Materia: Programación I

Comisión: 3
Grupo: 1

Docentes:
Sabrina Castro - scastro@campus.ungs.edu.ar

Rodrigo González - rgonzalez@campus.ungs.edu.ar

Ezequiel David Cañete -edcanete@campus.ungs.edu.ar

Integrantes:
Juan Manuel Corigliano Giuliani, 40.910.320, jmcg.ungs@gmail.com

Juan Francisco Scopa, 41.972.218, juanscopa@hotmail.com

Germán Pérez, 42.494.463, gergperez0021@gmail.com

Germán Schroeder, 44.214.139, gerimport37@gmail.com

Introducción

En este informe, abordaremos los conceptos fundamentales que hemos utilizado en la creación del juego, incluyendo la gestión de gráficos, la manipulación de eventos y la implementación de la lógica del juego. Además, discutiremos los desafíos enfrentados durante el proceso de desarrollo y las soluciones implementadas para superarlos.

Descripción

Clase Tortuga

Descripción General

La clase Tortuga maneja la lógica de movimiento y animación de una tortuga dentro de un juego, permitiendo que se mueva de forma dinámica y reaccione
a otros objetos en el entorno, esta clase se extiende de Personaje y representa a una tortuga en el juego, que puede moverse y tiene una animación de caminar.
El objetivo de estas tortugas es que puedan aniquilar a los gnomos y ser disparadas por Pep.

Variables de instancia

private int direccionX: Controla la dirección en la que se mueve la tortuga (izquierda o derecha).

private int isla: Identifica a qué isla pertenece la tortuga.

private image[] caminarALaDerecha / private image[] caminarALaIzquierda: Arrays de imágenes que contienen los frames de animación de la tortuga caminando a la derecha e izquierda.

private int tiempoTranscurrido: Lleva el tiempo desde el último cambio de frame en la animación.

private int imagenAnteriorDerecha / private int imagenAnteriorIzquierda: Almacenan los índices de la imagen actual en la animación.

Constructor

El constructor inicializa la tortuga en una posición (x, y), con un tamaño y velocidad definidos.
Carga las imágenes de la animación desde archivos en la carpeta imágenes.

Métodos Principales

public void aplicarGravedad(): Incrementa la posición vertical de la tortuga, simulando la gravedad.

public void rebotarTortugas(): Controla el movimiento de la tortuga. Si se encuentra en los bordes de un área definida, cambia su dirección.
También limita el movimiento para que no se salga de los bordes de la pantalla.

boolean estaCercaDePep(Pep pep): Determina si la tortuga está cerca de un objeto Pep. Esto se hace comparando las posiciones de ambos objetos.

public boolean intersectaConDisparo(Disparo disparo): Verifica si la tortuga colisiona con un objeto Disparo usando un cálculo de colisión basado en sus dimensiones y posiciones.

dibujar(Entorno entorno, int altoDeResolucion): Dibuja la imagen de la tortuga en la pantalla, cambia entre las imágenes de animación según la dirección y el tiempo,y la dibuja en su posición actual.

Implementación

```
public class Tortuga extends Personaje {
	private int direccionX;
	private int isla;
	// Array que contiene la animacion de caminar a la derecha
	private Image[] caminarALaDerecha = new Image[5];
	// Array que contiene la animacion de caminar a la izquierda
```

```
private Image[] caminarALaIzquierda = new Image[5];
// Tiempo tomado desde que la tortuga cambio de frame en la animacion
private int tiempoTranscurrido = 0;
// Variable para almacenar los indices del frame de animacion anterior
private int imagenAnteriorDerecha = 0;
private int imagenAnteriorIzquierda = 0;
public Tortuga(int x, int y, int ancho, int alto, int velocidad, int isla) {
        super(x, y, ancho, alto, velocidad);
        this.direccionX = (Math.random() < 0.5) ? -1 : 1;
        this.isla = isla;
        this.velocidadDeCaida=4;
        // Almacena cada frame de animacion dentro de los arrays
        for (int i = 0; i < caminarALaDerecha.length; i++) {
                this.caminarALaDerecha[i] = Herramientas.cargarImagen("imagenes/tortugacaminaderecha" + (i + 1) + ".png");
                this.caminarALaIzquierda[i] = Herramientas
                                .cargarImagen("imagenes/tortugacaminaizquierda" + (i + 1) + ".png");
        }
}
public int getVelocidad() {
        return velocidad;
}
public int getAncho() {
        return ancho;
}
public int getAlto() {
        return alto;
}
public int getIsla() {
        return isla;
}
public void setVelocidad(int velocidad) {
        this.velocidad = velocidad;
}
public void setX(int x) {
        this.x = x;
}
public void setY(int y) {
        this.y = y;
}
public void aplicarGravedad() {
        this.y += velocidadDeCaida;
}
public void rebotarTortugas(int centroX, int bordeIzquierdo, int bordeDerecho) {
        // cambiar dirección
        if (x < bordeIzquierdo || x > bordeDerecho) {
                direccionX = -direccionX;
        }
        if (x < centroX) {
                x += this.velocidad * direccionX; // Moverse a la derecha o izquierda
        } else if (x > centroX) {
                x -= velocidad * direccionX; // Moverse a la izquierda o derecha
        }
        // Limitar movimiento para que no se salga de los bordes de la pantalla
        if (x < 0) {
                x = 0;
        } else if (x > 1280) { // Asumiendo que el ancho de resolución es 1280
                x = 1280;
        }
```

```
	}
	boolean estaCercaDePep(Pep pep) {
		int pepIzquierda = pep.getX() - pep.getAncho() / 2;
		int pepDerecha = pep.getX() + pep.getAncho() / 2;
		int pepArriba = pep.getY() - pep.getAlto() / 2;
		int pepAbajo = pep.getY() + pep.getAlto() / 2;
		int tortugaIzquierda = this.x - this.ancho / 2;
		int tortugaDerecha = this.x + this.ancho / 2;
		int tortugaArriba = this.y - this.alto / 2;
		int tortugaAbajo = this.y + this.alto / 2;
		return tortugaDerecha > pepIzquierda && tortugaIzquierda < pepDerecha && tortugaAbajo > pepArriba
				&& tortugaArriba < pepAbajo;
	}
	public boolean intersectaConDisparo(Disparo disparo) {
		int xMin1 = this.x - ancho / 2;
		int xMax1 = this.x + ancho / 2;
		int yMin1 = this.y - alto / 2;
		int yMax1 = this.y + alto / 2;
		int xMin2 = disparo.getX() - disparo.getAncho() / 2;
		int xMax2 = disparo.getX() + disparo.getAncho() / 2;
		int yMin2 = disparo.getY() - disparo.getAlto() / 2;
		int yMax2 = disparo.getY() + disparo.getAlto() / 2;
		return (xMin1 < xMax2 && xMax1 > xMin2 &&
				yMin1 < yMax2 && yMax1 > yMin2);
	}


	public void dibujar(Entorno entorno, int altoDeResolucion) {
		Image imagen;
		if (this.direccionX == 1) {
			imagen = this.caminarALaDerecha[this.imagenAnteriorDerecha];
			if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
				this.imagenAnteriorDerecha++;
				this.tiempoTranscurrido = entorno.tiempo();
			}
			if (this.imagenAnteriorDerecha > this.caminarALaDerecha.length - 1) {
				this.imagenAnteriorDerecha = 0;
			}
		} else {
			imagen = this.caminarALaIzquierda[this.imagenAnteriorIzquierda];
			if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
				this.imagenAnteriorIzquierda++;
				this.tiempoTranscurrido = entorno.tiempo();
			}
			if (this.imagenAnteriorIzquierda > this.caminarALaIzquierda.length - 1) {
				this.imagenAnteriorIzquierda = 0;
			}
		}
		entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
	}
	public int getY() {
		return y;
	}
}
```

Clase Disparo

Descripción General

La clase Disparo genera un rectángulo el cual se llamará desde la clase Juego para que salga desde Pep y pueda borrar a las tortugas.

