

Universidad Nacional de General Sarmiento



“EL CAMINO DE GONDOLF”

PROFESORES: Rodrigo Gonzalez - Sabrina Castro - Ezequiel Cañete

MATERIA: Programación I

CICLO LECTIVO: 1er semestre de 2025

ALUMNOS:

- Gimena Ibars – ibarsgisel075@gmail.com
- Cristian Rodriguez – crisrodriguez.dev@gmail.com
- Federico Bethge – fedebethge202014@gmail.com

INTRODUCCION

El presente informe detalla el desarrollo del videojuego **“El Camino de Gondolf”**, un trabajo práctico grupal realizado en el marco de la asignatura de Programación Orientada a Objetos. El objetivo principal del proyecto fue aplicar los conceptos fundamentales de la programación en Java, integrando componentes gráficos, lógica de juego en tiempo real y diseño modular.

“El Camino de Gondolf” es un juego tipo arcade/supervivencia en 2D, en el cual el jugador asume el rol de un mago que debe enfrentar diversos enemigos a lo largo de distintos niveles, utilizando hechizos mágicos y pociones de ayuda. A través de una mecánica progresiva, el usuario se ve desafiado a combinar estrategia, reflejos y gestión de recursos para alcanzar la meta final: derrotar al jefe supremo.

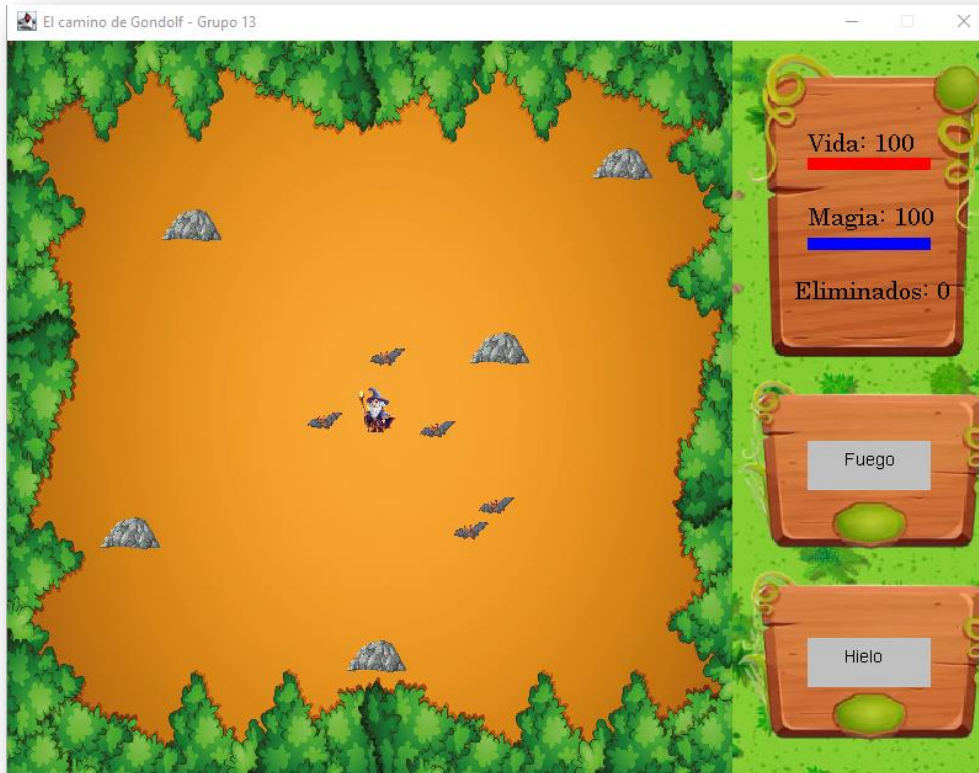
DESCRIPCIÓN

“El Camino de Gondolf” se sitúa en un entorno de fantasía. El protagonista, un mago llamado Gondolf, debe defenderse del ataque constante de criaturas enemigas mediante el uso de hechizos mágicos y pociones especiales.

El objetivo del jugador es superar todos los niveles, eliminando a los enemigos que aparecen y evitando ser derrotado. El desafío final consiste en enfrentarse a un poderoso jefe en el nivel cuatro.

Acciones disponibles para el jugador:

- Moverse libremente por el escenario.
- Seleccionar y lanzar hechizos (de fuego o de hielo).
- Recoger pociones para recuperar vida o generar escudos protectores.
- Atacar cuerpo a cuerpo haciendo clic sobre los enemigos.
- Visualizar en pantalla información clave como vida, magia, enemigos eliminados y nivel.



