

Programación I - Primer Semestre 2025

Trabajo Práctico:

El camino de Gondolf

Estudiantes:

Ozore, Naara Ariana -46905681- naharaozore@gmail.com

Ramirez Prado, Uriel Franco -47008921- urielfl123@hotmail.com

Fecha de entrega: 02.06.2025

El objetivo de este trabajo práctico es desarrollar un video juego en el cual controlamos al mago Gondolf. En el juego se busca sobrevivir al ataque de los murciélagos y derrotarlos usando distintos hechizos.

INTRODUCCION

Las consignas del trabajo práctico establecen los elementos esenciales que deben implementarse en el videojuego "El camino de Gondolf". Éstas incluyen aspectos como la interfaz de usuario, la mecánica de juego, el comportamiento de los enemigos y las reglas de interacción.

Movimiento del mago: Se controla con las teclas 'w', 'a', 's' y 'd' o las flechas del teclado. No puede salir de los límites de la pantalla ni estar en el área del menú.

Obstáculos: Rocas en el juego que bloquean el movimiento del mago, pero que los enemigos pueden volar sobre ellas.

Enemigos: Murciélagos que aparecen aleatoriamente desde los bordes de la pantalla y persiguen a Gondolf. Se debe establecer una cantidad total de enemigos a matar y un número máximo de cuántos hay en simultáneo en pantalla.

Ataques y hechizos: Gondolf lanza hechizos con su energía mágica. Debe haber al menos dos tipos de hechizos, uno de ellos sin costo de energía. Los hechizos eliminan enemigos dentro de su área de efecto.

CLASES

Juego

Variables de instancia:

- entorno: Controla la ventana del juego.
- menu: Menú al costado de la pantalla con la información del juego. También se encuentran los menú de inicio y fin de partida ya sea gana o perdida.
- mago: Instancia del mago.
- rocas, murciélagos, pociones: Listas de objetos en el juego.
- hechizos: Lista de hechizos disponibles.
- botones: Botones del menú.
- sonido: Efectos de sonido del juego.
- enemigosGenerados: Nos ayuda a la hora de ponerle un fin al juego, ya que cuenta cuántos enemigos se generan.

Métodos:

- tick(): Actualiza el estado del juego en cada frame.
- dibujarObjetos(): Dibuja todos los objetos en pantalla.

- spawnMurcielago(): Genera murciélagos en posiciones aleatorias con un random. Acá se suman los enemigos generados.
- mantenerMurcielagos(): Mantiene un mínimo de 10 murciélagos en pantalla sumando 1 cada que spawnea uno.
- recibirDanoMago(int cantidad): Llama a método recibirDano(cant) de mago para reducir su vida, y ajusta el botón de vida que aparece en el menú proporcionalmente a la vida del mago.
- restaurarVidaMago(int cantidad): Funciona de manera similar a recibirDanoMago, solo que restaura la vida del mago.
- gastarMana(int cantidad): Funciona como recibirDanoMago, pero quita mana.
- sumarMana(int cantidad): Funciona como gastarMana pero agrega maná.
- generarPoti(ArrayList<Potion>pociones): Con un random genera el x e y dela poción (fuera del menú) y son otro random se decide si es de vida o maná.
- seGenero(): Genera una poción de manera aleatoria con un 0.3 de probabilidad llamando a generarPoti.
- dibujarMenuInicio() : dibuja todo lo que aparece en pantalla antes de iniciar el juego.
- salirMenu() : inicia el juego cuando se presiona el botón en el menú de inicio.
- dibujarGameOver(): Le comunica al jugador que perdió.
- dibujarGameWin(): Le comunica al jugador que ganó.

Problemas y Soluciones

Problema: Las colisiones no se detectaban correctamente debido a cálculos imprecisos.

Solución: Se implementaron métodos específicos para cada tipo de colisión (con rocas, murciélagos, etc.) usando distancias y coordenadas.

Problema: La vida y el maná no se actualizaban visualmente en tiempo real.

Solución: Se ajustó el ancho de los botones de vida y maná para reflejar los valores actuales.

Problema: Los murciélagos no aparecían de manera equilibrada.

Solución: Se implementó un sistema de generación aleatoria desde los bordes de la pantalla.

Mago

Variables de instancia:

- x, y: Coordenadas del mago.
- ancho, alto: Dimensiones del mago.
- color: Color del mago.
- hp, mana: Vida y maná actuales.
- hpMaxima, manaMaxima: Valores máximos de vida y maná.

- enemigosAniquilados: Cantidad de enemigos aniquilados por el mago
- enemigosColisionados: cantidad de murciélagos que colisionaron con el mago

Métodos:

- dibujar(Entorno entorno): Dibuja al mago.
- mover izquierda, derecha, arriba y abajo(Entorno entorno): Métodos para mover al mago en diferentes direcciones del eje x e y.
- colisionaPorDerecha, Izq, arriba y abajo(Entorno entorno): métodos que no permiten al mago salirse del límite de la pantalla
- colisionConRoca, murciélago y poción: Métodos para detectar colisiones con rocas, murciélagos y pociones.
- colisionConAlgunMurcielago(ArrayList<Murcielago> murcielagos, double xm, double ym): Método que vuelve null al murciélago si éste colisionó con el mago. Recorre murciélagos para ello.
- colisionConAlgunaPucion(ArrayList<Pucion> pociones, double xp, double yp): funciona como colisionConAlgunMurcielago, asignando null cuando el mago pasa sobre una poción.
- colisionConAlgunaRoca: Hace que el mago no pueda seguir en esa dirección por chocar con la roca. Necesita de colisionConRoca para obtener los datos de donde está mago y dónde está la roca.
- recibirDano(int cantidad): Reduce la vida del mago.
- gastarMana(int cantidad): Reduce el maná del mago.
- estaVivo(Mago mago): Método que verifica si el mago está con vida
- perseguirMago(ArrayList<Murcielago> murcielagos, Mago mago): Método que hace que los murciélagos vivos persigan continuamente al mago
- restaurarSalud o Mana(int cantidadARestaurar): Método que agrega Salud o Mana

Problemas y Soluciones

Problema: Al comienzo, el mago al colisionar con la roca, se quedaba completamente quieto y no se podía seguir moviendo. Las colisiones no se detectaban correctamente debido a cálculos imprecisos.