Variables de Instancia

Private int x: posición en x en ese instante del disparo.

Private int y: posición en y en ese instante del disparo.

Private int ancho:el ancho de disparo de pep.

Private int alto:el alto del disparo de pep.

Private int velocidad: la velocidad que recorre el disparo.

Constructor

El constructor inicializa el disparo en una posición (x, y), con un tamaño y velocidad definidos.
Carga la imágen del disparo desde la carpeta imágenes.

Métodos Principales

public void moverALaDerecha(): Mueve el disparo en el eje X hacia la derecha

public void moverALaIzquierda(): Mueve el disparo en el eje X hacia la derecha

public void dibujar(Entorno entorno): Simplemente dibuja el disparo. Quedó comentado por si se necesitaba volver a ver la hitbox del disparo

public void dibujar(Entorno entorno, int altoDeResolucion, boolean pepMirabaALaDerechaCuandoDisparo): Se ocupa de cargar la imagen correspondiente del disparo

Implementación

```
public class Disparo {
  private int x;
  private int y;
  private int ancho;
  private int alto;
  private int velocidad;
  public Disparo(int x, int y, int ancho, int alto, int velocidad) {
     this.x = x;
     this.y = y;
     this.ancho = ancho;
     this.alto = alto;
     this.velocidad = velocidad;
  }
  public void moverALaDerecha() {
     this.x += this.velocidad;
  }
  public void moverALaIzquierda() {
     this.x -= this.velocidad;
  }
  /*
   * public void dibujar(Entorno entorno) {
   * entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.alto, 0,
   * Color.ORANGE);
   * }
   */
  public void dibujar(Entorno entorno, int altoDeResolucion, boolean pepMirabaALaDerechaCuandoDisparo) {
     Image imagen;
     if (pepMirabaALaDerechaCuandoDisparo) {
        imagen = Herramientas.cargarImagen("imagenes/poderderecha.png");
     } else {
        imagen = Herramientas.cargarImagen("imagenes/poderizquierda.png");
     }
     entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
     entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
  }
  public int getX() {
     return this.x;
  }
  public int getY() {
     return this.y;
  }
  public int getAncho() {
     return this.ancho;
  }
  public int getAlto() {
```

```
    return this.alto;
  }
  public int getVelocidad() {
    return this.velocidad;
  }
}
```

Clase Gnomos

Descripción general

Esta clase se encarga de generar los Gnomos, creando a la misma como una extensión de la clase "Personaje" así heredaba los atributos que ésta tiene.

Variables de instancia

int velocidad: le da una velocidad de "1" al gnomo.

private boolean salvado: es un booleano sobre si el gnomo está salvado por pep o si no lo está.

private int direccionX: asigna la dirección horizontal al gnomo, siendo -1 hacia la izquierda y 1 para la derecha.

private int tiempoTranscurrido: toma el tiempo desde que el gnomo cambia de frame en la animación dada.

private int imagenAnteriorDerecha y private int imagenAnteriorIzquierda: son variables que almacenan el frame de la animación anterior

.

private Image[] caminarALaDerecha = new Image[5]: Array que contiene la animación de caminar a la derecha.

private Image[] caminarALaIzquierda = new Image[5]: Array que contiene la animación de caminar a la izquierda.

this.direccionX = (Math.random() < 0.5) ? -1 : 1; :Asigna una dirección aleatoria al crear el gnomo.

Constructor

El constructor inicializa al gnomo en una posición (x, y) aleatoria, con un tamaño y velocidad definidos.
Carga las imágenes de la animación desde archivos en la carpeta imagenes.

Métodos principales

public boolean estaSalvado(): retorna la variable de instancia "salvado"

public void moverHaciaCentro(int centroX) : Le da un movimiento aleatorio a los gnomos cambiando la dirección con un 1% de probabilidad de cambio. Limita los movimientos para que se no salgan del borde de la pantalla y le da como parámetro el ancho de resolución de la pantalla

public void caer(): aplica la gravedad ya creada en personaje.

boolean estaCercaDePep(Pep pep): Este método es un booleano que marca los bordes del gnomo y los de Pep (derecha, izquierda, arriba, abajo)
para verificar si hay colisión entre los dos personajes.

boolean estaCercaDeTortuga(Tortuga tortuga): De la misma manera que el anterior método indica si hay colisión entre las tortugas y los gnomos.

public void dibujar(Entorno entorno, int altoDeResolucion): Dibuja el gnomo con el método dibujarImagen llamado desde "Entorno" y le da una imagen correspondiente por cada movimiento del gnomo, si se mueve para la derecha
dibuja la imagen correspondiente hacia la derecha y viceversa.

