Programación I - Primer Semestre 2025

Trabajo Práctico: El camino de Gondolf

Integrantes:

Nicolas Diaz Rueda:

- 44355742
- Nicodiazrojo05@gmail.com

Ariel Alejandro Espinoza Aldave:

- 43568660
- arielespinozaaldave@gmail.com

Carlos Josue Tapia:

- 43568660
- josuuetapiaa@hotmail.com

Lucas Guillermo Ibarra:

- 45150045
- Juveriver134@gmail.com

Introducción

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.

- **Descripción de las diferentes clases:**

Juego: Esta clase representa el núcleo principal del videojuego. Controla el flujo completo del juego, la inicialización de personajes y los diferentes elementos como el mago, los enemigos, hechizos, rocas y pociones, así como también la lógica principal que se ejecuta en cada ciclo (tick). Además se encarga de la creación de enemigos, hechizos, y las condiciones de victoria o derrota. Es la clase que controla todos los componentes y mantiene el estado general del juego en ejecución.
Métodos:

-Juego(): Constructor: inicializa el entorno gráfico, jugador, rocas, fondo, lista de pociones, y lanza el juego.

-tick(): Método principal que se ejecuta en cada frame. Maneja el menú, dibuja elementos, verifica inputs, lanza hechizos, genera enemigos y evalúa condiciones de victoria o derrota.

-reiniciarJuego(): Restaura el estado del juego al inicio: reinicia mago, enemigos, hechizos, pociones, menú y rocas.

-redimensionarImagen(imagen, ancho, alto): Devuelve la imagen escalada al tamaño deseado.

-generarPosicionAleatoriaFueraDePantalla(): Genera una coordenada aleatoria fuera de los bordes del entorno, para aparición de enemigos.

Menu: Esta clase representa el menú lateral del juego, que a su vez muestra información del jugador, como por ejemplo la vida, magia, kills y además permite seleccionar entre tres hechizos a través de botones visuales.
Metodos:

-public Menu(int x, int ancho, int alto): Inicializa el menú con una posición y tamaño determinados. Crea tres botones de hechizo y los posiciona en pantalla. Por defecto, no hay hechizos seleccionados.

-public void dibujar(Entorno entorno, int vida, int magia, int kills): Dibuja el menú en la pantalla: muestra un fondo oscuro, la vida, la magia, los enemigos eliminados y todos los botones de hechizos (resaltando el seleccionado).

-public void actualizarSeleccion(Entorno entorno): Detecta si el jugador hizo clic con el mouse sobre alguno de los botones y, de ser así, lo selecciona.

-public int getHechizoSeleccionado(): Devuelve el índice del hechizo actualmente seleccionado. Si no hay ninguno, devuelve -1.

-public void deseleccionar(): Quita la selección de cualquier hechizo (vuelve el índice a -1).

-public BotonHechizo getBoton(int indice): Devuelve el botón de hechizo que se encuentra en la posición indicada del arreglo.

-public boolean mouseEstaEnMenu(int mouseX): Indica si la posición horizontal del mouse está dentro del área visible del menú.

Gondolf: La clase `Gondolf` representa al personaje principal del juego, un mago controlado por el jugador. Quien es el responsable de su movimiento en pantalla, de utilizar sus habilidades, de eliminar a los murciélagos, etc..
Métodos:

- Gondolf(): Constructor que inicializa la posición, magia, vida, dimensiones e imagen del personaje.
- mover(): Controla el movimiento del personaje en función de las teclas presionadas. También valida que el movimiento no lo saque de los límites de la pantalla ni cause colisión con rocas.
- dibujar(): Dibuja al personaje en pantalla usando su imagen.
- recibirDanio(): Resta vida al personaje al recibir daño.
- tieneMagiaSuficiente(): Verifica si el personaje tiene suficiente magia para lanzar un hechizo.
- lanzarHechizo(): Disminuye la cantidad de magia si es suficiente para lanzar un hechizo.
- incrementaMagia(): Aumenta la magia del personaje.
- magoSimulado(): Permite configurar el tamaño del personaje simulado (usado para verificar colisiones sin mover al personaje real).
- Getters: Métodos para obtener valores de x, y, vida, magia, ancho y alto.