Producción propia - Imagen sacada del nivel 1

IMPLEMENTACIÓN

- **Lenguaje y herramientas utilizadas**

El videojuego fue desarrollado en Java, un lenguaje orientado a objetos, altamente portable, robusto y ampliamente utilizado en el desarrollo de aplicaciones gráficas. Se utilizó la biblioteca gráfica **Entorno**, pensada para facilitar la creación de interfaces visuales y juegos simples en 2D.

El enfoque orientado a objetos permitió organizar el código en clases bien definidas, facilitando el mantenimiento y la escalabilidad del proyecto.

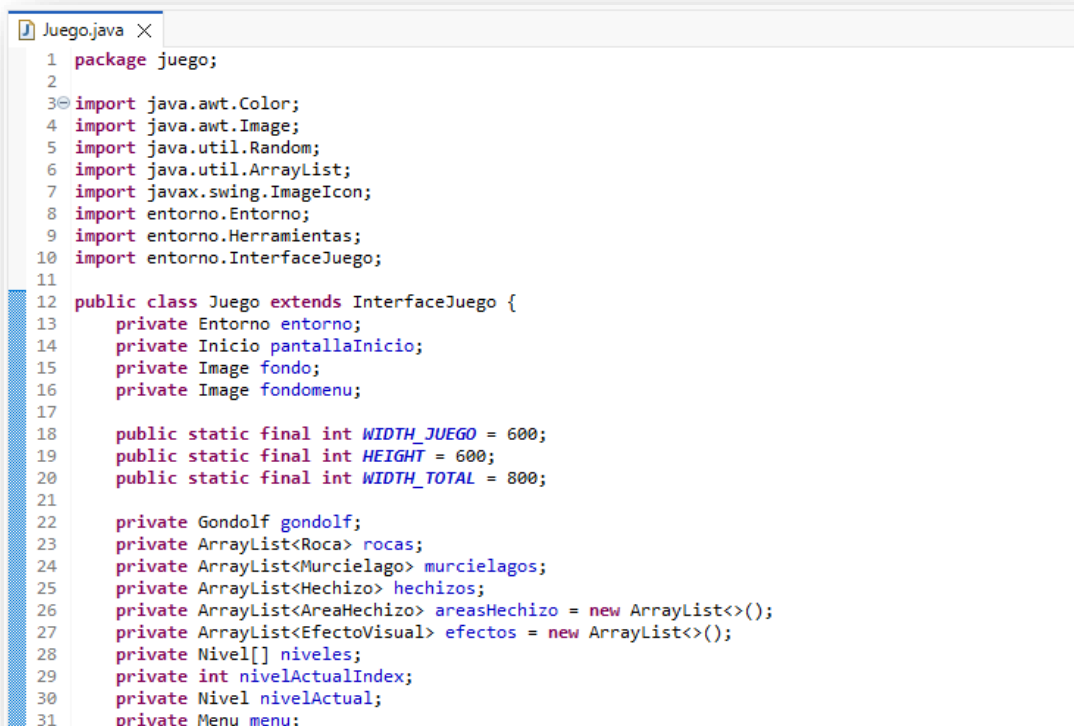
Clases principales del proyecto:

- **JUEGO**: clase principal. Controla el ciclo del juego (tick), maneja eventos y organiza la interacción entre los elementos.
- **GONDOLF**: representa al personaje del jugador. Contiene atributos como vida, magia, posición y métodos de ataque.

- **MURCIELAGO / JEFEFINAL**: representan enemigos con comportamientos y ataques propios.
- **HECHIZO**: instancia mágica lanzada por el jugador (de fuego o de hielo).
- **POCION**: otorga efectos temporales como recuperación de vida o escudo protector.
- **NIVEL**: ajusta la dificultad, velocidad y cantidad de enemigos.
- **ROCA**: obstáculos estáticos con detección de colisión.
- **EFFECTOVISUAL**: maneja animaciones temporales de hechizos.
- **BOMBA / PROYECTIL**: implementan los ataques del jefe final.
- **MENU**: interfaz gráfica con indicadores de vida, magia, enemigos eliminados y selección de hechizo.
- **INICIO**: presenta una introducción narrativa y las instrucciones del juego.