Implementación

```
public class Gnomo extends Personaje {
  int velocidad = 1;
  private boolean salvado = false;
  private int direccionX; // -1 para izquierda, 1 para derecha
```

```
// Array que contiene la animacion de caminar a la derecha
private Image[] caminarALaDerecha = new Image[5];
// Array que contiene la animacion de caminar a la izquierda
private Image[] caminarALaIzquierda = new Image[5];
// Tiempo tomado desde que el gnomo cambio de frame en la animacion
private int tiempoTranscurrido = 0;
// Variable para almacenar los indices del frame de animacion anterior
private int imagenAnteriorDerecha = 0;
private int imagenAnteriorIzquierda = 0;
public Gnomo(int x, int y, int ancho, int alto, int velocidad) {
    super(x, y, ancho, alto, velocidad);
    this.direccionX = (Math.random() < 0.5) ? -1 : 1; // Asigna una dirección aleatoria al crear el gnomo
    // Almacena cada frame de animacion dentro de los arrays
    for (int i = 0; i < caminarALaDerecha.length; i++) {
        this.caminarALaDerecha[i] = Herramientas.cargarImagen("imagenes/gnomocaminaderecha" + (i + 1) + ".png");
        this.caminarALaIzquierda[i] = Herramientas.cargarImagen("imagenes/gnomocaminaizquierda" + (i + 1) + ".png");
    }
}
public boolean estaSalvado() {
    return salvado;
}
public void moverHaciaCentro(int centroX) {
    // Cambia aleatoriamente la dirección lateral
    if (Math.random() < 0.01) { // Cambia de dirección con un 1% de probabilidad
        direccionX = -direccionX; // Cambiar dirección
    }
    // Mover en la dirección aleatoria
    if (x < centroX) {
        x += velocidad * direccionX; // Moverse a la derecha o izquierda
    } else if (x > centroX) {
        x -= velocidad * direccionX; // Moverse a la izquierda o derecha
    }
    // Limitar movimiento para que no se salga de los bordes de la pantalla
    if (x < 0) {
        x = 0;
    } else if (x > 1280) { // Asumiendo que el ancho de resolución es 1280
        x = 1280;
    }
}
public void caer() {
    this.aplicarGravedad();
}
boolean estaCercaDePep(Pep pep) {
    int pepIzquierda = pep.getX() - pep.getAncho() / 2;
    int pepDerecha = pep.getX() + pep.getAncho() / 2;
    int pepArriba = pep.getY() - pep.getAlto() / 2;
    int pepAbajo = pep.getY() + pep.getAlto() / 2;
    int gnomoIzquierda = this.x - this.ancho / 2;
    int gnomoDerecha = this.x + this.ancho / 2;
    int gnomoArriba = this.y - this.alto / 2;
    int gnomoAbajo = this.y + this.alto / 2;
    return gnomoDerecha > pepIzquierda && gnomoIzquierda < pepDerecha &&
           gnomoAbajo > pepArriba && gnomoArriba < pepAbajo;
}
boolean estaCercaDeTortuga(Tortuga tortuga) {
    int tortugaIzquierda = tortuga.getX() - tortuga.getAncho() / 2;
    int tortugaDerecha = tortuga.getX() + tortuga.getAncho() / 2;
    int tortugaArriba = tortuga.getY() - tortuga.getAlto() / 2;
```

```
    int tortugaAbajo = tortuga.getY() + tortuga.getAlto() / 2;
    int gnomoIzquierda = this.x - this.ancho / 2;
    int gnomoDerecha = this.x + this.ancho / 2;
    int gnomoArriba = this.y - this.alto / 2;
    int gnomoAbajo = this.y + this.alto / 2;
    return gnomoDerecha > tortugaIzquierda && gnomoIzquierda < tortugaDerecha &&
            gnomoAbajo > tortugaArriba && gnomoArriba < tortugaAbajo;
  }
  public void dibujar(Entorno entorno, int altoDeResolucion) {
    Image imagen;
    if (this.direccionX == 1) {
      imagen = this.caminarALaDerecha[this.imagenAnteriorDerecha];
      if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
        this.imagenAnteriorDerecha++;
        this.tiempoTranscurrido = entorno.tiempo();
      }
      if (this.imagenAnteriorDerecha > this.caminarALaDerecha.length - 1) {
        this.imagenAnteriorDerecha = 0;
      }
    } else {
      imagen = this.caminarALaIzquierda[this.imagenAnteriorIzquierda];
      if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
        this.imagenAnteriorIzquierda++;
        this.tiempoTranscurrido = entorno.tiempo();
      }
      if (this.imagenAnteriorIzquierda > this.caminarALaIzquierda.length - 1) {
        this.imagenAnteriorIzquierda = 0;
      }
    }
    entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
  }
}
```

Clase Nave:

Descripción general

La clase nave se ve representada por una nave dentro del juego, que sigue la dirección del cursor mediante la orden del click derecho manteniéndolo presionado y cumple la función de ayudar a pep a salvar los gnomos.

Variables de Instancia

Private int x: posiciona a la nave sobre el eje x mediante un valor ingresado.

Private int y: Posiciona a la nave sobre el eje y mediante un valor ingresado.

Private int ancho: el ancho exacto de la nave.

Private int alto: el alto exacto de la nave.

Constructor

El constructor inicializa la nave en una posición (x,y) con un ancho y alto definido.

Métodos principales

moverConCursor():Método para mover la nave mediante el cursor del mouse

colisionConNave():Método para comprobar si hay colisión entre la nave y pep,o entre las nave y los gnomos.

Clase Isla

Descripción general

La clase IslaFlotante es la encargada de generar las plataformas.

Variables de Instancia

anchoDeClase: Ancho de cada isla.

altoDeClase: Alto de cada isla.

Metodos principales

setVariablesDeClase: Le asigna los valores pasados como parámetros a las variables de clase 'anchoDeClase' y 'altoDeClase'. Este método debe llamarse antes de instanciar una isla, es decir, antes de llamar al constructor de esta clase. Se hizo de esta manera, para no tener que repetir el mismo código una y otra vez al instanciar las islas.

Variables de instancia

Private int x: Coordenada x del centro de cada isla.

Private int y: Coordenada y del centro de cada isla.

.

Private int ancho: Ancho de cada isla.

Private int alto: Alto de cada isla.

Constructor

IslaFlotante: Antes de crear un objeto con este constructor, debe llamarse al método 'setVariablesDeClase'.Le asigna los valores pasados como parámetros a las variables de instancia. 'ancho' y 'alto' le asigna los valores de las variables de clase 'anchoDeClase' y 'altoDeClase'.

Métodos principales

tieneTortuga(): Comprueba si alguna tortuga se encuentra entre el inicio y el final de una isla.

Dibujar(): Dibuja una isla en pantalla, en las coordenadas dadas.

Implementación

```
public class IslaFlotante {
  // Variables de clase
  private static int anchoDeClase;
  private static int altoDeClase;
  // Este metodo debe llamarse antes de la instanciacion
  public static void setVariablesDeClase(int ancho, int alto) {
     anchoDeClase = ancho;
     altoDeClase = alto;
  }
  private int x;
  private int y;
  private int ancho;
  private int alto;
  // Antes de la instanciacion debe llamarse al metodo 'setVariablesDeClase'
  public IslaFlotante(int x, int y) {
     this.x = x;
     this.y = y;
     // anchoDeClase & altoDeClase son variables de clase
     this.ancho = anchoDeClase;
     this.alto = altoDeClase;
  }

  boolean tieneTortuga(Tortuga[] tortugas) {
                        for (int i = 0; i < tortugas.length; i++) {
                                    int inicioIsla = getX() - getAncho() / 2;
                                    int finIsla = getX() + getAncho() / 2;
                                    if (tortugas[i] != null && tortugas[i].getX() >= inicioIsla && tortugas[i].getX() <= finIsla) {
                                                return true;
                                    }
                        }
                        return false;
            }
  public void dibujar(Entorno entorno, int altoDeResolucion) {
     Image imagen = Herramientas.cargarImagen("imagenes/isla.png");
```

```
    entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
}
public int getX() {
    return x;
}
public int getY() {
    return y;
}
public int getAncho() {
    return ancho;
}
public int getAlto() {
    return alto;
}
```

Clase Pep

Descripción general:

La clase Pep es la encargada de generar al personaje principal .

Variables de instancia

miraALaDerecha: Se utiliza para comprobar si Pep está mirando a la derecha.
Se inicia con el valor 'true'.

caminarALaDerecha: Es un arreglo que contiene las imágenes de la animación de caminar a la derecha.

caminarALaIzquierda: Es un arreglo que contiene las imágenes de la animacion de caminar a la izquierda.

tiempoTranscurrido: Representa el tiempo tomado desde que la imagen de Pep durante una animacion cambio por otra.Se utiliza principalmente para agregar algo de delay entre cada imagen de la animación.

imagenAnteriorDerecha: Almacena el índice de la última imagen que se dibujó durante una animacion en la que Pep está mirando a la derecha.

imagenAnteriorIzquierda: Almacena el índice de la última imagen que se dibujó durante una animacion en la que Pep está mirando a la izquierda.