Hechizo: La clase hechizo representa una habilidad mágica lanzada por el jugador. Cada hechizo tiene una posición, dirección, color o imagen, duración limitada y puede afectar enemigos si colisiona con ellos.
Metodos:

-public Hechizo(): Inicializa el hechizo con posición, dirección, imagen, color, tamaño, duración y costo de magia.
-public void mover(): Mueve el hechizo en la dirección previamente calculada multiplicando por su velocidad.
-public boolean estaDentroDePantalla(int ancho, int alto): Devuelve true si el hechizo todavía está dentro del área visible de la pantalla.
-public void dibujar(Entorno entorno): Dibuja el hechizo en pantalla. Usa una imagen si está disponible, o un círculo de color si no.
-public boolean estaActivo(int tiempoActual): Devuelve true si el hechizo no ha superado su duración máxima.
-public boolean afectaA(Murcielago m): Verifica si el hechizo colisionó con un murciélago, comparando la distancia entre ellos.
-public int getCostoMagia(): Devuelve el costo de magia del hechizo.

Murciélago: La clase murciélago representa a un enemigo dentro del juego que tiene como objetivo perseguir al personaje principal. Puede moverse por el entorno, colisionar con el jugador para causarle daño y también ser alcanzado por hechizos. Si un hechizo lo impacta, el murciélago es eliminado del juego.
Metodos:

-Murcielago(double xInicial, double yInicial, double velocidad): Inicializa la posición, velocidad, tamaño y estado del murciélago. Carga su imagen visual.
-void dibujar(Entorno entorno): Dibuja el murciélago en la pantalla si está vivo.

-void moverHacia(Gondolf gondolf): Mueve al murciélago en dirección hacia el personaje principal (Gondolf), si está vivo.
-boolean colisionaCon(double x, double y, double radio): Verifica si el murciélago colisiona con un objeto circular (por ejemplo, un hechizo) dado por su posición (x, y) y su radio.
-void morir(): Marca al murciélago como muerto (vivo = false), para que deje de moverse y dibujarse.
-boolean getVivo(): Devuelve true si el murciélago está vivo, false si está muerto.

Poción: Esta clase se encarga de mostrar una poción de vida en el juego durante un tiempo limitado. Si el personaje principal se acerca, la misma le restaura un porcentaje de vida.
Metodos:

-Pocion(double x, double y, int tiempoCreacion): Constructor que inicializa la posición, el tiempo de creación, la visibilidad y carga la imagen de la poción.
-void dibujar(Entorno entorno): Dibuja la poción en pantalla sólo si está visible.
-void actualizar(int tiempoActual, Gondolf gondolf): Actualiza el estado de la poción según el tiempo actual y la posición de Gondolf.
-boolean estaVisible(): Devuelve si la poción está visible o no, para que otras clases puedan consultarlo.

Roca: La clase roca representa un obstáculo fijo en el entorno del juego. Su función principal es impedir el movimiento del personaje si colisiona con ella.
Metodos:

-Roca(double x, double y): Constructor que inicializa la posición, tamaño y carga la imagen de la roca.
-void dibujar(Entorno entorno): Dibuja la imagen de la roca en pantalla, con una escala del 10% (0.1).
-boolean colisionaCon(Gondolf gondolf): Verifica si el personaje Gondolf colisiona con la roca usando detección de colisión rectangular.
-(getX(), getY(), getAncho(), getAlto(): Son los métodos para obtener las propiedades de posición y tamaño de la roca.