JUEGO:

- La clase Juego es el corazón de la aplicación, organizando la interacción entre todos los elementos del juego.



```

1 package juego;
2
3 import java.awt.Color;
4 import java.awt.Image;
5 import java.util.Random;
6 import java.util.ArrayList;
7 import javax.swing.ImageIcon;
8 import entorno.Entorno;
9 import entorno.Herramientas;
10 import entorno.InterfaceJuego;
11
12 public class Juego extends InterfaceJuego {
13     private Entorno entorno;
14     private Inicio pantallaInicio;
15     private Image fondo;
16     private Image fondomenu;
17
18     public static final int WIDTH_JUEGO = 600;
19     public static final int HEIGHT = 600;
20     public static final int WIDTH_TOTAL = 800;
21
22     private Gondolf gondolf;
23     private ArrayList<Roca> rocas;
24     private ArrayList<Murcielago> murcielagos;
25     private ArrayList<Hechizo> hechizos;
26     private ArrayList<AreaHechizo> areasHechizo = new ArrayList<>();
27     private ArrayList<EfectoVisual> efectos = new ArrayList<>();
28     private Nivel[] niveles;
29     private int nivelActualIndex;
30     private Nivel nivelActual;
31     private Menu menu;

```

Producción propia – Primeras 31 líneas del código Juego.java

- Funcionalidades principales:
 - **Inicialización:** El constructor de Juego configura el entorno, carga las imágenes de fondo, inicializa al personaje Gondolf, las Rocas, y los Niveles.
 - **Gestión de tick():** El método tick() es el bucle principal del juego, encargado de actualizar el estado de todos los elementos (movimientos, colisiones, lógica de juego).
 - **Dibujado:** La clase Juego coordina el dibujado de todos los elementos en el entorno gráfico en cada tick.
 - **Control del jugador:** Responde a la entrada del teclado para mover a Gondolf y lanzar hechizos.
 - **Gestión de niveles:** Controla la progresión entre niveles, la generación de enemigos y los objetivos de cada nivel.
 - **Colisiones:** Implementa la lógica para detectar y resolver colisiones entre Gondolf, Rocas, Murcielagos y Hechizos.
 - **Estado del juego:** Maneja los estados de inicio, transición de nivel, juego en curso, victoria y derrota.

GONDOLF:

- Vida inicial: 100 puntos.
- Magia inicial: 100 puntos.
- Dos hechizos disponibles: fuego y hielo.
- Dos pociones: de vida (recupera salud) y de protección (genera escudo).
- Puede eliminar enemigos con clic directo o mediante hechizos.
- Se desplaza libremente en una ventana de 600x600 píxeles.
- Colisiona con enemigos y obstáculos.

```

1 package juego;
2
3 import java.awt.Color;
4 import java.awt.Image;
5 import java.util.ArrayList;
6 import javax.swing.ImageIcon;
7 import entorno.Entorno;
8
9 public class Gondolf {
10     private double x, y;
11     private int vida;
12     private int magia;
13     private long ultimoTiempoRecargaMagia;
14     private final int ancho = 40;
15     private final int alto = 40;
16     private Image imagen;
17
18     //PARA AGREGAR LAS POCIONES DE PROTECCION Y VIDA
19     private boolean protegido;
20     private long tiempoProteccionInicio;
21     private static final long DURACION_PROTECCION = 5000; // TIEMPO DE HALO DE PROTECCION, 5 SEGUNDOS
22     private static final int RADIO_PROTECCION = 120;
23
24
25     public Gondolf(double x, double y) {}
26
27
28     // MOVIMIENTO DE GONDOLF EVITANDO CAMINAR SOBRE LAS ROCAS
29     public void mover(double dx, double dy, ArrayList<Roca> rocas) {}
30
31
32     public void dibujar(Entorno entorno) {}
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

Producción propia – Primeras 100 líneas del código Gondolf.java

▪ Funcionalidades principales:

- Movimiento: Métodos para mover a Gondolf en el entorno, considerando las colisiones con las Rocas.
- Vida y Magia: Métodos para gestionar la vida (perderVida, estaVivo) y la magia (getMagia, puedeUsarHechizo, consumirMagia, regenerarMagia) de Gondolf.
- Dibujado: Método para dibujar la imagen de Gondolf en el entorno.

HECHIZOS:

<i>Tipo de hechizo</i>	<i>Efecto principal</i>	<i>Costo de magia</i>	<i>Rango de acción</i>	<i>Efecto adicional</i>
Fuego	Mata instantáneamente a los enemigos en rango	30 puntos	120 píxeles	Daño total
Hielo	Congela enemigos durante 3 segundos	20 puntos	120 píxeles	Permite matarlos congelados