Constructor:

Pep: Le asigna los valores pasados como parámetros a las variables de instancia.Además, almacena cada imagen de las animaciones en sus arreglos correspondientes.

Métodos principales

moverIzquierda(): Reduce el valor de la coordenada x,haciendo que Pep se mueva hacia la izquierda.

moverDerecha(): Aumenta el valor de la coordenada x,haciendo que Pep se mueva hacia la derecha.

Saltar(): Si Pep no está en el aire, reduce el valor de la coordenada y,haciendo que Pep salte.

Dibujar(): Dibuja a Pep dependiendo de si está quieto,salta, camina o dispara.En el caso de que camine, se muestra la animación correspondiente.

Implementación

```
public class Pep extends Personaje {
  // Pep comienza mirando a la derecha
  private boolean miraALaDerecha = true;
  // Array que contiene la animacion de caminar a la derecha
  private Image[] caminarALaDerecha = new Image[5];
  // Array que contiene la animacion de caminar a la izquierda
  private Image[] caminarALaIzquierda = new Image[5];
  // Tiempo tomado desde que Pep cambio de frame en la animacion
  private int tiempoTranscurrido = 0;
  // Variable para almacenar los indices del frame de animacion anterior
  private int imagenAnteriorDerecha = 0;
  private int imagenAnteriorIzquierda = 0;
  public Pep(int x, int y, int ancho, int alto, int velocidad) {
     // 'super' utiliza el constructor de la clase padre
     super(x, y, ancho, alto, velocidad);
```

```
    // Almacena cada frame de animacion dentro de los arrays
    for (int i = 0; i < caminarALaDerecha.length; i++) {
        this.caminarALaDerecha[i] = Herramientas.cargarImagen("imagenes/pepcaminaderecha" + (i + 1) + ".png");
        this.caminarALaIzquierda[i] = Herramientas.cargarImagen("imagenes/pepcaminaizquierda" + (i + 1) + ".png");
    }
}
public void moverIzquierda() {
    this.x -= this.velocidad;
    this.miraALaDerecha = false;
}
public void moverDerecha() {
    this.x += this.velocidad;
    this.miraALaDerecha = true;
}
public void saltar() {
    if (!(this.enElAire)) {
        this.velocidadDeCaida = -this.velocidad * 6;
        this.y -= this.velocidad;
        this.enElAire = true;
    }
}
// Dibuja a Pep dependiendo de si esta quieto, salta, camina o dispara
public void dibujar(Entorno entorno, int altoDeResolucion, boolean pepDisparo, boolean pepCamina) {
    Image imagen;
    if (this.miraALaDerecha) {
        if (pepDisparo) {
            imagen = Herramientas.cargarImagen("imagenes/pepdisparoderecha.png");
        } else if (pepCamina && !this.enElAire) {
            imagen = this.caminarALaDerecha[this.imagenAnteriorDerecha];
            if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
                this.imagenAnteriorDerecha++;
                this.tiempoTranscurrido = entorno.tiempo();
            }
            if (this.imagenAnteriorDerecha > this.caminarALaDerecha.length - 1) {
                this.imagenAnteriorDerecha = 0;
            }
        } else if (this.enElAire) {
            imagen = Herramientas.cargarImagen("imagenes/pepsaltaderecha.png");
        } else {
            imagen = Herramientas.cargarImagen("imagenes/pepderecha.png");
        }
    } else {
        if (pepDisparo) {
            imagen = Herramientas.cargarImagen("imagenes/pepdisparoizquierda.png");
        } else if (pepCamina && !this.enElAire) {
            imagen = this.caminarALaIzquierda[this.imagenAnteriorIzquierda];
            if (entorno.tiempo() - this.tiempoTranscurrido > 50) {
                this.imagenAnteriorIzquierda++;
                this.tiempoTranscurrido = entorno.tiempo();
            }
            if (this.imagenAnteriorIzquierda > this.caminarALaIzquierda.length - 1) {
                this.imagenAnteriorIzquierda = 0;
            }
        } else if (this.enElAire) {
            imagen = Herramientas.cargarImagen("imagenes/pepsaltaizquierda.png");
        } else {
            imagen = Herramientas.cargarImagen("imagenes/pepizquierda.png");
        }
```

```
    }
    entorno.dibujarImagen(imagen, this.x, this.y, 0, altoDeResolucion / 200);
  }
  public boolean getMiraAlaDerecha() {
    return this.miraALaDerecha;
  }
  public void setY(int y) {
    this.y = y;
  }
}
```

Clase Personaje

Descripción general

La clase Personaje es la clase padre de las siguientes clases: Pep, Gnomo y Tortuga. Éstas derivan los métodos y variables de instancia que Personaje conlleva

.

Variables de instancia

protected int x; Asigna la posición x.

.

protected int y; Asigna la posición y.

protected int ancho; Asigna el ancho del personaje.

protected int alto; Asigna el alto del personaje.

protected int velocidad; Establece la velocidad que los personajes tienen.

protected int velocidadDeCaida; Velocidad de caida del personaje en eje Y.

protected boolean enElAire; Booleano que indica si el personaje está o no en el aire.

protected final double gravedad = 1; Asigna finalmente la gravedad de todos los personajes.

Constructor

El constructor cumple la función de inicializar a un Personaje en una posición (x,y) con su ancho y alto correspondiente.Además de 2 variables por defecto que son:velocidad de caida y momento en el aire.

Métodos principales

public void aplicarGravedad(): método que aplica la gravedad sumándole la velocidad de caída donde si la velocidad de caída es mayor a 7 se le asigna 7.

public void colisionConIslas(IslaFlotante[] islasFlotantes): Comprueba si los bordes del personaje colisionan con las islas y calcula el solapamiento, de tal manera encuentra el menor entre las estos dos, reinicia la velocidad de caída debido a la colisión y restablece al valor por defecto para poder volver a saltar.

Implementación

```
public class Personaje {
  protected int x;
  protected int y;
  protected int ancho;
  protected int alto;
  protected int velocidad;
  protected int velocidadDeCaida;
  protected boolean enElAire;
  // La gravedad es constante y nunca cambia
  protected final double gravedad = 1;
  public Personaje(int x, int y, int ancho, int alto, int velocidad) {
      this.x = x;
      this.y = y;
      this.ancho = ancho;
      this.alto = alto;
      this.velocidad = velocidad;
      // Variables con valores por defecto
      this.velocidadDeCaida = 0;
      this.enElAire = false;
  }
  public int getX() {
      return x;
  }
  public int getY() {
      return y;
  }
  public int getAncho() {
      return ancho;
  }
  public int getAlto() {
      return alto;
  }
```