Solución: Desarrollamos colisionConAlgunaRoca para mayor precisión particular con cada roca.

Hechizo

Variables de instancia:

- nombre: Nombre del hechizo.
- x, y: Coordenadas del hechizo.
- costoMana: Costo de maná para lanzar el hechizo.

- 
- radioEfecto: Área de efecto del hechizo.
 - danio: Daño infligido.
 - color: Color del hechizo.

Métodos:

- dibujar(Entorno entorno): Dibuja el hechizo.
- puedeLanzarse(Entorno entorno, double anchoMenu, Boton boton): Verifica si el hechizo puede ser lanzado.
- lanzar(ArrayList<Murcielago> murcielagos, Entorno entorno, Boton boton, Mago mago): Aplica el efecto del hechizo.
- hacerDanio(ArrayList<Murcielago> murcielagos, double x, double y, Mago mago): Infinge daño a los murciélagos.
- relentizar(ArrayList<Murcielago> murcielagos, double x, double y, Mago mago): Reduce la velocidad de los murciélagos.
- lanzarRelentizar(ArrayList<Murcielago> murcielagos, Entorno entorno, Boton boton, Mago mago): Aplica el efecto del relentizar
- colisionConMurcielago(Murcielago m, double xHechizo, double yHechizo): Verifica si existe la colisión entre el murciélagos y el hechizo

Problemas y Soluciones

Problema: La construcción de hechizos no fue difícil. Primero no sabíamos si hacer un array fijo o un ArrayList.

Solución: Por su facilidad de manejo, decidimos que ArrayList era muchísimo más cómodo ya que lo aprendimos a utilizar con los murciélagos y tal vez crearíamos nuevos hechizos aparte de los obligatorios.

Problema: Fue complicado entender la relación entre botón y hechizos.

Solución: Mediante fuimos desarrollando el código, fuimos viendo cómo era más fácil ir asociando cada acción con cada clase y conseguimos una lógica clara.

Boton

Características y constructor de Botón

Como botón es un rectángulo, necesitamos x, y, ancho y alto. Además, para identificarlos mejor les pusimos un color y un string para nombrarlos. El boolean seleccionado es para mayor facilidad a la hora de asignarle un hechizo a un botón, ésto se lo agregamos luego, cuando estuvimos creando y perfeccionando hechizos.

Variables de instancia:

- x, y: Coordenadas del botón.
- ancho, alto: Dimensiones del botón.
- color: Color del botón.
- str: Texto del botón.
- seleccionado: Estado de selección.

Métodos:

- dibujar(Entorno entorno): Para dibujar un botón usamos el método dibujar rectángulo de entorno, como éste nos pide un ángulo, lo dejamos en 0 simplemente.
- estaSeleccionado(Entorno entorno, double xmouse, double ymouse, Boton b): Este método toma el entorno y la posición del mouse para definir con estaSobreBoton si se ha hecho click, retorna true en dicho caso.
- estaSobreBoton(Entorno entorno, double xMouse, double yMouse): Este método toma el entorno y la posición del mouse para calcular los bordes del botón y así retornar true o false cuando el mouse esté dentro del área del botón.

Problema: No sabíamos cómo implementar de manera más cómoda si la selección del botón estaba activa.

Solución: Despues de ir probando, realizando los hechizos nos dimos cuenta que era más fácil y cómodo tenerlo como una variable de instancia.

Murcielago

Variables de instancia:

- x, y: Coordenadas del murciélagos.
- alto, diametro: Dimensiones del murciélagos.
- color: Color del murciélagos.
- vivo: Estado de vida.
- hp: Vida del murciélagos.
- speed: Velocidad de movimiento.

Métodos:

- dibujar(Entorno entorno): Dibuja al murciélagos.
- mover izquierda, derecha, arriba y abajo(Entorno entorno): Métodos que permiten moverse al murciélagos en diferentes direcciones del eje x e y con respecto al mago.
- movimiento(double posicion, Mago mago): Mueve al murciélagos hacia el mago.
- recibirDano(int cantidad): Reduce la vida del murciélagos.

- 
- EstaVivo(Murcielago murcielagos): Método que verifica si el murciélagos esta vivo.

Problemas y Soluciones

Problema: Al comienzo, fue difícil definir dónde se debía desarrollar el movimiento del murciélagos, no sabíamos en qué clase ponerlo y quedó muy desprolijas ésta clase.

Solución: Despues de ir probando, logramos comprender mejor el código y estar más familiarizados con el mismo, logrando emprolijar y buscar todo de manera correcta.

Menu

Variables de instancia:

- x, y: Coordenadas del Menú.
- ancho, alto: Dimensiones del Menú.
- color: Color del Menú.

Métodos:

dibujar(Entorno entorno): dibuja el rectángulo del menú.

Menu Inicio

Variables de instancia:

- x, y: Coordenadas del Menú.
- ancho, alto: Dimensiones del Menú.
- color: Color del Menú.
- botón: un botón para poder dar play al juego.

Métodos:

dibujar(Entorno entorno): dibuja el rectángulo del menú.

sePresionoIniciar(Entorno entorno): corrobora que se seleccione el botón para iniciar el juego.

Roca

La clase Roca son los obstáculos con los que el mago colisiona. Ésta clase, al ser las rocas un objeto fijo, es más corta y simple. El único problema que tuvimos fue asignarle una altura a la roca, la cual al final nunca usamos (porque entorno no lo pide) así que la descartamos.

Características y constructor de Roca:

Le asignamos un x e y para ubicarlo en el juego. Un diámetro para que sea un círculo. Y un color para distinguirlo del fondo.

Dibujar de Roca:

Dibuja un círculo con los parámetros anteriormente mencionados. Ésto gracias a que en entorno se encuentra el método dibujar círculo.

Musica

Variables de instancia:

Clip: Recibe el audio a reproducir

Métodos:

Sonido(String ubicacion): Método que recibe la ubicación del audio.

reproducir(): Método que reproduce el audio.

stop(): Método que corta la reproducción del sonido

Problemas y Soluciones

Problema: No podíamos subir los archivos de audio.

