TP Final Programación 1 Com.2

Al rescate de los gnomo

Diego Arias - 94941698 - Diegoalexisarag@gmail.com
Federico Lafuente - 39462570 - biersack.sixx14@gmail.com
Maylen Michel - 46498118 - maylenmichel220@gmail.com

Introducción

Para este trabajo se desarrollará un juego titulado "Al rescate de los gnomos", donde el objetivo principal es que el jugador controla al personaje "Pep" para rescatar a los gnomos en peligro. Estos gnomos se encuentran atrapados en una serie de islas flotantes y se desplazan continuamente de un lado a otro. Sin embargo, no están solos: las islas también están habitadas por tortugas, las cuales representan una amenaza constante para los gnomos.

La dinámica del juego consiste en que Pep debe moverse por las islas y salvar a los gnomos mediante colisiones. Los gnomos pueden ser rescatados solo en las islas más bajas, ya que en las superiores están fuera del alcance de colisión del jugador. A su vez, Pep debe evitar colisionar con las tortugas y sus bombas ya que estas le restan vidas al jugador y si se pierden todas las vidas o no se recatan cierta cantidad de gnomo se termina el juego.

Este trabajo integrador brinda una oportunidad de aplicar conocimientos de forma creativa y divertida en un contexto de desarrollo de videojuegos, permitiendo a los estudiantes experimentar con la lógica de programación en un entorno interactivo y dinámico, al mismo tiempo que fortalecen sus habilidades técnicas y su capacidad para la resolución de problemas. A través de este proyecto, se espera que los estudiantes logren una comprensión más profunda de la programación orientada a objetos, mientras desarrollan un juego desafiante y entretenido.

Descripción e Implementación

Clase: Juego

```
Juego()
	{
		// Inicializa el objeto entorno
		this.entor = new Entorno(this, "Proyecto para TP", 800, 600);
		this.jugador = new Jugador(270.0,455.0, entor);
		this.islas = new Isla[15];
		int k = 1;
		for(int i = 1; i<6;i++) {
			if(i==1) {
				islas[0] = new Isla((1)*this.entor.ancho()/(1+1), 100*1, entor,0.34);
			} else {
				for( int j =1; j <= i;j++) {
				islas[k++] = new Isla((j)*this.entor.ancho()/(i+1), 100*i, entor,1.3/(i+2));
					}
				}
		}
		this.gnomo = new Gnomo[4];
		this.bombas = new Bomba[1];
		this.tortugas = new Tortuga[10];
		this.ataques = new Ataque[3];
		this.fondoCielo = entorno.Herramientas.cargarImagen("cieloPrueba.png");
		this.fondoMuerte = entorno.Herramientas.cargarImagen("die.jpg");
		this.escudo = new Escudo(this.entor,this.jugador,3);
		// Inicializar lo que haga falta para el juego
		// ...

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

Esta variable inicializa tanto el juego como el entorno y todo lo que necesita para su funcionamiento base.

```
public void tick()
```

Durante el juego el método tick() será ejecutado en todo momento, por lo tanto, es el método más importante de esta clase, ya que es en donde se encuentras todo los códigos que hacen posible la correcta ejecución de las acciones de las tortugas, de los gnomos y de Pep.

```
private void colisionFuegoBomba(Ataque[] a, Bomba[] b)
```

En esta función cada vez que un fuego de Pep colisiona con una bomba de tortuga causando que ambas desaparezcan en el momento.

```
private boolean colisionBombaJugador(Bomba[] bomba, Jugador jugador)
```

Si Pep colisiona con una tortuga o una bomba de tortuga respectivamente, provoca que el jugador pierda una de sus vidas. Aunque, se puede evitar la colisión con una bomba de tortuga si Pep utiliza el escudo.

```
private void tortugasRebote(Tortuga tor, Isla islaDeLaTortuga)
```

Esta función se utiliza para que las tortugas se mantengan en la isla en la que caen, moviéndose de izquierda a derecha y viceversa hasta que Pep las elimine.

```
private void bombaFueraDePantalla(Bomba[] bom)

private void ataqueFueraDePantalla(Ataque[] ata)
```

Ambas funciones se utilizan para que tanto el fuego de Pep como las bombas de tortugas desaparezcan si pasan el límite la pantalla del juego sin colisionar con ningún objetivo.