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

public void setVelocidadDeCaida(int velocidadDeCaida) {
            this.velocidadDeCaida=velocidadDeCaida;
}
public void aplicarGravedad() {
    this.velocidadDeCaida += gravedad;
    if (this.velocidadDeCaida > 7) {
        this.velocidadDeCaida = 7;
    }
    this.y += velocidadDeCaida;
}
public void colisionConIslas(IslaFlotante[] islasFlotantes) {
    int pepIzquierda = this.getX() - (this.getAncho() / 2);
    int pepDerecha = this.getX() + (this.getAncho() / 2);
    int pepArriba = this.getY() - (this.getAlto() / 2);
    int pepAbajo = this.getY() + (this.getAlto() / 2);
    for (IslaFlotante islaFlotante : islasFlotantes) {
        int islaFlotanteIzquierda = islaFlotante.getX() - (islaFlotante.getAncho() / 2);
        int islaFlotanteDerecha = islaFlotante.getX() + (islaFlotante.getAncho() / 2);
        int islaFlotanteArriba = islaFlotante.getY() - (islaFlotante.getAlto() / 2);
        int islaFlotanteAbajo = islaFlotante.getY() + (islaFlotante.getAlto() / 2);
        // Comprobar si hay colision
        if (pepDerecha > islaFlotanteIzquierda && pepIzquierda < islaFlotanteDerecha &&
                pepAbajo > islaFlotanteArriba && pepArriba < islaFlotanteAbajo) {
            // Calcular cada solapamiento
            int solapamientoIzquierda = pepDerecha - islaFlotanteIzquierda;
            int solapamientoDerecha = islaFlotanteDerecha - pepIzquierda;
            int solapamientoArriba = pepAbajo - islaFlotanteArriba;
            int solapamientoAbajo = islaFlotanteAbajo - pepArriba;
            // Encontrar el menor solapamiento
            int solapamientoMinimo = Math.min(Math.min(solapamientoIzquierda, solapamientoDerecha),
                    Math.min(solapamientoArriba, solapamientoAbajo));
            if (solapamientoMinimo == solapamientoIzquierda) {
                // Colision desde la izquierda
                this.x = islaFlotanteIzquierda - (this.ancho / 2);
                break;
            } else if (solapamientoMinimo == solapamientoDerecha) {
                // Colision desde la derecha
                this.x = islaFlotanteDerecha + (this.ancho / 2);
                break;
            } else if (solapamientoMinimo == solapamientoArriba) {
                // Colision desde arriba
                this.y = islaFlotanteArriba - (this.alto / 2);
                // Reiniciar la velocidad de caida debido a la colision
                this.velocidadDeCaida = 0;
                // Restablecer al valor por defecto para poder volver a saltar
                enElAire = false;
                break;
            } else if (solapamientoMinimo == solapamientoAbajo) {
                // Colision desde abajo
                this.y = islaFlotanteAbajo + (this.alto / 2);
                break;
            }
        }
    }
```

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

Clase EstadoDelJuego

Descripción General

La clase EstadoDeJuego se ocupa de contar y mostrar las estadísticas de la partida,estas serían los gnomos salvados, gnomos perdidos, tortugas eliminadas y el tiempo transcurrido.

Variables de instancia

private int gnomosSalvados: Cuenta los gnomos salvados, inicializa en 0.

private int gnomosPerdidos:Cuenta los gnomos perdidos, inicializa en 0.

private int enemigosEliminados: Cuenta las tortugas eliminadas, inicializa en 0.

Métodos Principales

public String cadenaDeTiempoRestante(int milisegundos, int limiteDeTiempo): Crea la cuenta atrás para el final de la partida.

public void mostrarEnPantalla(Entorno entorno, int anchoDeResolucion, int altoDeResolucion): Muestra en pantalla las estadísticas de tiempo faltante, gnomos salvados, gnomos perdidos y tortugas eliminadas.

Implementación

```
public class EstadoDelJuego {
  private int gnomosSalvados;
  private int gnomosPerdidos;
  private int enemigosEliminados;
```

```
    public EstadoDelJuego() {
        this.gnomosSalvados = 0;
        this.gnomosPerdidos = 0;
        this.enemigosEliminados = 0;
    }
    public int getGnomosSalvados() {
        return gnomosSalvados;
    }
    public int getGnomosPerdidos() {
        return gnomosPerdidos;
    }
    public int getEnemigosEliminados() {
        return enemigosEliminados;
    }
    public void setGnomosSalvados(int gnomosSalvados) {
        this.gnomosSalvados = gnomosSalvados;
    }
    public void setGnomosPerdidos(int gnomosPerdidos) {
        this.gnomosPerdidos = gnomosPerdidos;
    }
    public void setEnemigosEliminados(int enemigosEliminados) {
        this.enemigosEliminados = enemigosEliminados;
    }
    // Devuelve la cadena del tiempo restante para que termine el juego
    public String cadenaDeTiempoRestante(int milisegundos, int limiteDeTiempo) {
        int segundos = (limiteDeTiempo - milisegundos + 999) / 1000;
        int minutos = segundos / 60;
        int restoDeSegundos = segundos - (60 * minutos);
        if (restoDeSegundos < 10) {
            return minutos + ":0" + restoDeSegundos;
        }
        return minutos + ":" + restoDeSegundos;
    }
    // Muestra el estado del juego en pantalla
    public void mostrarEnPantalla(Entorno entorno, int anchoDeResolucion, int altoDeResolucion) {
        // Fuente del juego
        entorno.cambiarFont("Comic Sans MS", anchoDeResolucion / 30, Color.DARK_GRAY);
        int mitadDePantalla = anchoDeResolucion / 2;
        // Muestra los 4 mensajes en pantalla
        entorno.escribirTexto("Tiempo: " + this.cadenaDeTiempoRestante(entorno.tiempo(), 120000), mitadDePantalla / 20,
                altoDeResolucion / 15);
        entorno.escribirTexto("Salvados: " + this.gnomosSalvados, mitadDePantalla / 2, altoDeResolucion / 15);
        entorno.escribirTexto("Perdidos: " + this.gnomosPerdidos, mitadDePantalla + (mitadDePantalla / 10),
                altoDeResolucion / 15);
        entorno.escribirTexto("Eliminados: " + this.enemigosEliminados, mitadDePantalla + (mitadDePantalla / 2),
                altoDeResolucion / 15);
    }
}
```

Clase Juego

Variables de instancia

entorno: Incluye los métodos necesarios para poder llevar a cabo el desarrollo del juego sin que sea necesario escribir todo de cero.

estadoDelJuego: Objeto de la clase EstadoDelJuego.

islas Flotantes: Arreglo de objetos de la clase Isla Flotante.

Pep: Objeto de la clase Pep.

Disparo: Objeto de la clase Disparo.

gnomos: Arreglo de objetos de la clase Gnomo.

tortugas: Arreglo de objetos de la clase Tortuga.

nave: Objeto de la clase Nave.

anchoDeResolucion: Ancho de resolución del juego.Se inicializa con un valor entero de '1280'.

altoDeResolucion: Alto de resolución del juego.Se inicializa con un valor entero de '720'.

xDePepCuandoDisparo: Valor de la coordenada x cuando Pep realizó un disparo.

pepMirabaALaDerechaCuandoDisparo: Se utiliza para comprobar la dirección en la que Pep disparo.

pepCamina: Se utiliza para comprobar si Pep está caminando. Se inicializa con un valor booleano 'false'.

xDePepAnterior: Se utiliza para comprobar si Pep está quieto.

Implementación

```
public class Juego extends InterfaceJuego {
	// El objeto Entorno que controla el tiempo y otros
	private Entorno entorno;
```