Solución: Logramos agregarlos en una carpeta de git llamada “resources” luego de investigar.

Pociones

Variables de instancia:

x, y: Coordenadas de la poción.

alto, diametro: Dimensiones de la poción.

color: Color de la poción.

Métodos:

dibujar(Entorno entorno): Dibuja la poción como un círculo.

Problemas y Soluciones

Problema: Como tuvimos poco tiempo, resultó difícil pensar en cómo desarrollar las pociones de manera más compleja

Solución: Desarrollamos una versión más simple, donde las pociones aparecen de forma aleatoria, así el código no se volvía más complejo.

IMPLEMENTACIÓN

Código Fuente:

Juego:

```
package juego;
import java.awt.Color;
import java.util.ArrayList;
import entorno.Entorno;
import entorno.InterfaceJuego;
public class Juego extends InterfaceJuego
{
    // El objeto Entorno que controla el tiempo y otros
    private Entorno entorno;
    private Menu menu;
    private Mago mago;
    private ArrayList<Roca> rocas;
    private ArrayList<Murcielago> murcielagos;
    private ArrayList<Boton> boton;
    private ArrayList<Hechizo> hechizos;
    private Boton botonVida;
    private Boton botonMana;
    private Boton hechizoBasico;
    private Boton hechizoSec;
    private Boton hechizoUlti;
    private Hechizo basico;
    private Hechizo secundario;
    private Hechizo ulti;
    private Sonido musica;
    private Menulnicio menulnicio;
    private boolean enMenulnicio;
    private int enemigosGenerados;
    private Menu menuWin;
```

```

private ArrayList <Potion> pociones;
private Sonido sonidoBasico;
private Sonido sonidoSecu;
private Sonido sonidoUlti;

// Variables y métodos propios de cada grupo
// ...

Juego()
{
    // Inicializa el objeto entorno
    this.entorno = new Entorno(this, "El Camino de Gondolf - Grupo 12", 1024, 600);
    this.entorno.setBackground(Color.white);
    this.menuInicio = new MenuInicio(entorno.ancho()/2, entorno.alto()/2, 400, 300);
    this.enMenuInicio = true;
    this.enemigosGenerados = 0;
    this.menuWin = new Menu(entorno.ancho(), entorno.alto(), 400, 300);

    double menuX = entorno.ancho()+entorno.ancho()/4;
    double menuY = 0.5*entorno.alto();
    menu = new Menu(menuX,menuY,entorno.ancho(),entorno.alto());

    // Iniciar lo que haga falta para el juego
    // ...
    mago = new Mago((entorno.ancho()-250)/2, entorno.alto()/2, 25, 40, Color.red, 100,
250,0);

    boton = new ArrayList<>();
    //      boton.add(new Boton (menuX - menuX/3,menuY*0.2, 120, 60, Color.darkGray,
    "basico"));//hechizo
    hechizoBasico =(new Boton (menuX - menuX/3,menuY*0.2, 120, 60, Color.yellow,
"basico",false));
    //      boton.add(new Boton(menuX - menuX/3,menuY*0.45, 120, 60,
    Color.darkGray,"hechizo"));//hechizo
    hechizoSec =(new Boton (menuX - menuX/3,menuY*0.45, 120, 60, Color.green,
"Bombinha",false));
    //      boton.add(new Boton(menuX - menuX/3,menuY*0.70, 120, 60, Color.darkGray,"ulti"
));//hechizo
    hechizoUlti=(new Boton (menuX - menuX/3,menuY*0.70, 120, 60, Color.cyan, "Freeze
Power",false));

    boton.add(new Boton(menuX - menuX/3.3,menuY*1.25, 180, 30, Color.darkGray,
"HP",false));//vida descontada
    botonVida = new Boton(menuX - menuX/3.3,menuY*1.25, 180, 30, Color.RED,
"HP",false);//vida
}

```

```

botonMana = new Boton(menuX - menuX/3.3, menuY*1.40,
180,30,Color.BLUE,"Mana",false);
boton.add(new Boton(menuX - menuX/3.3,menuY*1.40, 180, 30, Color.darkGray,
"Mana",false));//mana descontado
hechizos = new ArrayList<>();
basico= new Hechizo("Basico",entorno.mouseX(),entorno.mouseY(), 0, 20, 10,
Color.MAGENTA);
secundario=new Hechizo("Bombinha",entorno.mouseX(), entorno.mouseY(), 20, 40, 30,
Color.red);
ulti=new Hechizo("Freeze Power", entorno.mouseX(), entorno.mouseY(), 100, 169, 50,
Color.blue);

rocas = new ArrayList<>();
rocas.add(new Roca(entorno.ancho()/4,entorno.alto()/2,50,Color.gray));
rocas.add(new Roca(entorno.ancho()-entorno.ancho()/3, entorno.alto()/2 , 50,
Color.gray));
rocas.add(new Roca(entorno.ancho()/2, entorno.alto()*0.25, 50, Color.gray));
rocas.add(new Roca(entorno.ancho()/5, entorno.alto()*0.25, 50, Color.gray));
rocas.add(new Roca(entorno.ancho()/3, entorno.alto()*0.75, 50, Color.gray));
rocas.add(new Roca(entorno.ancho()/3 + entorno.ancho()/3, entorno.alto()*0.75, 50,
Color.gray));
murcielagos = new ArrayList<>();
pociones = new ArrayList<>();
musica = new Sonido("C:/Users/sebinor2/Desktop/El Mago
Gondolf/Gondolf/Gondolf/ProyectoLimpio/src/resource/musica-de-mago.wav"); // Ruta del archivo
de música

sonidoBasico = new
Sonido("C://Users/sebinor2/Desktop/magordito/ramirez-ozore-tp-p1/ProyectoLimpio/resources/hec
hizoBasico.wav");
sonidoSecu = new
Sonido("C://Users/sebinor2/Desktop/magordito/ramirez-ozore-tp-p1/ProyectoLimpio/resources/bo
mba.wav");
sonidoUlti = new
Sonido("C://Users/sebinor2/Desktop/magordito/ramirez-ozore-tp-p1/ProyectoLimpio/resources/free
zingSkill.wav");
// Inicia el juego!
this.entorno.iniciar();
}

/**
 * Durante el juego, el método tick() será ejecutado en cada instante y
 * por lo tanto es el método más importante de esta clase. Aquí se debe
 * actualizar el estado interno del juego para simular el paso del tiempo
 * (ver el enunciado del TP para mayor detalle).