```
private void tortugaMuere(Tortuga[] t, Ataque[] ata)
```

Cuando la tortuga colisiona con un fuego de Pep, esta función provoca que la tortuga “muera” y el fuego desaparezcan, es decir, se vuelven null.

```
private void nuevoAtaque()
```

Esta función genera un nuevo objeto ataque mientras que este sea null, es decir, solo puede haber un ataque a la vez.

```
private void nuevaTortuga(Tortuga[] tortu)
```

Esta función rellena los espacios nulos del arreglo de tortugas para crear una nueva tortuga de manera random.

```
private Isla islaDeLaTortuga(Tortuga[] tortu, Isla[] i)
```

Si la totuga esta apoyada en una isla, devuelve la isla en la que esta apollada.

```
private void salto()
```

Si el jugador toca la flecha de arriba, se guarda el segundo en el que se guarda y se cambia al estado de saltando.

```
public boolean estaApoyado(Jugador j, Isla[] i)

public boolean estaApoyado(Jugador j, Isla i)
```

La primera función utiliza a la segunda y retornan True si Pep está pisando una isla del arreglo, de lo contrario retorna False.

```
public boolean estaApoyado(Tortuga t, Isla i)

public boolean estaApoyado(Tortuga j, Isla[] i)
```

La primera función utiliza a la segunda y retornan True si una tortuga está pisando alguna isla del arreglo, de lo contrario retorna False.

```
private void nuevoGnomo(Gnomo[] gnom)
```

Esta función rellena los espacios nulos del arreglo de gnomo para crear un nuevo gnomo, a diferencia del arreglo de tortugas, solo pueden existir 4 gnomos en la pantalla al mismo tiempo.

```
public boolean estaApoyado(Gnomo g, Isla i)

public boolean estaApoyadoGnomo(Gnomo n, Isla[] i)
```

Estas funciones retornan True si un gnomo está pisando alguna isla del arreglo, de lo contrario retorna False.

```
public Isla islaDelGnomo(Gnomo g, Isla[] i)

private boolean estaEnLaIsla(Gnomo g, Isla i)
```

Estas funciones retornan true si y solo si el gnomo está apoyado en alguna de las islas.

```
public boolean colisionBombaGnomo(Bomba[] bomba, Gnomo gnomo)

public boolean tortugaColicionGnomo(Tortuga[] tortugas, Gnomo gnomo)
```

Ambas funciones retornan true si un gnomo colisiona con una tortuga o una bomba de tortuga respectivamente, provocando que desaparezca, es decir, se vuelva null y se guarde como gnomo perdido.

```
public boolean jugadorColicionGnomo(Jugador jugador, Gnomo gnomo)
```

Esta esta función retorna true si un gnomo colisiona con Pep, provocando que el gnomo se anule y se guarde como gnomo rescatado.

```
private void tirarBomba(Bomba[] bom, Tortuga tor)
```

Esta función se ejecuta de manera random mostrando en la pantalla del juego las bombas de tortuga que colisionan con el jugador o los gnomos.

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

Ejecuta el programa de forma repetitiva.

```
private boolean tortugaColicionJugador(Tortuga[] tortugas, Jugador jugador)
```

Recorre un array de tortugas, si el jugador colisiona con una tortuga retorna true, si no false.

```
private boolean colisionBombaJugador(Bomba[] bomba, Jugador jugador)
```

Recorre un array de bombas y cuando un objeto bomba de un array de bombas colisiona con el jugador devuelve true, si no, false.

```
public void tiempo(int posicionX, int posicionY, int tiempo, Color col)
private void gnomosPerdidos(int posicionX)
private void gnomosSalvados(int posicionX)
private void tortugasEliminadas(int posicionX)
private void mostrarUsosEscudo()
private void mostrarVida()
```

Estas funciones muestran en pantalla el tiempo transcurrido, la cantidad de gnomos perdidos, la cantidad de gnomos rescatados, la cantidad de tortugas eliminadas, la cantidad de usos que le quedan al escudo y la cantidad de vida del jugador, respectivamente.