```
// Variables y métodos propios de cada grupo
// Variables para cada clase
private EstadoDelJuego estadoDelJuego;
private IslaFlotante[] islasFlotantes;
private Pep pep;
private Disparo disparo;
private Gnomo[] gnomos;
private Tortuga[] tortugas;
private Nave nave;
// Resolucion del juego
private int anchoDeResolucion = 1280;
private int altoDeResolucion = 720;
// Variables para determinar el movimiento, duracion
// y dibujo del disparo
private int xDePepCuandoDisparo;
private boolean pepMirabaALaDerechaCuandoDisparo;
// Variables para determinar que Pep esta caminando
private boolean pepCamina = false;
private int xDePepAnterior;
// Variables para determinar que tipo de casa dibujar
private boolean algunGnomoNoExiste = false;
private int milisegundosHastaAhora;
// Variable para saber si Pep sigue vivo
private boolean pepEstaVivo = true;
Juego() {
        // Instancia el objeto entorno
        this.entorno = new Entorno(this, "Al Rescate de los Gnomos", anchoDeResolucion, this.altoDeResolucion);
        // Inicializar lo que haga falta para el juego
        // (agregar todo debajo de este punto)
        // Valores automaticamente calculados dependiendo de la resolucion
        // Se utilizan para instanciar las islas
        int centroX = this.anchoDeResolucion / 2;
        int centroY = this.altoDeResolucion / 2;
        int separacionHorizontal = centroX / 3;
        int separacionVertical = centroY / 3;
        // Variables para el posicionamiento de las tortugas
        int numeroDeIsla = 0;
        int islasElegidas[] = new int[3];
        // Instanciacion del estado del juego
        this.estadoDelJuego = new EstadoDelJuego();
        // Calcula el ancho y alto de cada isla flotante dependiendo de la resolucion
        IslaFlotante.setVariablesDeClase(anchoDeResolucion / 6, this.altoDeResolucion / 20);
        // Arreglo de islas flotantes (se establece la cantidad)
        this.islasFlotantes = new IslaFlotante[9];
        // Instanciacion de cada isla (los valores se calculan automaticamente)
        this.islasFlotantes[0] = new IslaFlotante(centroX, separacionVertical);
        this.islasFlotantes[1] = new IslaFlotante(centroX - separacionHorizontal, separacionVertical * 2);
        this.islasFlotantes[2] = new IslaFlotante(centroX + separacionHorizontal, separacionVertical * 2);
        this.islasFlotantes[3] = new IslaFlotante(centroX - separacionHorizontal * 2, centroY);
        this.islasFlotantes[4] = new IslaFlotante(centroX, centroY);
        this.islasFlotantes[5] = new IslaFlotante(centroX + separacionHorizontal * 2, centroY);
        this.islasFlotantes[6] = new IslaFlotante(centroX - separacionHorizontal, separacionVertical * 4);
        this.islasFlotantes[7] = new IslaFlotante(centroX + separacionHorizontal, separacionVertical * 4);
        this.islasFlotantes[8] = new IslaFlotante(centroX, separacionVertical * 5);
        // Instanciacion de los Gnomos
        this.gnomos = new Gnomo[3];
        for (int i = 0; i < gnomos.length; i++) {
                gnomos[i] = new Gnomo(this.anchoDeResolucion / 2, this.altoDeResolucion / 10, this.altoDeResolucion / 20,
```

```
                                        this.altoDeResolucion / 20, 2);
                }
                // Instanciacion de las Tortugas
                this.tortugas = new Tortuga[3];
                for (int i = 0; i < tortugas.length; i++) {
                        // Elige una isla entre la 0 y la 5 excluyendo la 0 y la 4
                        while (numeroDeIsla == 0 || numeroDeIsla == 4 || numeroDeIsla == islasElegidas[0]
                                        || numeroDeIsla == islasElegidas[1]) {
                                numeroDeIsla = (int) Math.floor(Math.random() * 6);
                        }
                        islasElegidas[i] = numeroDeIsla;
                        tortugas[i] = new Tortuga(this.islasFlotantes[islasElegidas[i]].getX(), 0, this.altoDeResolucion / 20,
                                        this.altoDeResolucion / 20, 0, 0);
                }
                // Instanciacion de Pep
                this.pep = new Pep(centroX, (separacionVertical * 5) - (this.altoDeResolucion / 40), this.altoDeResolucion / 20,
                                this.altoDeResolucion / 20, 3);
                // Instanciacion de la Nave
                this.nave = new Nave(centroX, (separacionVertical * 6) - (this.altoDeResolucion / 20),
                                this.altoDeResolucion / 6,
                                this.altoDeResolucion / 20);
                // 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
                // dibuja el fondo
                this.entorno.dibujarImagen(Herramientas.cargarImagen("imagenes/fondo.png"), anchoDeResolucion / 2,
                                altoDeResolucion / 2, 0, 2);
                // Comprueba si el juego termino,
                // y si es asi, muestra el mensaje correspondiente
                if (!(this.pepEstaVivo)) { // Si Pep cae al vacio o toca una tortuga
                        limpiarPantalla();
                        this.entorno.cambiarFont("Comic Sans MS", anchoDeResolucion / 10, Color.RED);
                        this.entorno.escribirTexto("Has muerto...", this.anchoDeResolucion / 5, this.altoDeResolucion / 2);
                } else if (this.entorno.tiempo() >= 120000) { // Si se acaba el tiempo
                        limpiarPantalla();
                        this.entorno.cambiarFont("Comic Sans MS", anchoDeResolucion / 10, Color.RED);
                        this.entorno.escribirTexto("Tiempo terminado", this.anchoDeResolucion / 14, this.altoDeResolucion / 2);
                } else if (this.estadoDelJuego.getGnomosPerdidos() >= 20) { // Si muere cierta cantidad de gnomos
                        limpiarPantalla();
                        this.entorno.cambiarFont("Comic Sans MS", anchoDeResolucion / 10, Color.RED);
                        this.entorno.escribirTexto("Gnomos aniquilados", this.anchoDeResolucion / 14, this.altoDeResolucion / 2);
                } else if (this.estadoDelJuego.getGnomosSalvados() >= 10) { // Si se rescata cierta cantidad de gnomos
                        limpiarPantalla();
                        this.entorno.cambiarFont("Comic Sans MS", anchoDeResolucion / 10, Color.DARK_GRAY);
                        this.entorno.escribirTexto("¡ Has ganado !", this.anchoDeResolucion / 6, this.altoDeResolucion / 2);
                }
                // Comprueba si los gnomos existen
                if (this.gnomos != null) {
                        // Comprueba si algun gnomo no existe
                        for (Gnomo gnomo : this.gnomos) {
```