```

```
 */
public void tick()
{
    // Procesamiento de un instante de tiempo
    // ...

    if (enMenuInicio) {

        dibujarMenuInicio();
        salirMenu();
    } else {

        this.dibujarObjetos();

        botonVida.dibujar(entorno);
        botonMana.dibujar(entorno);

        if(mago.estaVivo(mago)) {
            musica.reproducir();
            if(mago.getEnemigosAniquilados()<50) {

                if((entorno.estaPresionada(entorno.TECLA_DERECHA)|| entorno.estaPresionada('d'))&& !mago.colisionaPorDerecha(entorno) && mago.colisionConAlgunaRoca(rocas, 5 , 0) == false ) {
                    mago.moverDerecha(entorno);
                }
                if((entorno.estaPresionada(entorno.TECLA_IZQUIERDA)|| entorno.estaPresionada('a')) && !mago.colisionaPorIzquierda(entorno)&& mago.colisionConAlgunaRoca(rocas, -5 , 0) == false) {
                    mago.moverIzquierda(entorno);
                }
                if((entorno.estaPresionada(entorno.TECLA_ARRIBA)|| entorno.estaPresionada('w'))&& !mago.colisionaPorArriba(entorno)&& mago.colisionConAlgunaRoca(rocas, 0 , -5 ) == false) {
                    mago.moverArriba(entorno);
                }
                if((entorno.estaPresionada(entorno.TECLA_ABAJO )|| entorno.estaPresionada('s'))&& !mago.colisionaPorAbajo(entorno)&& mago.colisionConAlgunaRoca(rocas, 0 , 5 ) == false) {
                    mago.moverAbajo(entorno);
                }
            }

            mantenerMurcielagos();

            mago.perseguirMago(murcielagos, mago);
        }
    }
}
```

```
if(hechizoBasico.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), hechizoBasico)&& mago.getMana()>= basico.getCostoMana() ){
    basico.dibujar(entorno);
    hechizoBasico.setColor(Color.red);
    if(entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
        basico.lanzar( murcielagos, entorno, hechizoBasico, mago);
        gastarMana(basico.getCostoMana());
        hechizoBasico.setSeleccionado(false);
        if(!hechizoBasico.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), hechizoBasico)) {
            sonidoBasico.reproducir();
            hechizoBasico.setColor(Color.yellow);
            seGenero();
        }
    }
}

if(hechizoSec.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), hechizoSec)&& mago.getMana()>= secundario.getCostoMana() ){
    secundario.dibujar(entorno);
    hechizoSec.setColor(Color.red);
    if(entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
        secundario.lanzar( murcielagos, entorno, hechizoSec, mago);
        gastarMana(secundario.getCostoMana());
        hechizoSec.setSeleccionado(false);
        if(!hechizoSec.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), hechizoSec)) {
            sonidoSecu.reproducir();
            hechizoSec.setColor(Color.green);
            seGenero();
        }
    }
}

if(hechizoUlti.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), hechizoUlti)&& mago.getMana()>= ulti.getCostoMana() ){
    ulti.dibujar(entorno);
    hechizoUlti.setColor(Color.red);
    if(entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
        ulti.lanzarRelentizar(murcielagos, entorno, hechizoUlti, mago);
        gastarMana(ulti.getCostoMana());
    }
}
```

```
        hechizoUlti.setSeleccionado(false);
        if(!hechizoUlti.estaSeleccionado(entorno, entorno.mousePosition(),
entorno.mouseY(), hechizoUlti)) {
            hechizoUlti.setColor(Color.cyan);
            sonidoUlti.reproducir();

        }

    }

if (mago.colisionConAlgunMurcielago(murcielagos, 0, 3) || 
mago.colisionConAlgunMurcielago(murcielagos, 0, -3) || 
mago.colisionConAlgunMurcielago(murcielagos, 3, 0) || 
mago.colisionConAlgunMurcielago(murcielagos, 0, -3) || 
mago.colisionConAlgunMurcielago(murcielagos, -3, 0)) {
    recibirDanoMago(10);
}

if (mago.colisionConAlgunaPotion(pociones, 0, 3) || 
    mago.colisionConAlgunaPotion(pociones, 0, 3) || 
    mago.colisionConAlgunaPotion(pociones, 3, 0) || 
    mago.colisionConAlgunaPotion(pociones, 0, -3) || 
    mago.colisionConAlgunaPotion(pociones, -3, 0)) {
    restaurarVidaMago(20);
    sumarMana(10);

}

}

else {
    dibujarGameWin();
    musica.stop();
}

}

else {
    dibujarGameOver();
    musica.stop();
}

}

}

public void spawnMurcielago() {
```

```

int lado = (int)(Math.random() * 4);
double x = 0, y = 0;

if (lado == 0) { // arriba
    x = Math.random() * (entorno.ancho()-200);
    y = -20;
}
else if (lado == 2) { // abajo
    x = Math.random() * (entorno.ancho()-200);
    y = entorno.alto() + 20;
}
else if (lado == 1) { // der
    x = entorno.ancho() - 200;
    y = Math.random() * entorno.alto();
}
else { // izq
    x = -20;
    y = Math.random() * entorno.alto();
}
if (enemigosGenerados - mago.getEnemigosColisionados() < 50) {
    murcielagos.add(new Murcielago(x, y, 20, 20, Color.YELLOW, true, 5, 1.3));
    enemigosGenerados++;
}

public void mantenerMurcielagos() {
    int vivos = 0;
    for (Murcielago m : murcielagos) {
        if (m != null && m.estaVivo(m)) {
            vivos++;
        }
    }
    // Si hay menos de 20 murciélagos vivos, genera los necesarios
    while (vivos < 10) {
        spawnMurcielago();
        vivos++;
    }
}

public void recibirDanoMago(int cantidad) {
    mago.recibirDano(cantidad); // Reduce la vida del mago
    double porcentajeVida = Math.max(0, Math.min(1, mago.getHp() /
mago.getHpMaxima())));
}