```
private void proteccionDelEscudo(Escudo es, Bomba[] bom)
```

Recorre un array de bombas no nulas, si el escudo colisiona con una bomba del array de bombas, se resta un uso al escudo y la bomba se vuelve null.

Clase: Jugador

```
public class Jugador{
        private double x,y;
        private int velocidad;
        private boolean direccion;
        private double ancho;
        private double alto;
        private double escala;
        boolean saltando;
        Image Izq;
        Image Der;
        Entorno e;
        boolean apoyado;
        Juego j;

        public Jugador(double x, double y, Entorno e) {
                this.x = x;
                this.y = y;
                this.e = e;
                this.direccion = false;
                this.velocidad = 2;
                this.escala =0.35;
                this.Izq = entorno.Herramientas.cargarImagen("personajePrueba2.png");
                this.Der = entorno.Herramientas.cargarImagen("personajePrueba.png");
                this.apoyado = false;
                this.alto = this.Izq.getHeight(null) * this.escala;
                this.ancho = this.Izq.getWidth(null) * this.escala;
        }
        public void moverIzquierda()
        public void moverDerecha() //si se choca con el borde para

        //el jugador caerá simulando gravedad
        public void gravedad()

        //Dibujar Jugador
        public void dibujar()

        public void saltando(int momentoDeSalto, int i)

        public boolean seCayoJugador()
                return false;

        public double getBorderSuperior()// retorna el borde de la parte superior de la imagen

        public double getBorderInferior()// retorna el borde de la parte inferior de la imagen

        public double getBorderIzquierdo()// retorna el borde de la parte izquierda de la imagen

        public double getBorderDerecho()// retorna el borde de la parte derecha de la imagen

        public double getX() {return x;}
        public double getY() {return y;}
        public void setY(double y)

        public boolean getDireccion()

}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno del jugador o “Pep”, incluyendo movilidad, gravedad en la caída y saltos, diseño del personaje, punto de spawn, las vidas, etc.

Clase: Tortugas

```
public class Tortuga {
        double x,y;
        private double velocidad;
        private double ancho;
        private double alto;
        private double escala;
        boolean direccion;
        Entorno e;
        Image Izq;
        Image Der;
        boolean apoyado;

        public Tortuga(Entorno e) {
                this.x = spawnRandom(e);
                this.y = -20;
                this.e = e;
                this.direccion = false;
                this.velocidad = 0.5;
                this.escala =0.09;
                this.Izq = entorno.Herramientas.cargarImagen("enemigoPrueba.png");
                this.Der = entorno.Herramientas.cargarImagen("enemigoPrueba.png");
                this.apoyado = false;
                this.alto = this.Izq.getHeight(null) * this.escala;
                this.ancho = this.Izq.getWidth(null) * this.escala;
        }
        public void gravedad()

        //Dibujar Tortuga
                public void dibujar()

        //Elige unas coordenadas aleatorioas din contar el "medio"
        private int spawnRandom(Entorno e)

        public void movimientoX()

        public double getBorderSuperior()// retorna el borde de la parte superior de la imagen


        public double getBorderInferior() // retorna el borde de la parte inferior de la imagen


        public double getBorderIzquierdo()// retorna el borde de la parte izquierda de la imagen

        public double getBorderDerecho() // retorna el borde de la parte derecha de la imagen

        public double getX()

        public double getY()

        public boolean getDireccion()

}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno de las tortugas, incluyendo movilidad, gravedad, diseño del personaje, punto de spawn aleatorio, etc.

Clase: Gnomo

```
public class Gnomo  {
    Entorno e;
    private Gnomo gnomos;
    private Jugador Jugador;
    private ArrayList<Tortuga> Tortugas;
    double x,y;
        private double velocidad;
        private double ancho;
        private double alto;
        private double escala;
        boolean direccion;
        Image Izq;
        Image Der;
        boolean apoyado;
        boolean direccionAleatoria;

    public Gnomo(Entorno e) {
                this.x = 400;
                this.y = 40;
                this.e = e;
                this.direccion = false;
                this.direccionAleatoria = true;
                this.velocidad = 0.5;
                this.escala =0.04;
                this.Izq = entorno.Herramientas.cargarImagen("planton(gnomo)d.png");
                this.Der = entorno.Herramientas.cargarImagen("planton(gnomo).png");
                this.apoyado = false;
                this.alto = this.Izq.getHeight(null) * this.escala;
                this.ancho = this.Izq.getWidth(null) * this.escala;
    }


    public void dibujar()

    public void gravedad()

        public void movimientoX()

        public boolean seCayoGnomo()

        public  double getBorderSuperior()// retorna el borde de la parte superior de la imagen

        public double getBorderInferior()// retorna el borde de la parte inferior de la imagen

        public double getBorderIzquierdo()// retorna el borde de la parte izquierda de la imagen

        public double getBorderDerecho() // retorna el borde de la parte izquierda de la imagen

        public double getX()

        public double getY()

        public boolean getDireccion()

}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno de los gnomos, incluyendo movilidad, gravedad para la caída, diseño del personaje, punto de spawn, etc.