Hechizo: La clase hechizo representa un botón en la interfaz del juego que permite seleccionar un hechizo para lanzar. Cada botón muestra el nombre del hechizo y su costo en magia.
Metodos:

-Hechizo(): Inicializa la posición, dirección normalizada hacia el destino, tamaño, costo de magia, color, duración, tiempo de creación y carga la imagen escalada del hechizo.
-void mover(): Actualiza la posición del hechizo sumando el vector dirección multiplicado por la velocidad.

-boolean estaDentroDePantalla(int ancho, int alto): Verifica si la posición actual del hechizo está dentro de los límites dados (ancho y alto) de la pantalla.
-void dibujar(Entorno entorno): Dibuja la imagen del hechizo en la posición actual; si no hay imagen, dibuja un círculo con el color definido.
-boolean estaActivo(int tiempoActual): Indica si el hechizo sigue activo basado en si no ha superado su duración desde tickInicial.
-boolean afectaA(Murcielago m): Determina si el hechizo colisiona con un murciélago, verificando si la distancia entre ellos es menor a la suma de sus radios.
-(getCostoMagia(), getX(), getY(), getRadio(): Métodos para obtener valores privados importantes de la clase.

Implementación:
A continuación se presenta el código fuente del proyecto, correctamente formateado y comentado cuando corresponde. El desarrollo se realizó en Java utilizando el entorno de desarrollo Eclipse. El proyecto se encuentra disponible en el siguiente repositorio:
https://github.com/Aespinozaaldave/diaz-espinoza-tapia-ibarra-tp-p1.git