```
                                        if (gnomo == null) {
                                                this.algunGnomoNoExiste = true;
                                                this.milisegundosHastaAhora = this.entorno.tiempo();
                                                break;
                                        }
                                }
                        }
                        if (this.pep != null) {
                                // Muestra en pantalla el estado del juego
                                this.estadoDelJuego.mostrarEnPantalla(this.entorno, anchoDeResolucion, altoDeResolucion);
                                // Si algun gnomo no existe, se dibuja la casa con la puerta abierta.
                                // En caso contrario, se dibuja la casa normal
                                if (this.algunGnomoNoExiste) {
                                        this.entorno.dibujarImagen(Herramientas.cargarImagen("imagenes/casadegnomosabierta.png"),
                                                        anchoDeResolucion / 2, altoDeResolucion / 15, 0, 3);
                                } else {
                                        this.entorno.dibujarImagen(Herramientas.cargarImagen("imagenes/casadegnomos.png"),
                                                        anchoDeResolucion / 2, altoDeResolucion / 15, 0, 3);
                                }
                                // Se reinicia la variable 'algunGnomoNoExiste' despues de medio segundo
                                // de que se detectara que un gnomo no existe
                                if (this.algunGnomoNoExiste && this.entorno.tiempo() - this.milisegundosHastaAhora >= 500) {
                                        this.algunGnomoNoExiste = false;
                                }
                        }
                        // Comprueba si las islas existen
                        // y dibuja cada isla en pantalla
                        if (this.islasFlotantes != null) {
                                for (int i = 0; i < islasFlotantes.length; i++) {
                                        this.islasFlotantes[i].dibujar(this.entorno, this.altoDeResolucion);
                                }
                        }
                        // Comprueba si Pep existe y si
                        // no se cayo al vacio
                        if (this.pep != null) {
                                // Dibuja a Pep
                                this.pep.dibujar(this.entorno, this.altoDeResolucion, this.disparo != null, this.pepCamina);
                                // Se reinicia la variable 'pepCamina' si Pep deja de caminar
                                if (this.pep.getX() == this.xDePepAnterior) {
                                        this.pepCamina = false;
                                }
                                detectarMovimientosDePep();
                                // Se crea un disparo si actualmente no hay uno
                                // y se presiona la tecla/boton correspondiente
                                if (this.disparo == null
                                                && (this.entorno.sePresiono('c') ||
this.entorno.sePresionoBoton(this.entorno.BOTON_IZQUIERDO))) {
                                        this.disparo = new Disparo(this.pep.getX(), this.pep.getY(), this.pep.getAncho(),
this.pep.getAncho(),
                                                        5);
                                        // Si Pep disparo, se guardan la posicion y direccion
                                        if (this.disparo != null) {
                                                this.xDePepCuandoDisparo = this.pep.getX();
                                                this.pepMirabaALaDerechaCuandoDisparo = this.pep.getMiraAlaDerecha();
                                        }
                                }
                                // Si el disparo existe, lo dibuja
                                if (this.disparo != null) {
```

```
                                        this.disparo.dibujar(this.entorno, this.altoDeResolucion,
this.pepMirabaALaDerechaCuandoDisparo);
                                        // this.disparo.dibujar(this.entorno);
                                        // Mueve el disparo dependiendo de la direccion
                                        // a la cual Pep miraba
                                        if (this.pepMirabaALaDerechaCuandoDisparo) {
                                                this.disparo.moverALaDerecha();
                                        } else {
                                                this.disparo.moverALaIzquierda();
                                        }
                                        // Condiciones para que el disparo desaparezca
                                        if (this.disparo.getX() <= 0 || this.disparo.getX() >= this.anchoDeResolucion
                                                        || this.disparo.getX() > this.xDePepCuandoDisparo + (anchoDeResolucion /
6)
                                                        || this.disparo.getX() < this.xDePepCuandoDisparo - (anchoDeResolucion /
6)) {
                                                this.disparo = null;
                                        }
                                }
                                // Aplica gravedad a menos que haya colision
                                this.pep.aplicarGravedad();
                                this.pep.colisionConIslas(islasFlotantes);
                                // Comprueba si Pep se cayo al vacio
                                // y elimina todo en caso de ser asi
                                if (this.pep.getY() - (this.pep.getAlto() / 2) > this.altoDeResolucion) {
                                        this.pepEstaVivo = false;
                                }
                        }
                        // Comprueba si los gnomos existen
                        if (this.gnomos != null) {
                                for (int i = 0; i < gnomos.length; i++) {
                                        if (gnomos[i] == null) {
                                                // Crear un nuevo gnomo si el actual es null
                                                gnomos[i] = new Gnomo(this.anchoDeResolucion / 2, this.altoDeResolucion / 10,
                                                                this.altoDeResolucion / 20, this.altoDeResolucion / 20, 2);
                                                continue; // Saltar a la siguiente iteración después de crear el nuevo gnomo
                                        }
                                        if (gnomos[i] != null) {
                                                // Movimiento de los gnomos
                                                gnomos[i].moverHaciaCentro(anchoDeResolucion);
                                                gnomos[i].aplicarGravedad();
                                                gnomos[i].dibujar(this.entorno, altoDeResolucion);
                                                // Verificar si el gnomo ha sido salvado por Pep
                                                if (gnomos[i].estaCercaDePep(this.pep) && gnomos[i].getY() > this.altoDeResolucion /
3) {
                                                        gnomos[i] = null; // Gnomo salvado, eliminarlo del juego

this.estadoDelJuego.setGnomosSalvados(this.estadoDelJuego.getGnomosSalvados() + 1); // Sumarlo a


                                // la


                                // cantidad


                                // de
```

```
                                        // gnomos


                                        // salvados
                                                continue; // Saltar a la siguiente iteración para evitar otros métodos en este
ciclo
                                        }
                                        // Verificar si el gnomo ha caído al vacío
                                        if (gnomos[i].getY() > altoDeResolucion) {
                                                gnomos[i] = null; // Eliminar gnomo caído

this.estadoDelJuego.setGnomosPerdidos(this.estadoDelJuego.getGnomosPerdidos() + 1); // Sumarlo a


                                        // la


                                        // cantidad


                                        // de


                                        // gnomos


                                        // perdidos
                                                continue; // Saltar a la siguiente iteración para evitar otros métodos en este
ciclo
                                        }
                                        // Verifica colisión con islas solo si el gnomo no es null
                                        gnomos[i].colisionConIslas(this.islasFlotantes);
                                        // Verificar si el gnomo ha sido aniquilado por una tortuga
                                        if (this.gnomos[i] != null) {
                                                for (int j = 0; j < tortugas.length; j++) {
                                                        if (this.tortugas[j] != null && this.gnomos[i] != null
                                                                        &&
this.gnomos[i].estaCercaDeTortuga(this.tortugas[j])) {
                                                                gnomos[i] = null; // Gnomo perdido, eliminarlo del
juego

this.estadoDelJuego.setGnomosPerdidos(this.estadoDelJuego.getGnomosPerdidos() + 1); // Sumarlo