```

Hechizo.java X
1 package juego;
2
3 import java.awt.Color;
4
5
6 public class Hechizo {
7     private String nombre;
8     private int area;
9     private int costoMagia;
10    private int daño;
11    private Color color;
12
13    public Hechizo(String nombre, int area, int costoMagia, int daño, Color color) {}
14
15
16    public void lanzar(int x, int y, ArrayList<Murcielago> murcielagos, Gondolf gondolf) {}
17
18
19    public String getNombre() { return nombre; }
20    public int getArea() { return area; }
21    public int getCostoMagia() { return costoMagia; }
22    public int getDaño() { return daño; }
23    public Color getColor() { return color; }
24
25 }
26
27

```

Producción propia – código Hechizo.java

- Funcionalidades principales:

- Lanzar: Método lanzar que aplica el efecto del hechizo (congelar o quemar) a los Murciélagos dentro de su área de efecto. También marca a los murciélagos como eliminados por el jugador si es un hechizo ofensivo.
- Getters: Métodos para obtener los atributos del hechizo (nombre, área, costo de magia).
-

POCIONES:

- **Protección:**

- Nivel 1 a 3: aparece cada 8 enemigos eliminados.
- Nivel 4: aparece cada 8 segundos.
- Efecto: genera un halo de protección de 3 segundos.

- **Vida:**

- Nivel 1 a 3: aparece cada 15 enemigos eliminados.

- Nivel 4: aparece cada 15 segundos.
- Efecto: restaura la vida de Gondolf al 100%.

```

Pocion.java X
1 package juego;
2
3+ import entorno.Entorno;
4
5
6
7 public class Pocion {
8     public enum Tipo { VIDA, PROTECCION }
9
10    private double x, y;
11    private Tipo tipo;
12    private boolean activa;
13    private Image imagen;
14
15    // FUNCION PARA UBICACION DE LA POCION Y NOMBRE
16+ public Pocion(double x, double y, String string) {
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31+ public void dibujar(Entorno entorno) {
32
33
34
35+ public boolean estaActiva() {
36
37
38
39+ public void desactivar() {
40
41
42
43+ public boolean colisionaCon(Gondolf g) {
44
45
46
47+ public Tipo getTipo() {
48
49
50
51    public double getX() { return x; }
52    public double getY() { return y; }
53 }
54

```

Producción propia – código Pocion.java

- Funcionalidades principales:
 - Vida: Regenera la vida perdida del personaje a 100 puntos.
 - Protección: Genera una barrera de protección en un rango y tiempo determinado.

MURCIELAGOS:

- Aparecen aleatoriamente en el escenario.
- Se dirigen hacia Gondolf.
- Cada contacto le quita 10 puntos de vida.
- Solo pueden ser eliminados por clic o mediante hechizos.

```
Murcielago.java X
1 package juego;
2
3 import java.awt.Image;
4
5
6
7 public class Murcielago {
8     private double x, y;
9     private int ancho = 30;
10    private int alto = 30;
11    private boolean vivo = true;
12    private Image imagen;
13    private Image imagenCongelado;
14    private Image imagenQuemado;
15    private boolean congelado = false;
16    private long tiempoDescongelacion = 0;
17    private boolean quemado = false;
18    private long tiempoDesquemado = 0;
19    private long muertoVisibleHasta = 0;
20    private boolean eliminadoPorJugador = false;
21
22    public Murcielago(double x, double y) {}
23
24
25
26    public void marcarComoEliminadoPorJugador() {}
27
28
29    public boolean fueEliminadoPorJugador() {}
30
31
32
33    // CUANDO LA FUNCION ESTA ACTIVA DURA 3 SEGUNDOS
34    public void congelar(long tiempoActual) {}
35
36
37
38
39    public void eliminarSinAnimacion() {}
40
41
42
43
44    // CUANDO LA FUNCION ESTA ACTIVA MUESTRA LA IMAGEN DE FUEGO POR 2 SEGUNDOS
45    public void quemar(long ahora) {}
46
47
48
49
50
51
52
53
54
55
56
57
58
```

Producción propia – primeras 58 líneas de código Murcielago.java

- Funcionalidades principales:
 - Movimiento: Método para actualizar la posición del murciélago, persiguiendo a Gondolf y esquivando rocas.

- Estados: Maneja los estados de vivo, congelado y quemado, incluyendo la lógica de duración de los efectos.
- Colisiones y Ataque: Detecta colisiones con Gondolf y le resta vida si colisionan.
- Muerte: Método para marcar al murciélago como muerto y gestionar su desaparición.
- Dibujado: Método para dibujar la imagen correspondiente al estado actual del murciélago.

NIVEL:

Nivel	Enemigos necesarios	Velocidad	Notas adicionales
1	5	Lenta	Nivel introductorio
2	10	Media	Mayor intensidad
3	20	Rápida	Aumenta dificultad
4	Jefe Final	Alta	Aparece jefe con mecánicas especiales