Juego.java: Clase principal donde se inicializa el entorno y se controla la lógica general del juego.

```
package juego;

// Importación de herramientas necesarias
import entorno.Herramientas;
import java.awt.Color;
import java.awt.Image;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import entorno.Entorno;
import entorno.InterfaceJuego;

// Clase principal del juego
public class Juego extends InterfaceJuego {
	private Entorno entorno; // Entorno gráfico del juego
	private Image fondo; // Imagen de fondo

	// Enum que representa los distintos estados del juego
	private enum EstadoJuego {
	    MENU,     // Pantalla de inicio
	    JUGANDO,  // El juego está activo
	    TERMINADOGANADO,// El juego terminó (victoria)
	    TERMINADODERROTA// El juego terminó (derrota)
	}
```

```
     // Permite volver todos los valores nuevamente a su inicio
     private void reiniciarJuego() {
          // Reinicia el estado del juego
          estado = EstadoJuego.JUGANDO;

          // Reinicia al mago
          mago = new Gondolf(anchoPantalla / 2 - 100,
altoPantalla / 2, magiaInicial, vidaInicial);

          // Reinicia enemigos
          enemigos = new Murcielago[10];
          totalGenerados = 0;
          tiempoUltimoMurcielago = 0;

          // Reinicia hechizos
          hechizos = new Hechizo[10];

          // Reinicia pociones
          pociones.clear();

          // Reinicia contador de enemigos eliminados
          enemigosEliminados = 0;

          // Reinicia menú
          menu.deseleccionar();

          // (Opcional) Si querés reiniciar la posición de las
rocas:
          rocas[0] = new Roca(200, 300);
          rocas[1] = new Roca(400, 150);
          rocas[2] = new Roca(300, 500);
          rocas[3] = new Roca(100, 200);
          rocas[4] = new Roca(500, 300);
     }

     // Estado actual del juego
     private EstadoJuego estado = EstadoJuego.MENU;

     private List<Pocion> pociones; // Lista de pociones activas
en el juego

     private int totalGenerados = 0; // Cuántos murciélagos se
han generado hasta ahora
     private int maxGenerados = 50; // Límite máximo de
murciélagos a generar
     double velocidadMurcielagos = 2.0; // Velocidad de
movimiento de los murciélagos
     private Murcielago[] enemigos = new Murcielago[10]; //
Arreglo de enemigos activos
     private int tiempoUltimoMurcielago = 0; // Tiempo en que se
generó el último enemigo
     private int intervalo = 1000; // Intervalo mínimo entre
apariciones de murciélagos (en ticks)
```

```
     private Hechizo[] hechizos = new Hechizo[10]; // Hechizos
activos


     int anchoPantalla = 800;
     int altoPantalla = 600;

     private Roca[] rocas = new Roca[5]; // Obstáculos en el
escenario

     // Estado inicial de Gondolf
     int magiaInicial = 100;
     int vidaInicial = 100;
     Gondolf mago = new Gondolf(anchoPantalla / 2 - 100,
altoPantalla / 2, magiaInicial, vidaInicial);

     Menu menu = new Menu(600, 200, altoPantalla); // Menú
lateral de hechizos
     private int enemigosEliminados = 0; // Contador de enemigos
derrotados

     // Constructor principal del juego
     Juego() {
          // Inicializa el entorno gráfico
          this.entorno = new Entorno(this, "El camino de Gondolf
- Grupo 7 - v1", anchoPantalla, altoPantalla);

          // Carga y ajusta el fondo
          this.fondo = Herramientas.cargarImagen("imagenes/fondo
amarillo.png");
          this.fondo = redimensionarImagen(this.fondo,
anchoPantalla, altoPantalla);

          // Inicializa la lista de pociones
          pociones = new ArrayList<>();

          // Crea las rocas en posiciones fijas
          rocas[0] = new Roca(200, 300);
          rocas[1] = new Roca(400, 150);
          rocas[2] = new Roca(300, 500);
          rocas[3] = new Roca(100, 200);
          rocas[4] = new Roca(500, 300);

          this.entorno.iniciar(); // Inicia el ciclo principal
del entorno
     }

     // Redimensiona una imagen al tamaño especificado
     private Image redimensionarImagen(Image imagen, int ancho,
int alto) {
          return imagen.getScaledInstance(ancho, alto,
Image.SCALE_SMOOTH);
     }

     // Genera una posición aleatoria fuera de los bordes de la
```

```
pantalla
     private double[] generarPosicionAleatoriaFueraDePantalla()
{
          int borde = (int) (Math.random() * 4); // Elige un
borde al azar
          double x = 0;
          double y = 0;

          // Según el borde, genera coordenadas fuera del área
visible
          if (borde == 0) {
               x = Math.random() * 600;
               y = -20;
          } else if (borde == 1) {
               x = Math.random() * 600;
               y = altoPantalla + 20;
          } else if (borde == 2) {
               x = -20;
               y = Math.random() * altoPantalla;
          } else if (borde == 3) {
               x = 620;
               y = Math.random() * altoPantalla;
          }
          double[] posicion = { x, y };
          return posicion;
     }

     // Método principal que se ejecuta en cada "tick" del juego
     public void tick() {
          int tiempoActual = entorno.tiempo(); // Tiempo actual
del entorno
          // Dibuja el fondo centrado
          entorno.dibujarImagen(fondo, entorno.ancho() / 2,
