Trabajo Práctico: Juego 2D - Gondolf y Murciélagos

**Integrantes:**Valentino Muto – valentinomuto050@gmail.com  
Felipe Martínez Tanoira – ¿  
Ignacio Juárez - ¿

# Introducción

Este trabajo práctico consiste en la implementación de un videojuego 2D con vista aérea, donde el jugador controla a un mago llamado “Gondolf” que debe sobrevivir a una horda de murciélagos. El juego está desarrollado en Java utilizando la biblioteca Entorno para gráficos y manejo de eventos. El objetivo es explorar conceptos de programación orientados a objetos, manejo de sprites, animaciones, colisiones, eventos, etc.…

# Descripción de Clases

A continuación, se describen:

* clases implementadas en el proyecto
* variables de instancia
* métodos principales
* la lógica detrás de cada una.

## Clase Gondolf

Personaje principal controlado por el jugador.  
Variables de instancia:  
- x, y: Posición del personaje en el mundo.  
- velocidad: Velocidad de movimiento.  
- imagen: Sprite principal del personaje.  
- dirección: Dirección a la que mira (arriba, abajo, izquierda, derecha).  
- vida: Puntos de vida.  
- proyectiles: Lista de proyectiles disparados.  
  
Métodos principales:  
- mover(): Actualiza la posición según entrada del jugador.  
- dibujar(): Dibuja el personaje en pantalla.  
- atacar(): Crea y lanza un proyectil en la dirección del mouse.  
- recibirDaño(): Reduce la vida ante un ataque.  
- colision(): Verifica colisiones con enemigos y obstáculos.

## Clase Murciélago

Enemigo que persigue al jugador.  
Variables de instancia:  
- x, y: Posición.  
- tamaño: Tamaño para colisiones.  
- velocidad: Velocidad de movimiento.  
- activo: Estado de vida (vivo o muerto).  
- vida: Puntos de vida.  
- sprites para cada dirección y animación.  
- dirección, frame, contadorFrames para animar.  
  
Métodos principales:  
- mover(): Se mueve hacia la posición de Gondolf.  
- dibujar(): Dibuja la animación según la dirección.  
- recibirDaño(): Resta vida y desactiva si llega a cero.  
- desactivar(): Cambia estado a inactivo.  
- getRectangulo(): Devuelve la hitbox para colisiones.

## Clase Proyectil

Representa un proyectil disparado por Gondolf.  
Variables de instancia:  
- x, y: Posición.  
- tamaño, velocidad, daño.  
- activo: Estado del proyectil.  
- sprites para animación.  
- direcciónX, direcciónY: Vector normalizado de movimiento.  
  
Métodos principales:  
- mover(): Actualiza posición en dirección hacia el objetivo.  
- dibujar(): Dibuja el proyectil animado.  
- desactivar(): Cambia estado a inactivo.  
- getRectangulo(): Hitbox para colisiones.

## Clase Zap

Representa un hechizo de rayo que puede ser lanzado por Gondolf.  
Variables de instancia:  
- x, y: Posición.  
- tamaño, daño, duración.  
- casteado: Estado de activación.  
- direcciónX, direcciónY para movimiento.  
- sprites para animación y botón para menú.  
  
Métodos principales:  
- castear(): Activa el hechizo.  
- mover(): Mueve el hechizo mientras está activo.  
- dibujarse(): Dibuja la animación.  
- descastear(): Desactiva el hechizo.  
- getHitbox(): Devuelve la zona de colisión.

## Clase Obstáculo

Los obstáculos fijos en el mapa que bloquean el paso (Barriles).  
Variables de instancia:  
- x, y: Posición.  
- imagen: Sprite del obstáculo.  
- hitbox: Área para colisiones.  
  
Métodos principales:  
- dibujar(): Dibuja el obstáculo.  
- getHitbox(): Devuelve el área de colisión.

## Clase Menú

Muestra la interfaz gráfica con botones para seleccionar hechizos.  
Variables de instancia:  
- x, y, ancho, alto: Dimensiones y posición del menú.  
- imagenFondo, imagenMenuHechizos: Imágenes de fondo.  
- objetos FireBall y Zap y sus botones asociados.  
  
Métodos principales:  
- dibujarMenu(): Dibuja el menú y los botones.  
- dibujarBotones(): Dibuja cada botón en el menú.

# Problemas encontrados y soluciones

- Carga de imágenes: Manejo de excepciones para evitar caídas si no se encuentran recursos.  
- Normalización de vectores para movimiento uniforme.  
- Control de animaciones mediante frames y contador para evitar cambios bruscos.  
- Colisiones: Se usaron Rectángulos para detección precisa.  
- Gestión del ciclo de vida de objetos (activación/desactivación).

# Implementación

A continuación, se presentan fragmentos clave del código fuente, con comentarios para facilitar su comprensión.

public class Gondolf {  
 private int x, y;  
 private int velocidad;  
 private Image imagen;  
 private String direccion;  
 private int vida;  
 // Otros atributos y listas para proyectiles  
  
 public Gondolf(int x, int y) {  
 this.x = x;  
 this.y = y;  
 this.velocidad = 3;  
 this.vida = 10;  
 // Cargar imagen y otros atributos  
 }  
  
 public void mover() {  
 // Lógica para mover según teclas presionadas  
 }  
  
 public void dibujar(Entorno e) {  
 e.dibujarImagen(imagen, x, y, 0);  
 }  
  
 public void atacar(int mouseX, int mouseY) {  
 // Crear proyectil hacia posición mouse  
 }  
  
 public void recibirDaño(int cantidad) {  
 vida -= cantidad;  
 if (vida <= 0) {  
 // Morir o reiniciar  
 }  
 }  
}

# Conclusiones

Durante el desarrollo de este juego, pudimos profundizar en conceptos fundamentales de programación orientados a objetos, manejo de gráficos y animaciones, así como la lógica de interacción entre objetos del juego. A pesar de que nos costó poder dominar ciertos conceptos, pudimos finalmente aprender a manejar la carga y gestión de recursos, implementación de colisiones y control de estados. El proyecto resultó una experiencia valiosa para entender la arquitectura de un juego 2D y el uso de herramientas Java.