```

1 package juego;
2
3 public class Nivel {
4     private int numero;
5     private int murcielagosObjetivo;
6     private double velocidadMurcielagos;
7     private String nombre;
8
9
10    // FUNCION PARA SABER EN QUE NIVEL SE ESTA
11    public Nivel(int numero, int murcielagosObjetivo, double velocidadMurcielagos, String nombre) {}
12
13
14    public int getNumero() {}
15
16    public int getMurcielagosObjetivo() {}
17
18    public double getVelocidadMurcielagos() {}
19
20    public String getNombre() {}
21
22 }

```

Producción propia – código Nivel.java

- Funcionalidades principales:
 - Getters: Métodos para acceder a las propiedades del nivel.

JEFE FINAL:

- Vida inicial: 100 puntos.
- Ataques especiales:
 - Abanico: dispara proyectiles cada 8 segundos (daño: 5 puntos).
 - Bomba: lanza una bomba cada 7 segundos (daño continuo de 1 punto por tick si Gondolf permanece en el área).
- Movimiento:
 - Desaparece durante 2 segundos y reaparece en una posición aleatoria durante 3 segundos.
- Recibe daño:
 - 5 puntos por clic.
 - 30 puntos por hechizo de fuego.
 - 20 puntos por hechizo de hielo.