entorno.alto() / 2, 0);
          if (!(estado == EstadoJuego.JUGANDO)) {
             // Dibuja fondo
             //entorno.dibujarRectangulo(anchoPantalla / 2,
altoPantalla / 2, anchoPantalla, altoPantalla, 0,
Color.LIGHT_GRAY);

             // Título del juego
                if (estado == EstadoJuego.MENU) {
                     entorno.cambiarFont("Arial", 36,
Color.BLACK);
                     entorno.escribirTexto("El camino de
Gondolf", anchoPantalla / 2 - 180, 150);
                }
             // Coordenadas de botones
             int botonJugarX = anchoPantalla / 2;
             int botonJugarY = 250;
             int botonSalirX = anchoPantalla / 2;
             int botonSalirY = 350;
             int anchoBoton = 200;
             int altoBoton = 60;
```

```
            //Textos dependiendo si se gano o perdio el juego
            if (estado == EstadoJuego.TERMINADOGANADO) {
               entorno.cambiarFont("Arial", 36, Color.GREEN);
                  entorno.escribirTexto("¡Has ganado!",
anchoPantalla / 2 - 110, 150);
            }
            if (estado == EstadoJuego.TERMINADODERROTA) {
               entorno.cambiarFont("Arial", 36, Color.RED);
                  entorno.escribirTexto("¡Has perdido!",
anchoPantalla / 2 - 110, 150);
            }

            // Botón JUGAR
            entorno.dibujarRectangulo(botonJugarX, botonJugarY,
anchoBoton, altoBoton, 0, Color.GREEN);
            entorno.cambiarFont("Arial", 24, Color.BLACK);
            entorno.escribirTexto("JUGAR", botonJugarX - 40,
botonJugarY + 10);

            // Botón SALIR
            entorno.dibujarRectangulo(botonSalirX, botonSalirY,
anchoBoton, altoBoton, 0, Color.RED);
            entorno.cambiarFont("Arial", 24, Color.BLACK);
            entorno.escribirTexto("SALIR", botonSalirX - 35,
botonSalirY + 10);

            // Detectar clics del mouse
            if
(entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
                int mouseX = entorno.mouseX();
                int mouseY = entorno.mouseY();

                // Si se hizo clic en "Jugar"
                if (mouseX >= botonJugarX - anchoBoton / 2 &&
mouseX <= botonJugarX + anchoBoton / 2 &&
                    mouseY >= botonJugarY - altoBoton / 2 &&
mouseY <= botonJugarY + altoBoton / 2) {
                    reiniciarJuego();
                }

                // Si se hizo clic en "Salir"
                if (mouseX >= botonSalirX - anchoBoton / 2 &&
mouseX <= botonSalirX + anchoBoton / 2 &&
                    mouseY >= botonSalirY - altoBoton / 2 &&
mouseY <= botonSalirY + altoBoton / 2) {
                    System.exit(0); // Cierra el programa
                }

            }

            return; // No ejecuta más del tick si está en el
menú
         }
          if (estado == EstadoJuego.JUGANDO) {
```

```
                // Verifica condiciones de fin de juego
                if (enemigosEliminados >= maxGenerados) {
                     estado = EstadoJuego.TERMINADOGANADO;
                }
                if (mago.getVida() <= 0) {
                     estado = EstadoJuego.TERMINADODERROTA;
                }

                menu.actualizarSeleccion(entorno); // Verifica
selección de hechizo
                menu.dibujar(entorno, mago.getVida(),
mago.getMagia(), enemigosEliminados);
                mago.dibujar(entorno); // Dibuja al personaje
principal
                mago.mover(entorno, rocas); // Permite
movimiento, evitando rocas
                double velocidad = velocidadMurcielagos + 0.5 *
(enemigosEliminados / 15); // Se incremetna la velocidad 0.5 cada
15 enemigos eliminados

                // Dibuja todas las rocas
                for (int i = 0; i < rocas.length; i++) {
                     if (rocas[i] != null) {
                          rocas[i].dibujar(entorno);
                     }
                }

                // Generación de murciélagos enemigos
                if (tiempoActual - tiempoUltimoMurcielago >=
intervalo && totalGenerados < maxGenerados) {
                     for (int i = 0; i < enemigos.length; i++) {
                          if (enemigos[i] == null) {
                               double[] pos =
generarPosicionAleatoriaFueraDePantalla();
                               enemigos[i] = new
Murcielago(pos[0], pos[1], velocidad);
                               tiempoUltimoMurcielago =
tiempoActual;
                               totalGenerados++;

                               break;
                          }
                     }
                }

                // Lanzamiento de hechizos con el mouse
                String rutaImagen = ""; // Ruta de la imagen de
hechizos
                if
(entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO) &&
!menu.mouseEstaEnMenu(entorno.mouseX())) {
                     int seleccionado =
menu.getHechizoSeleccionado();
                     if (seleccionado != -1) {
                          BotonHechizo boton =
```

```
menu.getBoton(seleccionado);
                            if
(mago.tieneMagiaSuficiente(boton.getCostoMagia())) {
                                  for (int i = 0; i <
hechizos.length; i++) {
                                        if (hechizos[i] == null) {
