        if(!posicionesXUsadas[i]) {
            nuevoArray[indiceArray]=posicionesX[i];
            indiceArray++;
        }
    }
    return nuevoArray;
}

//aca compruebo que la posicion no esta siendo usada
private void verificarIslaDisponible() { 1 usage 2 ni
    for (int i=0; i<islasDelSpawn.length;i++) {
        if (islasDelSpawn[i].hayTortuga()) {
            posicionesXUsadas[i]=true;
        }
        else {
            posicionesXUsadas[i]=false;
        }
    }
}
}

```


//metodo que crea las islas

```
public static Isla[] crearIslas(Entorno entorno) { 1 usage 2 m
    int pisos=5;
    Isla[] islas=new Isla[pisos*(pisos+1)/2];
    int y=0;
    int x=0;
    int separacion=75;
    int indice=0;
    for(int i=1 ;i<=pisos; i++) {
        x=x-30;
        y=y+100;
        int expansion=-40*i;
        for(int j=1 ; j<=i; j++) {
            x=(entorno.ancho()-expansion)/(i+1)*j+expansion/2;
            islas[indice]= new Isla(x,y, ancho: 100, alto: 30);
            indice++;
        }
    }
    return islas;
}
```

```
public void tick() { 2 nicofvargas +2
```

```
    if (estadoJuego == EstadoJuego.INICIO) {
        menu.dibujar();
        if (menu.botonIniciarPresionado()) {
            estadoJuego = EstadoJuego.EN_JUEGO;
            gano = false;
            perdio = false;
            ui.iniciarCronometro(entorno);
        } else if (menu.botonSalirPresionado()) {
            System.exit( status: 0);
        }
    }
    else if (estadoJuego == EstadoJuego.EN_JUEGO) {

        if (jugador != null && (jugador.colisionaAbajoTortu(tortugas) || jugador.colisionaArribaTortu(tortugas) ||
            jugador.colisionaDerechaTortu(tortugas) || jugador.colisionaIzquierdaTortu(tortugas) ||
            jugador.hayColisionVentanaAbajo(entorno))) {
            jugador = null;
        }
    }
}
```

```

//a partir de aca esta la interaccion de todos los objetos entre si
if (jugador != null) {
    ui.actualizarCronometro(entorno);
    ui.dibujarFondo(entorno);
    jugador.dibujar(entorno);
    jugador.aplicarGravedad(islas);
    casa.dibujar(entorno);
    ui.dibujarCronometro(entorno);
    ui.dibujarDuendesSalvados(entorno);
    ui.dibujarDuendesMuertos(entorno);
    ui.dibujarEnemigosEliminados(entorno);

    // Dibuja y maneja las islas y tortugas
    for (Isla isla : islas) {
        if (isla != null) {
            isla.dibujar(entorno);
            isla.hayTortuga(tortugas);
        }
    }
}

//con tiempoactual e intervalo haciendo un calculo simple podemos obtener un temporizador usando el valor de intervalo
int tiempoactual = ui.getCronometro();
int intervalo = 2;
for (int i = 0; i < tortugas.length; i++) { //recorre el array de tortugas y crea una nueva en una posicion null
    if (tortugas[i] == null && tiempoactual - ultimo >= intervalo) { //se utiliza el intervalo
        double posRandom = getPosicionAleatoria(); //obtiene posicion aleatoria
        tortugas[i] = new Tortuga(posRandom); //crea una nueva tortuga
        ultimo = tiempoactual;
    }
    if (tortugas[i] != null) { //si no es null se aplica el comportamiento con los demas objetos
        tortugas[i].aplicarGravedad(islas);
        tortugas[i].actualizarPosicion(islas);
        tortugas[i].dibujar(entorno);
        //se comprueba si bola de fuego colisiona con tortuga
        if (bolaFuego != null && (tortugas[i].colisionaAbajoBolaFuego(bolaFuego) ||
            tortugas[i].colisionaIzquierdaBolaFuego(bolaFuego) ||
            tortugas[i].colisionaDerechaBolaFuego(bolaFuego))) {
            tortugas[i] = null; //establece en null si colisiona
            bolaFuego = null; //se establece en null si colisiona
            ui.setEnemigosEliminado(); //aumento el contador
        }
    }
}
}

```

//comportamiento de duende