```
JefeFinal.java X
1 package juego;
2
3 import java.awt.Color;
4
5
6
7
8
9
10 public class JefeFinal {
11     private double x, y;
12     private int vida;
13     private Image imagen;
14     private boolean visible;
15     private long tiempoCambioVisibilidad;
16     private boolean enModoVisible;
17
18     private long ultimoAtaqueDistancia;
19     private long ultimoAtaqueArea;
20     private long ultimoAtaqueBomba;
21     private long ultimoAtaqueAbanico;
22
23     private static final long TIEMPO_VISIBLE = 3000; // TIEMPO DE VISIBILIDAD DEL JEFE
24     private static final long TIEMPO_INVISIBLE = 2000; // TIEMPO DE INVISIBILIDAD DEL JEFE
25     private static final long INTERVALO_ATAQUE_DISTANCIA = 4000;
26     private static final long INTERVALO_ATAQUE_AREA = 6000;
27     private static final long INTERVALO_ATAQUE_BOMBA = 7000; // ATAQUE DE BOMBA CADA 7 SEGUNDOS
28     private static final long INTERVALO_ATAQUE_ABANICO = 8000; // ATAQUE ABANICO CADA 8 SEGUNDOS
29
30     private boolean entrando = true;
31     private double yObjetivo = 200;
32
33     private Random random = new Random(); // DEVUELVE DE MANERA ALEATORIA
34     private ArrayList<Proyectil> proyectiles = new ArrayList<>(); // LISTA DE PROYECTILES
```

Producción propia – Primeras 34 líneas de código JefeFinal.java

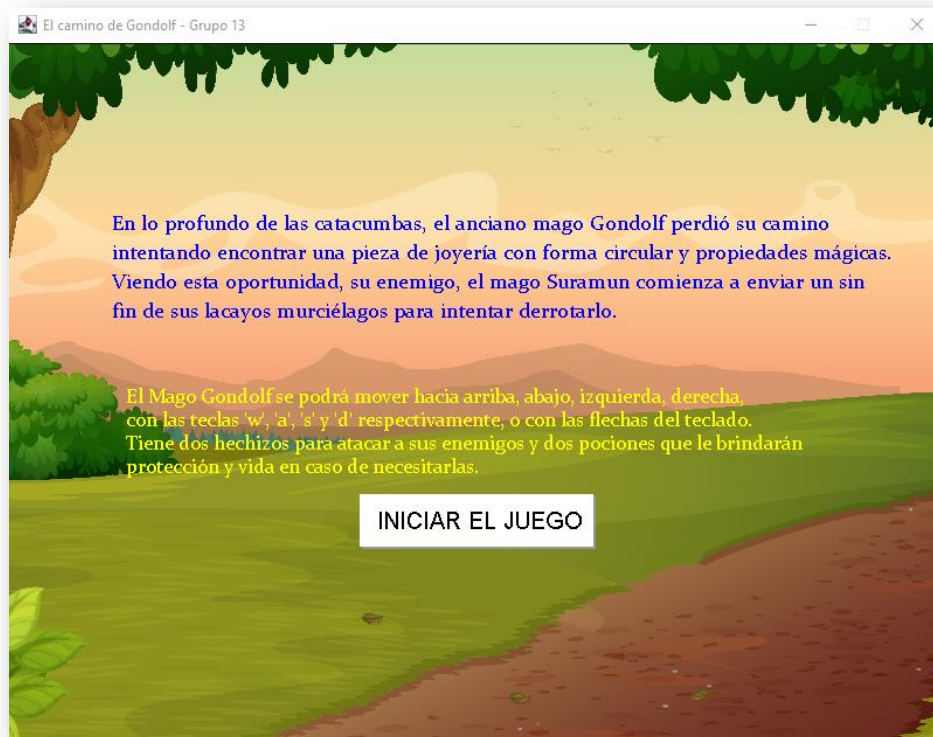
- **Funcionalidades principales:**
 - Ataques especiales: Métodos para acceder a los ataques de bomba y abanico.
 - Movimientos: Método para actualizar la posición.
 - Apariciones: Método para realizar apariciones aleatorias.
 - Daño: Métodos para recibir daños causados por Gondolf.

Problemas encontrados, decisiones tomadas y soluciones encontradas:

- Tuvimos problemas iniciales para sincronizar el movimiento de los enemigos (murciélagos) con los obstáculos (rocas), ya que a veces los enemigos se quedaban atascados o se superponían con las rocas. Para solucionarlo, implementamos una verificación de colisiones y reajuste de posiciones, asegurándonos de que los murciélagos busquen caminos alternativos si detectan una colisión inminente.
- La regeneración de magia de Gondolf no funcionaba correctamente al principio. Esto se debía a que no se estaba actualizando correctamente el tiempo transcurrido desde la última recarga. Se resolvió almacenando un tiempo de referencia con ``System.currentTimeMillis()`` y sumando un control por tick.
- En algunos casos, los hechizos no afectaban a los enemigos aunque se encontraran en el área de efecto. Al revisar la lógica, descubrimos que los cálculos de distancia no eran precisos debido a un error en la fórmula de verificación. Se corrigió utilizando ``Math.hypot`` para comparar distancias de manera más confiable.
- El manejo de los estados visuales de congelación y quemadura fue complicado, ya que había que coordinar los efectos gráficos, la lógica del estado y su duración. Para resolverlo, se encapsuló la lógica en la clase ``EfectoVisual`` y se agregó control de ticks para definir la duración de los efectos.

JUGABILIDAD Y EXPERIENCIA DEL USUARIO

- **Controles de movimiento:** Flechas del teclado o teclas "W", "A", "S", "D".
- **Selección de hechizos:** Clic en el menú lateral.
- **Ataque a distancia:** Clic izquierdo del mouse.
- **Ataque cuerpo a cuerpo:** Clic directo sobre el enemigo.
- **Indicadores en pantalla:** vida, magia, hechizo activo, enemigos eliminados y barra del jefe.
- **Condición de derrota:** Vida de Gondolf llega a cero.
- **Condición de victoria:** Derrotar al jefe final en el nivel 4.





Se puede observar en esta imagen el menú, donde esta la barra de vida, la barra de magia y la cantidad de enemigos eliminados. Este ultimo contador se reinicia con cada nivel.

También se pueden observar los botones dedicados a cada hechizo cuando está disponible.

En esta otra imagen se puede ver el estado del botón cuando la magia no es suficiente para ser utilizado, ya que se pone en modo inactivo.



CONCLUSIÓN

Este trabajo práctico nos permitió aplicar los conceptos aprendidos durante el cuatrimestre, integrando programación orientada a objetos con lógica de juegos 2D. Aprendimos a organizar el código de forma modular, trabajar con múltiples clases, listas y estados de juego.

Además, desarrollamos habilidades para resolver problemas de colisiones, sincronización de eventos y gestión de recursos como la magia y la vida del personaje. También logramos implementar mejoras visuales que hacen al juego más intuitivo y entretenido para el usuario.

En resumen, esta experiencia nos dio una comprensión más profunda del ciclo de desarrollo de un videojuego, desde la lógica del personaje hasta la interacción entre todos los elementos del entorno.