                                              // Crea hechizo según
el botón seleccionado
                                              if (seleccionado == 0)
{
                                                    rutaImagen =
"src/imagenes/explosionn.png";
                                                    hechizos[i] = new
Hechizo(mago.getX(), mago.getY(), entorno.mouseX(),

entorno.mouseY(), 20, boton.getCostoMagia(), Color.CYAN,
entorno.tiempo(),
                                                                500,
rutaImagen);
                                              } else if
(seleccionado == 1) {
                                                    rutaImagen =
"src/imagenes/fuego2.png";
                                                    hechizos[i] = new
Hechizo(entorno.mouseX(), entorno.mouseY(), entorno.mouseX(),

entorno.mouseY(), 35, boton.getCostoMagia(), Color.RED,
entorno.tiempo(),
                                                                400,
rutaImagen);
                                              } else if
(seleccionado == 2) {
                                                    rutaImagen =
"src/imagenes/agujeroNegro.png";
                                                    hechizos[i] = new
Hechizo(entorno.mouseX(), entorno.mouseY(), entorno.mouseX(),

entorno.mouseY(), 60, boton.getCostoMagia(), Color.BLACK,
entorno.tiempo(),
                                                                300,
rutaImagen);
                                              }

mago.lanzarHechizo(boton.getCostoMagia());
                                              menu.deseleccionar();
                                              break;
                                        }
                                  }
                            }
                      }
                }

                // Lógica de enemigos (murciélagos)
                for (int i = 0; i < enemigos.length; i++) {
                      Murcielago m = enemigos[i];
```

```
                    if (m != null && m.getVivo()) {
                        m.moverHacia(mago);
                        if (m.getX() < 600) {
                            m.dibujar(entorno);
                        }
                        // Verifica colisión con el mago
                        if (m.colisionaCon(mago.getX(),
mago.getY(), 20)) {
                            mago.recibirDanio(10);
                            enemigos[i] = null;
                            enemigosEliminados++;
                        }
                    }
                }

                // Lógica de los hechizos
                for (int i = 0; i < hechizos.length; i++) {
                    Hechizo h = hechizos[i];
                    if (h != null) {
                        if (!h.estaDentroDePantalla(600,
altoPantalla)) {
                            hechizos[i] = null;
                        }
                        h.dibujar(entorno);
                        h.mover();

                        // Verifica si el hechizo impacta un
enemigo
                        for (int j = 0; j < enemigos.length;
j++) {
                            Murcielago m = enemigos[j];
                            if (m != null && m.getVivo() &&
h.afectaA(m)) {
                                m.morir(); // Marca el
murciélago como muerto

                                // Genera una poción cada 5
enemigos eliminados si está dentro de los límites
                                // visibles
                                if (enemigosEliminados % 5
== 0 && enemigosEliminados > 0) {
                                    if (m.getX() >= 0 &&
m.getX() <= anchoPantalla && m.getY() >= 0
                                            && m.getY()
<= altoPantalla) {
                                        pociones.add(new
Pocion(m.getX(), m.getY(), entorno.tiempo()));
                                    }
                                }

                                enemigos[j] = null;
                                enemigosEliminados++;

                                // Recupera un poco de
magia
```

```
                                        if (mago.getMagia() <
magiaInicial) {

mago.incrementaMagia((int) (magiaInicial * 0.1));
                                        }
                                  }
                            }

                            if (!h.estaActivo(tiempoActual)) {
                                  hechizos[i] = null;
                            }
                      }
                }

                // Lógica de las pociones
                Iterator<Pocion> it = pociones.iterator();
                while (it.hasNext()) {
                      Pocion p = it.next();
                      p.dibujar(entorno); // Muestra la poción
                      p.actualizar(tiempoActual, mago); //
Verifica si Gondolf la recoge
                      if (!p.estaVisible()) {
                            it.remove(); // Elimina la poción si
ya fue usada o expiró
                      }
                }

           }

     }

     // Método principal para iniciar el juego
     @SuppressWarnings("unused")
     public static void main(String[] args) {
           Juego juego = new Juego();
     }
}
```