```

```
botonVida.setAncho(180 * porcentajeVida); // Ajusta el ancho del botón  
proporcionalmente  
}  
public void restaurarVidaMago(int cantidad) {  
    mago.restaurarVida(cantidad); // Reduce la vida del mago  
    double porcentajeVida = Math.max(0, Math.min(1, mago.getHp() /  
mago.getHpMaxima()));  
  
    botonVida.setAncho(180 * porcentajeVida); // Ajusta el ancho del botón  
proporcionalmente  
}  
  
public void gastarMana(int cantidad) {  
    double porcentajeMana = Math.max(0, Math.min(1, mago.getMana() /  
mago.getManaMaxima()));  
  
    botonMana.setAncho(180 * porcentajeMana);  
}  
public void sumarMana(int cantidad) {  
    mago.restaurarMana(cantidad);  
    double porcentajeMana = Math.max(0, Math.min(1, mago.getMana() /  
mago.getManaMaxima()));  
  
    botonMana.setAncho(180 * porcentajeMana);  
}  
  
public void generarPoti(ArrayList<Potion>pociones) {  
    double x = Math.random() * (entorno.ancho() - 40) + 20; // Evitar bordes  
    double y = Math.random() * (entorno.alto() - 40) + 20; // Evitar bordes  
    String tipo = (Math.random() < 0.5) ? "vida" : "mana";  
    pociones.add(new Potion(x,y, 20, Color.CYAN));  
}  
  
public void seGenero() {  
    if (Math.random() < 0.3) {  
        generarPoti(pociones);  
    }  
}
```

```

public void dibujarObjetos() {

    if(this.mago != null) {
        this.mago.dibujar(entorno);
    }
    if (rocas != null) {
        for(Roca r : rocas) {
            r.dibujar(entorno);
        }
    }
    if (murcielagos != null) {
        for(Murcielago m : murcielagos) {
            if (m != null) {
                m.dibujar(entorno);
            }
        }
    }
    for (Potion p : pociones) {
        if (p != null) {
            p.dibujar(entorno);
        }
    }
    entorno.cambiarFont("Arial", 15, Color.WHITE);
    entorno.escribirTexto("Enemigos ANIQUILADOS: " + mago.getEnemigosAniquilados()
,20,20);
    entorno.escribirTexto("Enemigos COLISIONADOS: " +
mago.getEnemigosColisionados() ,20,40);
    menu.dibujar(entorno);

    if(boton !=null) {
        hechizoBasico.dibujar(entorno);
        hechizoSec.dibujar(entorno);
        hechizoUlti.dibujar(entorno);
        for(Boton b : boton) {
            b.dibujar(entorno);
        }
    }
}

private void dibujarMenuInicio() {
    entorno.setBackground(Color.BLACK);
    menuInicio.dibujar(entorno);
    entorno.cambiarFont("Times New Roman", 22, Color.white);
    entorno.escribirTexto("El Mago Gondolf", entorno.ancho()*0.43, entorno.alto()/3);
}

```

```
entorno.escribirTexto("Basico", entorno.ancho()*0.07, 300);
entorno.escribirTexto("Habilidad basica que mata murcielagos", entorno.ancho()*0.01, 325);
entorno.escribirTexto("Costo Mana: 0", entorno.ancho()*0.01, 350);
entorno.escribirTexto("Bombinha", entorno.ancho()*0.65, 300);
entorno.escribirTexto("Habilidad que mata murcielagos con mayor rango que Basico",
entorno.ancho()*0.44, 325);
entorno.escribirTexto("Costo Mana: 20", entorno.ancho()*0.5, 350);
entorno.escribirTexto("Freeze Power", entorno.ancho()*0.07, 400);
entorno.escribirTexto("Habilidad especial que ralentiza a los murcielagos",
entorno.ancho()*0.01, 425);
entorno.escribirTexto("(no acumulable)", entorno.ancho()*0.01, 450);
entorno.escribirTexto("Costo Mana: 100", entorno.ancho()*0.01, 475);
entorno.escribirTexto("Las pociones regeneran tanto vida como maná", entorno.ancho()*0.5,
475);
}
public void salirMenu() {
    if (menuInicio.sePresionoIniciar(entorno)) {
        enMenuInicio = false;
    }
}

private void dibujarGameOver() {
    entorno.cambiarFont("Times New Roman", 40, Color.RED);
    entorno.escribirTexto("GAME OVER", 350, 300);
}

private void dibujarGameWin() {
    menuWin.dibujar(entorno);
    entorno.cambiarFont("Times New Roman", 40, Color.RED);
    entorno.escribirTexto("YOU WIN", 350, 300);
}

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

Mago:

```
package juego;

import java.awt.Color;
import java.util.ArrayList;
import entorno.Entorno;

public class Mago {

    private double x,y;
    private double ancho,alto;
    private Color color;
    private double hp;
    private double mana;
    private double hpMaxima;
    private double manaMaxima;
    private int enemigosAniquilados;
    private int enemigosColisionados;

    public Mago(double x,double y,double ancho,double alto, Color color, double hp, double mana, int enemigosAniquilados) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.color = color;
        this.hp = hp;
        this.mana = mana;
        this.hpMaxima = hp;
        this.manaMaxima = mana;
        this.enemigosAniquilados = enemigosAniquilados;
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, color);
    }
}
```