Clase: Ataque

```
public class Ataque {
	private double x,y;
	private double ancho;
	private double alto;
	private double escala;
	Image Izq;
	Image Der;
	Entorno e;
	Jugador j;
	private boolean direccion;

	public Ataque(Jugador j, Entorno e) {
		this.x = j.getX();
		this.y = j.getY()+5;
		this.j = j;
		this.escala = 0.03;
		this.e = e;
		this.direccion = j.getDireccion();
		this.Izq = entorno.Herramientas.cargarImagen("fuegoPrueba.png");
		this.Der =entorno.Herramientas.cargarImagen("fuegoPrueba2.png");
		this.alto = this.Izq.getHeight(null) * this.escala;
		this.ancho = this.Izq.getWidth(null) * this.escala;

	}
	public void movimientoX() {
		if(this.direccion) {
			this.x -= 4;
			this.e.dibujarImagen(this.Der, getX(), getY(), 0, escala);
		} else {
			this.x +=4;
			this.e.dibujarImagen(this.Izq, getX(), getY(), 0, escala);
		}
	}
	private double getY() {
		return this.y;
	}

	private double getX() {
		return this.x;
	}

	public double getBorderSuperior() {
// retorna el borde de la parte superior de la imagen
		return this.y - this.alto/2;
	}
	public double getBorderInferior() {
// retorna el borde de la parte inferior de la imagen
		return this.y + this.alto/2;
	}
	public double getBorderIzquierdo() {
// retorna el borde de la parte izquierda de la imagen
		return this.x - this.ancho/2;
	}
	public double getBorderDerecho() {
// retorna el borde de la parte izquierda de la imagen
		return this.x + this.ancho/2;
	}
}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno de los ataques del jugador o “Pep”, incluyendo las direcciones de movimiento, el diseño y el punto de spawn.

Clase: Bombas

```
public class Bomba {
	private double x,y;
	private double ancho;
	private double alto;
	private double escala;
	private Image imagen;
	Entorno e;
	Tortuga t;
	private boolean direccion;

	public Bomba(Tortuga t, Entorno e) {
		this.x = t.getX();
		this.y = t.getY()+5;
		this.t = t;
		this.escala = 0.08;
		this.e = e;
		this.direccion = t.getDireccion();
		this.imagen = entorno.Herramientas.cargarImagen("bombaPrueba.png");
		this.alto = this.imagen.getHeight(null) * this.escala;
		this.ancho = this.imagen.getWidth(null) * this.escala;

	}

	public void movimientoX() {
		if(this.direccion) {
			this.x -= 2;
		} else {
			this.x +=2;
		}
		this.e.dibujarImagen(this.imagen, getX(), getY(), 0, escala);
	}



	private double getY() {
		return this.y;
	}

	private double getX() {
		return this.x;
	}

	public double getBorderSuperior() {
// retorna el borde de la parte superior de la imagen
		return this.y - this.alto/2;
	}
	public double getBorderInferior() {
// retorna el borde de la parte inferior de la imagen
		return this.y + this.alto/2;
	}
	public double getBorderIzquierdo() {
// retorna el borde de la parte izquierda de la imagen
		return this.x - this.ancho/2;
	}
	public double getBorderDerecho() {
// retorna el borde de la parte izquierda de la imagen
		return this.x + this.ancho/2;
	}

}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno de las bombas de tortuga, incluyendo las direcciones de movimiento, el diseño y el punto de spawn.