                                        // a


                                        // la


                                        // cantidad


                                        // de
```

```
                                        // gnomos


                                        // perdidos
                                                continue; // Saltar a la siguiente iteración para evitar
otros métodos en este ciclo
                                            }
                                        }
                                    }
                                }
                            }
                            // Tortugas
                            if (this.tortugas != null) {
                                for (int i = 0; i < tortugas.length; i++) {
                                    if (tortugas[i] != null) {
                                        tortugas[i].aplicarGravedad();
                                        tortugas[i].dibujar(this.entorno, altoDeResolucion);
                                        tortugas[i].colisionConIslas(this.islasFlotantes);
                                        // La tortuga cae en isla 1:
                                        if (tortugas[i].getX() > (anchoDeResolucion / 2) - ((anchoDeResolucion / 2) / 3)
                                                - (anchoDeResolucion / 12) && tortugas[i].getX() <
islasFlotantes[0].getX()) {
                                            tortugas[i].rebotarTortugas(anchoDeResolucion,
                                                    (anchoDeResolucion / 2) - ((anchoDeResolucion / 2)
/ 3) - (anchoDeResolucion / 12),
                                                    (anchoDeResolucion / 2) - ((anchoDeResolucion / 2)
/ 3) + (anchoDeResolucion / 12));
                                            if (tortugas[i].getY() + (tortugas[i].getAlto() / 2) == islasFlotantes[1].getY()
                                                    - (islasFlotantes[1].getAlto() / 2)) {
                                                tortugas[i].setVelocidad(1);
                                            }
                                            // La tortuga cae en isla 2:
                                        } else if (tortugas[i].getX() > islasFlotantes[0].getX()
                                                && tortugas[i].getX() < (anchoDeResolucion / 2) +
((anchoDeResolucion / 2) / 3)
                                                        + (anchoDeResolucion / 12)) {
                                            tortugas[i].rebotarTortugas(anchoDeResolucion,
                                                    (anchoDeResolucion / 2) + ((anchoDeResolucion /
2) / 3) - (anchoDeResolucion / 12),
                                                    (anchoDeResolucion / 2) + ((anchoDeResolucion /
2) / 3) + (anchoDeResolucion / 12));
                                            if (tortugas[i].getY() + (tortugas[i].getAlto() / 2) == islasFlotantes[2].getY()
                                                    - (islasFlotantes[1].getAlto() / 2)) {
                                                tortugas[i].setVelocidad(1);
                                            }
                                            // La tortuga cae en isla 3:
                                        } else if (tortugas[i].getX() < islasFlotantes[1].getX()) {
                                            tortugas[i].rebotarTortugas(anchoDeResolucion,
                                                    ((anchoDeResolucion / 2) - ((anchoDeResolucion /
2) / 3) * 2)
                                                            - (anchoDeResolucion / 12),
                                                    ((anchoDeResolucion / 2) - ((anchoDeResolucion /
2) / 3) * 2)
                                                            + (anchoDeResolucion / 12));
                                            if (tortugas[i].getY() + (tortugas[i].getAlto() / 2) == islasFlotantes[3].getY()
```

```
                                                                                - (islasFlotantes[1].getAlto() / 2)) {
                                                                        tortugas[i].setVelocidad(1);
                                                                }
                                                                // La tortuga cae en isla 5:
                                                        } else if (tortugas[i].getX() > islasFlotantes[2].getX()) {
                                                                tortugas[i].rebotarTortugas(anchoDeResolucion,
                                                                                ((anchoDeResolucion / 2) + ((anchoDeResolucion /
2) / 3) * 2)
                                                                                                - (anchoDeResolucion / 12),
                                                                                ((anchoDeResolucion / 2) + ((anchoDeResolucion /
2) / 3) * 2)
                                                                                                + (anchoDeResolucion / 12));
                                                                if (tortugas[i].getY() + (tortugas[i].getAlto() / 2) == islasFlotantes[5].getY()
                                                                                - (islasFlotantes[1].getAlto() / 2)) {
                                                                        tortugas[i].setVelocidad(1);
                                                                }
                                                        }
                                                        // Si la tortuga choca con Pep:
                                                        if (tortugas[i].estaCercaDePep(this.pep)) {
                                                                this.pepEstaVivo = false;
                                                        }
                                                        // Si un disparo toca a una tortuga:
                                                        if (disparo != null) {
                                                                if (tortugas[i].intersectaConDisparo(disparo)) {
                                                                        tortugas[i] = null;
                                                                        tortugas[i] = this.crearNuevaTortuga();

this.estadoDelJuego.setEnemigosEliminados(this.estadoDelJuego.getEnemigosEliminados() + 1);
                                                                }
                                                        }
                                                }
                                        }
                                }
                                // Comprueba si pep y los gnomos existen
                                if (this.pep != null && this.gnomos != null) {
                                        // X actual del mouse
                                        int mouseX = this.entorno.mouseX();
                                        // Manteniendo presionado el boton derecho del mouse,
                                        // se puede mover la nave hacia los lados
                                        if (this.entorno.estaPresionado(this.entorno.BOTON_DERECHO)) {
                                                this.nave.moverConCursor(mouseX);
                                        }
                                        this.nave.dibujar(this.entorno);
                                        // Permite que Pep y los gnomos se paren sobre la nave
                                        this.nave.colisionConNave(this.pep, this.gnomos);
                                }
                        }
                        private Tortuga crearNuevaTortuga() {
                                int islaAleatoria = 0;
                                while (islaAleatoria == 0 || islaAleatoria == 4
                                                || this.islasFlotantes[islaAleatoria].tieneTortuga(this.tortugas)) {
                                        islaAleatoria = (int) Math.floor(Math.random() * 6);
                                }
                                Tortuga nuevaTortuga = new Tortuga(islasFlotantes[islaAleatoria].getX(), 0, this.altoDeResolucion / 20,
                                                this.altoDeResolucion / 20, 0, islaAleatoria);
                                nuevaTortuga.setVelocidadDeCaida(2);
                                return nuevaTortuga;
                        }
```

```
	public void limpiarPantalla() {
		this.pep = null;
		this.islasFlotantes = null;
		this.gnomos = null;
		this.tortugas = null;
	}
	public void detectarMovimientosDePep() {
		// Movimiento hacia la derecha
		boolean sePresionoLaTeclaDerecha = this.entorno.estaPresionada(this.entorno.TECLA_DERECHA)
				|| this.entorno.estaPresionada('d');
		if (sePresionoLaTeclaDerecha) {
			this.pep.moverDerecha();
			this.pepCamina = true;
			this.xDePepAnterior = this.pep.getX();
		}
		// Movimiento hacia la izquierda
		boolean sePresionoLaTeclaIzquierda = this.entorno.estaPresionada(this.entorno.TECLA_IZQUIERDA)
				|| this.entorno.estaPresionada('a');
		if (sePresionoLaTeclaIzquierda) {
			this.pep.moverIzquierda();
			this.pepCamina = true;
			this.xDePepAnterior = this.pep.getX();
		}
		// Salto
		boolean sePresionoLaTeclaArriba = this.entorno.estaPresionada(this.entorno.TECLA_ARRIBA)
				|| this.entorno.estaPresionada('w');
		if (sePresionoLaTeclaArriba) {
			this.pep.saltar();
		}
	}
	@SuppressWarnings("unused")
	public static void main(String[] args) {
		Juego juego = new Juego();
	}
}
```

Conclusión

Durante el desarrollo del trabajo práctico aprendimos formas de organización para poder dividir el trabajo, así como optimizar el trabajo reutilizando código que ya está en otras partes del código para resolver problemas relacionados o implementar clases parecidas, también pudimos fortalecer conocimientos que teníamos pero no habíamos llegado a implementarlos en un entorno más realista y teniendo en cuenta los cambios que se podrían a llegar a necesitar en un hipotético futuro.