```
public void dibujar(Entorno entorno) {
    entorno.dibujarRectangulo(x, y, ancho, alto, 0, color);
}

public void moverDerecha(Entorno entorno) {
    this.x +=3;
}

public void moverIzquierda(Entorno entorno) {
    this.x -=3;
}

public void moverArriba(Entorno entorno) {
    this.y -=3;
}

public void moverAbajo(Entorno entorno) {
    this.y +=3;
}

public boolean colisionaPorDerecha(Entorno entorno) {
    return this.x + (this.ancho)/2 >= entorno.ancho() -entorno.ancho()/4;
}

public boolean colisionaPorIzquierda(Entorno entorno) {
    return this.x - this.ancho/2 <= 0;
}

public boolean colisionaPorArriba(Entorno entorno) {
    return this.y - this.alto/2 <= 0;
}

public boolean colisionaPorAbajo(Entorno entorno) {
    return this.y + this.alto/2 >= entorno.alto();
}

public boolean estaVivo(Mago mago) {
```

```

public boolean estaVivo(Mago mago) {
    if (this.hp <= 0) {
        mago = null;
    }
    if (mago == null) {
        return false;
    }
    return true;
}

public boolean colisionConRoca(Roca r, double xr, double yr) {
    if (r == null) {
        return false;
    }
    double ladoIzquierdo = this.x - (this.ancho/2);
    double ladoDerecho = this.x + (this.ancho/2);
    double ladoSuperior = this.y - (this.alto/2);
    double ladoInferior = this.y + (this.alto/2);

    double xCercano = Math.max(ladoIzquierdo, Math.min(r.getX(), ladoDerecho));
    double yCercano = Math.max(ladoSuperior, Math.min(r.getY(), ladoInferior));

    double alto= yCercano - r.getY();
    double ancho= xCercano - r.getX();
    double distancia = (int) Math.sqrt( Math.pow(alto, 2) + Math.pow(ancho, 2));

    boolean colisiona = distancia <= (r.getDiametro() / 2);

    if (colisiona) {
        if (xr > 0 && this.x < r.getX()) return true;
        if (xr < 0 && this.x > r.getX()) return true;
        if (yr > 0 && this.y < r.getY()) return true;
        if (yr < 0 && this.y > r.getY()) return true;
    }
    return false;
}

public boolean colisionConAlgunaRoca(ArrayList<Roca> rocas, double dx, double dy) {
    if (rocas == null || rocas.isEmpty()) return false;

    for (Roca r : rocas) {
        if (colisionConRoca(r, dx, dy)) {
            return true;
        }
    }
    return false;
}

```

```

public boolean colisionConMurcielago(Murcielago m, double xr, double yr) {
    if(m == null) {
        return false;
    }
    double ladoIzquierdo = this.x - (this.ancho/2);
    double ladoDerecho = this.x + (this.ancho/2);
    double ladoSuperior = this.y - (this.alto/2);
    double ladoInferior = this.y + (this.alto/2);

    double xCercano = Math.max(ladoIzquierdo, Math.min(m.getX(), ladoDerecho));
    double yCercano = Math.max(ladoSuperior, Math.min(m.getY(), ladoInferior));

    double alto= yCercano - m.getY();
    double ancho= xCercano - m.getX();
    double distancia = (int) Math.sqrt( Math.pow(alto, 2) + Math.pow(ancho, 2));

    boolean colisiona = distancia <= (m.getDiametro() / 2);

    if (colisiona) {
        if (xr > 0 && this.x < m.getX()) return true;
        if (xr < 0 && this.x > m.getX()) return true;
        if (yr > 0 && this.y < m.getY()) return true;
        if (yr < 0 && this.y > m.getY()) return true;
    }
    return false;
}

public boolean colisionConAlgunMurcielago(ArrayList<Murcielago> murcielagos, double xm, double ym) {
    if (murcielagos != null) {
        for (int i = 0; i < murcielagos.size(); i++) {
            Murcielago m = murcielagos.get(i);
            if (m != null && colisionConMurcielago(m, xm, ym)) {
                murcielagos.set(i, null); // Asignar null al murciélagos que colisionó
                this.setEnemigosColisionados(getEnemigosColisionados() +1);
                return true; // Se detectó colisión
            }
        }
    }
    return false; // No hubo colisión
}

public void recibirDano(int cantidad) {
    this.hp -= cantidad;
    if (this.hp < 0) {
        this.hp = 0; // No permite valores negativos
    }
}

public void gastarMana(int cantidad) {
    this.mana-=cantidad;
}

public void perseguirMago(ArrayList<Murcielago> murcielagos, Mago mago) {
    if (murcielagos != null) {
        for (Murcielago m : murcielagos) {
            if (m != null && m.estaVivo(m)) {
                m.movimiento(m.getSpeed(), mago); // Velocidad ajustada
            }
        }
    }
}

```

```
public boolean colisionConPotion(Potion p, double xp, double yp) {  
    if(p == null) {  
        return false;  
    }  
    double ladoIzquierdo = this.x - (this.ancho/2);  
    double ladoDerecho = this.x + (this.ancho/2);  
    double ladoSuperior = this.y - (this.alto/2);  
    double ladoInferior = this.y + (this.alto/2);  
  
    double xCercano = Math.max(ladoIzquierdo, Math.min(p.getX(), ladoDerecho));  
    double yCercano = Math.max(ladoSuperior, Math.min(p.getY(), ladoInferior));  
  
    double alto= yCercano - p.getY();  
    double ancho= xCercano - p.getX();  
    double distancia = (int) Math.sqrt( Math.pow(alto, 2) + Math.pow(ancho, 2));  
  
    boolean colisiona = distancia <= (p.getDiametro() / 2);  
  
    if (colisiona) {  
        if (xp > 0 && this.x < p.getX()) return true;  
        if (xp < 0 && this.x > p.getX()) return true;  
        if (yp > 0 && this.y < p.getY()) return true;  
        if (yp < 0 && this.y > p.getY()) return true;  
    }  
  
    return false;  
}
```

```
public boolean colisionConAlgunaPotion(ArrayList<Potion> pociones, double xp, double yp) {  
    if (pociones != null) {  
        for (int i = 0; i < pociones.size(); i++) {  
            Potion p = pociones.get(i);  
            if (p != null && colisionConPotion(p, xp, yp)) {  
                pociones.set(i, null);  
  
                return true; // Se detectó colisión  
            }  
        }  
        return false; // No hubo colisión  
    }  
  
    public void restaurarVida(int cantidadARestaurar) {  
        this.hp += cantidadARestaurar;  
        if (this.hp > 100) {  
            this.hp = 100; // No permite valores mayores al maximo  
        }  
    }  
    public void restaurarManá(int cantidadARestaurar) {  
        this.mana += cantidadARestaurar;  
        if (this.mana > 250) {  
            this.mana = 250; // No permite valores mayores al maximo  
        }  
    }  
  
    public double getX() {  
        return x;  
    }  
  
    public void setX(double x) {  
        this.x = x;  
    }  
}
```

//demás getters y setters de mago.

Murcielago:

```
package juego;

import java.awt.Color;

import entorno.Entorno;

public class Murcielago {

    private double x,y;
    private double alto;
    private double diametro;
    private Color color;
    private boolean vivo;
    private int hp;
    private double speed;

    public Murcielago(double x, double y, double alto, double diametro, Color color, boolean vivo, int hp, double speed) {
        this.x = x;
        this.y = y;
        this.alto = alto;
        this.diametro = diametro;
        this.color = color;
        this.vivo = vivo;
        this.hp = hp;
        this.speed = speed;
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarCirculo(x, y, diametro, color);
    }

    public void moverDerecha(double posicion, Mago mago) {
        if(this.x < mago.getX()) {
            this.x += posicion;
        }
    }

    public void moverIzquierda(double posicion,Mago mago) {
        if(this.x > mago.getX()) {
            this.x -= posicion;
        }
    }

    public void moverArriba(double posicion,Mago mago) {
        if(this.y > mago.getY()) {
            this.y -= posicion;
        }
    }

    public void moverAbajo(double posicion,Mago mago) {
        if(this.y < mago.getY()) {
            this.y += posicion;
        }
    }
}
```

```
public void movimiento(double posicion, Mago mago) {  
    // x  
    if (this.x < mago.getX()) {  
        moverDerecha(posicion, mago);  
    } else if (this.x > mago.getX()) {  
        moverIzquierda(posicion, mago);  
    }  
  
    // y  
    if (this.y < mago.getY()) {  
        moverAbajo(posicion, mago);  
    } else if (this.y > mago.getY()) {  
        moverArriba(posicion, mago);  
    }  
}  
  
public void recibirDano(int cantidad) {  
    hp -= cantidad;  
    if (hp <= 0) {  
        vivo = false;  
    }  
}  
  
public boolean estaVivo(Murcielago murcielagos) {  
    if (this.hp <= 0) {  
        murcielagos = null;  
    }  
    if (murcielagos == null) {  
        return false;  
    }  
}  
}  
return true;  
}
```

```
public double getX() {  
    return x;  
}  
public void setX(double x) {  
    this.x = x;  
}  
public double getY() {  
    return y;  
}  
public void setY(double y) {  
    this.y = y;  
}  
public double getAlto() {  
    return alto;  
}  
public void setAlto(double alto) {  
    this.alto = alto;  
}  
public double getDiametro() {  
    return diametro;  
}  
public void setDiametro(double diametro) {  
    this.diametro = diametro;  
}  
public Color getColor() {  
    return color;  
}  
public void setColor(Color color) {  
    this.color = color;  
}
```

//demas getters y

setters

Hechizo:

```
package juego;

import java.awt.Color;

public class Hechizo {
    private String nombre;
    private double x,y;
    private int costoMana;
    private double radioEfecto;
    private int danio;
    private Color color;

    public Hechizo(String nombre, double x, double y , int costoMana, double radioEfecto, int danio,
                   Color color) {
        this.nombre = nombre;
        this.x = x;
        this.y = y;
        this.costoMana = costoMana;
        this.radioEfecto = radioEfecto;
        this.danio = danio;
        this.color = color;
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarCirculo(entorno.mouseX(), entorno.mouseY(), radioEfecto*2, color);
    }

    public boolean puedeLanzarse(Entorno entorno, double anchoMenu, Boton boton) {
        return boton.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), boton) &&
               entorno.mouseX() <= (entorno.ancho() - anchoMenu);
    }
}
```

```
public void lanzar(ArrayList<Murcielago> murcielagos, Entorno entorno, Boton boton,Mago mago) {  
    if (puedeLanzarse(entorno, entorno.ancho()/4, boton)) {  
        hacerDanio(murcielagos, entorno.mousePosition(), entorno.mousePosition(), mago);  
        mago.gastarMana(costoMana);  
  
        boton.setSeleccionado(false);  
    }  
}  
  
  
public void hacerDanio(ArrayList<Murcielago> murcielagos, double x, double y, Mago mago) {  
    for (int i = murcielagos.size() - 1; i >= 0; i--) {  
        Murcielago m = murcielagos.get(i);  
        if (m != null && m.estaVivo(m) && colisionConMurcielago(m, x, y)) {  
            m.recibirDano(this.danio);  
            if (!m.estaVivo(m)) {  
                mago.setEnemigosAniquilados(mago.getEnemigosAniquilados()+1);  
                murcielagos.set(i,null);  
            }  
        }  
    }  
}  
  
public void relentizar(ArrayList<Murcielago> murcielagos, double x, double y, Mago mago) {  
    for (int i = murcielagos.size() - 1; i >= 0; i--) {  
        Murcielago m = murcielagos.get(i);  
        if (m != null && m.estaVivo(m) && colisionConMurcielago(m, x, y)) {  
            m.setSpeed(0.7);  
        }  
    }  
}
```

```
public void lanzarRelentizar(ArrayList<Murcielago> murcielagos, Entorno entorno, Boton boton,Mago mago) {  
    if (puedeLanzarse(entorno, entorno.ancho()/4, boton)) {  
        mago.gastarMana(costoMana);  
        relentizar(murcielagos, entorno.mousePosition(), entorno.mousePosition(), mago);  
        boton.setSeleccionado(false);  
    }  
}  
  
public boolean colisionConMurcielago(Murcielago m, double xHechizo, double yHechizo) {  
    if (m == null || !m.estaVivo(m)) {  
        return false;  
    }  
  
    double distanciaX = xHechizo - m.getX();  
    double distanciaY = yHechizo - m.getY();  
    double distancia = Math.sqrt(distanciaX * distanciaX + distanciaY * distanciaY);  
  
    return distancia <= (this.radioEfecto + m.getDiametro() / 2);  
}  
  
// Getters y Setters  
public String getNombre() {  
    return nombre;  
}
```

//demas getters y setters

Roca:

```
package juego;
import java.awt.Color;

public class Roca {
    private double x,y;
    private double diametro;
    private Color color;

    public Roca(double x, double y, double diametro, Color color) {
        this.x = x;
        this.y = y;
        this.diametro = diametro;
        this.color = color;
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarCirculo(x, y, diametro, color);
    }

    public double getX() {
        return x;
    }

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

    public double getY() {
        return y;
    }

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

    public Color getColor() {
        //demas getters y setters
    }
}
```

Boton:

```
1 package juego;
2
3 import java.awt.Color;
4
5 public class Potion {
6     private double x, y;
7     private double diametro;
8     private Color color;
9
10    public Potion(double x, double y, double diametro, Color color) {
11        this.x = x;
12        this.y = y;
13        this.diametro = diametro;
14        this.color = color;
15    }
16
17
18    public void dibujar(Entorno entorno) {
19        entorno.dibujarCirculo(x, y, diametro, color);
20    }
21
22    public double getX() {
23        return x;
24    }
25
26    public void setX(double x) {
27        this.x = x;
28    }
29
30    public double getY() {
31        return y;
32    }
33
34    public void setY(double y) {
35        this.y = y;
36    }
37}
```

//demas getters y setters

Potion:

```
package juego;

import java.awt.Color;

public class Boton {
    private double x,y;
    private double ancho,alto;
    private Color color;
    private String str;
    private boolean seleccionado;

    public Boton(double x, double y, double ancho, double alto, Color color, String str, boolean seleccionado) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.color = color;
        this.str = str;
        this.seleccionado = seleccionado;
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, color);

        if(this.str != null) {
            entorno.cambiarFont("Sans-Serif", 16, Color.WHITE);
            entorno.escribirTexto(str, x - str.length() * 4, y + 5);
        }
    }
}

//demas getters y setters
```

Menu:

```

package juego;

import java.awt.Color;

public class Menu {
    private double x, y;
    private double ancho, alto;
    private Color colorFondo;

    public Menu(double x, double y, int ancho, int alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.colorFondo = new Color(100, 100, 60);
    }
    public void dibujar(Entorno entorno) {
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, colorFondo);
    }
}

```

//demas getters y setters

Menu Inicio:

```

package juego;

import java.awt.Color;

public class MenuInicio {
    private double x;
    private double y;
    private double ancho;
    private double alto;
    private Boton botonIniciar;

    public MenuInicio(double x, double y, double ancho, double alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.botonIniciar = new Boton(x, y - 50, 200, 60, Color.GREEN, "Iniciar Juego", false);
    }

    public void dibujar(Entorno entorno) {
        entorno.cambiarFont("Times New Roman", 40, Color.BLACK);
        entorno.escribirTexto("El Camino de Gondolf", 10,10);
        botonIniciar.dibujar(entorno);
    }

    public boolean sePresionoIniciar(Entorno entorno) {
        return botonIniciar.estaSeleccionado(entorno, entorno.mouseX(), entorno.mouseY(), botonIniciar) &&
               entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO);
    }
}

```

Musica

```
package juego;

import javax.sound.sampled.*;

public class Musica {
    private Clip clip;

    public Musica(String filePath) {
        try {
            File audioFile = new File(filePath);
            AudioInputStream audioStream = AudioSystem.getAudioInputStream(audioFile);
            clip = AudioSystem.getClip();
            clip.open(audioStream);
        } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {
            e.printStackTrace();
        }
    }

    public void play() {
        if (clip != null) {
            clip.start();
            clip.loop(Clip.LOOP_CONTINUOUSLY); // Repetir música en bucle
        }
    }

    public void stop() {
        if (clip != null && clip.isRunning()) {
            clip.stop();
        }
    }

    public void restart() {
        if (clip != null) {
            clip.setFramePosition(0); // Reinicia desde el inicio
            clip.start();
        }
    }
}
```

Sonido:

```
1 package juego;
2
3+import java.io.File;[]
4
5 public class Sonido {
6     private Clip clip;
7
8
9     public Sonido(String ubicacion){
10         try {
11             File efectoSonido = new File(ubicacion);
12             AudioInputStream sonidito = AudioSystem.getAudioInputStream(efectoSonido);
13             clip = AudioSystem.getClip();
14             clip.open(sonidito);
15         } catch (Exception e) {
16             e.printStackTrace();
17             // TODO: handle exception
18         }
19     }
20
21     public void reproducir() {
22         if(clip != null) {
23             clip.setFramePosition(0);
24             clip.start();
25         }
26     }
27
28     public void stop() {
29         if (clip != null && clip.isRunning()) {
30             clip.stop();
31         }
32     }
33
34+    public Clip getClip() {
35         return clip;
36     }
37
38+    public void setClip(Clip clip) {
39         this.clip = clip;
40     }
41
42
43 }
```

CONCLUSIONES:

Fue un proyecto muy divertido de hacer, a veces la mejor forma de aprender cosas nuevas es adentrándose de lleno en eso, investigar, probar y fallas pero aprender a resolverlo. Creemos que lo más valioso de éste trabajo es aprender a trabajar en equipo, saber administrar los tiempos de cada miembro y poder distribuir las actividades de manera equitativa. La organización es crucial. Pero lo más importante es tener la voluntad de sentarse y estar dispuesto a equivocarse pero arreglar y aprender arreglando los errores, no hay mejor manera de aprender que metiendo la pata, rompiendo el código (como nos pasó más de una vez) y solucionando este percance (una satisfacción que pocas cosas te pueden dar).

Encontramos la forma de personalizar el juego divirtiéndonos en el proceso y logrando investigar para llegar más lejos de lo que hubierámos logrado al principio. Si bien al comienzo de cada punto se podía llegar a volver frustrante que algo no salga, era ese mismo sentimiento el que nos motivaba a terminar el trabajo y buscar esa sensación de satisfacción cuando algo comienza a andar.