Clase: Escudo

```
public class Escudo {
	private double x;
	private double y;
	private int usos;
	private double ancho;
	private double alto;
	private double escala;
	private boolean direccion;
	Image Izq;
	Image Der;
	Entorno e;
	Jugador j;

	public Escudo(Entorno e, Jugador j, int usos) {
		this.j = j;
		this.x = j.getX();
		this.y = j.getY();
		this.usos = usos;
		this.escala = 0.05;
		this.direccion = false;
		this.e = e;
		this.Izq = entorno.Herramientas.cargarImagen("escudoPrueba.png");
		this.Der = entorno.Herramientas.cargarImagen("escudoPrueba.png");
		this.ancho = this.Izq.getWidth(null) * this.escala;
		this.alto = this.Der.getHeight(null) * this.escala;
	}
	public void proteccionEscudo(boolean direc, Jugador j) {
		if(!direc) {
			this.direccion = true;
			this.x = j.getX() + 15;
			this.y = j.getY();
			} else {
			this.direccion = false;
			this.x = j.getX() - 15;
			this.y = j.getY();
		}
	}
	public void dibujar() {
		if(this.direccion) {
			this.e.dibujarImagen(this.Izq, getX(), getY(),0, escala);
		}
		this.e.dibujarImagen(this.Der, getX(), getY(), 0,escala);
	}
	private double getY() {
		return this.y;
	}
	private double getX() {
		return this.x;
	}
	public double getBorderSuperior() {
// retorna el borde de la parte superior de la imagen
		return this.y - this.alto/2;
	}
	public double getBorderInferior() {
// retorna el borde de la parte inferior de la imagen
		return this.y + this.alto/2;
	}
	public double getBorderIzquierdo() {
// retorna el borde de la parte izquierda de la imagen
		return this.x - this.ancho/2;
	}
	public double getBorderDerecho() {
// retorna el borde de la parte izquierda de la imagen
		return this.x + this.ancho/2;
	}
}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno del escudo del jugador o “Pep” incluyendo la protección del escudo, la durabilidad, el diseño y el punto de spawn.

Clase: Islas

```
public class Isla {
	private double x,y;
	private double ancho;
	private double alto;
	private double escala;
	Image isla;
	Entorno e;

	public Isla(double x, double y, Entorno e, double escala) {
		this.x = x;
		this.y = y;
		this.e = e;
		this.escala = escala;
		this.isla = entorno.Herramientas.cargarImagen("islaPrueba.png");
		this.alto = this.isla.getHeight(null) * this.escala -2;
		this.ancho = this.isla.getWidth(null) * this.escala - 10;
	}


	public Isla(Gnomo gnomo, String string, int i, int escala2) {
		// TODO Auto-generated constructor stub
	}


	public void dibujar() {
			this.e.dibujarImagen(this.isla, getX(), getY(), 0, escala);
	}


	public double getBorderSuperior() {
		return this.y - this.alto/2;
	}
	public double getBorderInferior() {
		return this.y + this.alto/2;
	}
	public double getBorderIzquierdo() {
		return this.x - this.ancho/2;
	}
	public double getBorderDerecho() {
		return this.x + this.ancho/2;
	}

	public double getX() {return x;}
	public double getY() {return y;}


	public void iniciar() {
		// TODO Auto-generated method stub

	}
}
```

Esta clase es la que ejecuta todas las acciones y configuraciones en el entorno de las islas, incluyendo su ubicación y diseño.

Algunos problemas encontrados

- El primer problema encontrado fue cuando se quería volver Null al objeto tortuga. Esto sucedió porque cuando se usó for each para iterar sobre la lista, no se permitió convertir al objeto de la lista en nulo. Se solucionó dicho problema al recorrer una lista usando un bucle For normal.
- Para cambiar el número de gnomos guardados necesarios para ganar, se debía modificar dos fragmentos del código. Esto se resolvió usando una variable privada que contiene la cantidad deseada de gnomos.
- En la pantalla final queríamos colocar un texto que muestre el tiempo que se tarda en perder o ganar. Una vez implementada, la hora mostrada continuaba ejecutándose en la pantalla final porque la función de hora es llamada todo el tiempo y actualizándose. Esto finalmente se resolvió guardando el momento exacto en que terminó el juego en una variable y llamando a la función usando esa variable fija.

Conclusión

Este trabajo nos ha proporcionado una oportunidad única para entender la importancia de la organización de un código, el manejo de interacciones entre múltiples objetos y la planificación de un flujo de juego continuo. Además, el juego nos permitió experimentar con la creación de instancias, uso de estructuras de control, manejo de arreglos y aplicación de algoritmos básicos. A través de este proyecto, hemos ganado una comprensión más sólida de la programación orientada a objetos y el desarrollo de juegos en un ambiente gráfico, sentando una base sólida.