```
for (int i = 0; i < duendes.size(); i++) {
    Duende duende = duendes.get(i);
    if (duende == null) {
        duendes.set(i, new Duende(casa));
        continue;
    }
    if (duende.getX() < 0 || duende.getX() > entorno.ancho() || duende.getY() < 0 || duende.getY() > entorno.alto()) {
        duendes.set(i, null);
        ui.setDuendesMuerto();
        continue;
    }
    duende.dibujar(entorno);
    duende.aplicarGravedad(islas);
    duende.patronDeMovimiento(islas);
    duende.duendeEnElAire(islas);

    if (duende.colisionaIzquierdaTortu(tortugas) || duende.colisionaDerechaTortu(tortugas) ||
        duende.colisionaAbajoTortu(tortugas) || duende.colisionaArribaTortu(tortugas)) {
        duendes.set(i, null);
        ui.setDuendesMuerto();
    }
    if ((duende.colisionaAbajoJugador(jugador) || duende.colisionaArribaJugador(jugador) ||
        duende.colisionaDerechaJugador(jugador) || duende.colisionaIzquierdaJugador(jugador)) &&
        jugador.getY() > islas[5].getY()) {
        duendes.set(i, null);
        ui.setDuendesSalvado();
    }
}
```

```

// Control de movimiento con teclas
if (entorno.estaPresionada(entorno.TECLA_DERECHA) && !jugador.colisionaDerecha(islas)) {
    jugador.moverDerecha(entorno, islas);
}
if (entorno.estaPresionada(entorno.TECLA_IZQUIERDA) && !jugador.colisionaIzquierda(islas)) {
    jugador.moverIzquierda(entorno, islas);
}
if (entorno.estaPresionada(entorno.TECLA_ARRIBA) && !jugador.colisionaArriba(islas)) {
    jugador.saltar(islas);
}
if (entorno.estaPresionada(key: 'c') && bolaFuego == null) {
    bolaFuego = new bolaFuego(jugador);
}

// Comprobación de colisiones de bola de fuego
if (bolaFuego != null) {
    bolaFuego.mover();
    bolaFuego.dibujar(entorno);
    if (bolaFuego.colisionaDerecha(islas) || bolaFuego.colisionaIzquierda(islas) ||
        bolaFuego.hayColisionDer(entorno) || bolaFuego.hayColisionIzq()) {
        bolaFuego = null;
    }
}

// Verificación de victoria o derrota
if (condicionVictoria == ui.getDuendesSalvados()) {
    gana = true;
    jugador = null;
}
if (condicionDerrota == ui.getDuendesMuertos()) {
    perdio = true;
    jugador=null;
}

// Cambia a FINALIZADO si el jugador gana o pierde
if (jugador == null) {
    estadoJuego = EstadoJuego.FINALIZADO;
}
}

```

```

else if (estadoJuego == EstadoJuego.FINALIZADO) {
    if (gano) {
        menu.dibujarVictoria();
    } else {
        menu.dibujarDerrota();
    }
    if (menu.botonIniciarPresionado()) {
        reiniciarJuego();
    } else if (menu.botonSalirPresionado()) {
        System.exit( status: 0);
    }
}
}
}

```

```

private void reiniciarJuego() { 1 usage  ⚡ Darmus88 +1
    estadoJuego = EstadoJuego.EN_JUEGO;
    gano = false;
    perdio = false;
    jugador = new Jugador(entorno);
    tortugas = new Tortuga[3];
    duendes.clear();
    Duende.crearDuendesConDelay(duendes, maximoDuendes, casa);
    ui.iniciarCronometro(entorno);
    ultimo=0;
    ui.reiniciarContadores();
}

```

```

@SuppressWarnings("unused")  ⚡ martindev3 +1
public static void main(String[] args)
{
    Juego juego = new Juego();
}
}

```