Gondolf.java: Define al personaje principal, su movimiento y acciones.

```
package juego;

import java.awt.Image;
import javax.swing.ImageIcon;
import entorno.Entorno;

// Clase que representa al personaje principal, Gondolf
public class Gondolf {
     private double x; // Posición horizontal del personaje
```

```
     private double y; // Posición vertical del personaje
     private double velocidad; // Velocidad de movimiento
     private int magia; // Puntos de magia actuales
     private int vida; // Puntos de vida actuales
     private double ancho; // Ancho del personaje (para
colisiones)
     private double alto; // Alto del personaje (para
colisiones)

     // Imágenes del personaje según la dirección de movimiento
     private Image imagenIzquierda;
     private Image imagenDerecha;
     private Image imagenArriba;
     private Image imagenAbajo;
     private Image imagenActual;  // Imagen que se está usando
actualmente
     private Image imagenFrente;  // Imagen cuando está quieto

     // Constructor: inicializa la posición, atributos y carga
las imágenes del mago
     public Gondolf(double xInicial, double yInicial, int
magiaInicial, int vidaInicial) {
          this.x = xInicial;
          this.y = yInicial;
          this.velocidad = 5.0;
          this.magia = magiaInicial;
          this.vida = vidaInicial;
          this.ancho = 16;
          this.alto = 30;

          // Carga y escala las imágenes del mago
          this.imagenIzquierda = new
ImageIcon("src/imagenes/mago izquierda
2.0.png").getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);
        this.imagenDerecha   = new ImageIcon("src/imagenes/mago
derecha 2.0.png").getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);
        this.imagenArriba    = new ImageIcon("src/imagenes/mago
espalda 2.0.png").getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);
        this.imagenAbajo     = new ImageIcon("src/imagenes/mago
abajo 2.0.png").getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);
        this.imagenFrente    = new ImageIcon("src/imagenes/mago
quieto.png").getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);

        this.imagenActual = imagenFrente; // Imagen inicial
(quieto)
     }

     // Método para mover al personaje con teclas y detectar
colisiones con rocas
     public void mover(Entorno entorno, Roca[] rocas) {
```

```
           double nuevoX = x;
          double nuevoY = y;
          boolean seMovio = false; // Bandera para saber si se
presionó una tecla

          // Movimiento hacia arriba
          if (entorno.estaPresionada('W') ||
entorno.estaPresionada(entorno.TECLA_ARRIBA)) {
              nuevoY -= velocidad;
              this.imagenActual = imagenArriba;
              seMovio = true;
          }

          // Movimiento hacia abajo
          if (entorno.estaPresionada('S') ||
entorno.estaPresionada(entorno.TECLA_ABAJO)) {
              nuevoY += velocidad;
              this.imagenActual = imagenAbajo;
              seMovio = true;
          }

          // Movimiento hacia la izquierda
          if (entorno.estaPresionada('A') ||
entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) {
              nuevoX -= velocidad;
              this.imagenActual = imagenIzquierda;
              seMovio = true;
          }

          // Movimiento hacia la derecha
          if (entorno.estaPresionada('D') ||
entorno.estaPresionada(entorno.TECLA_DERECHA)) {
              nuevoX += velocidad;
              this.imagenActual = imagenDerecha;
              seMovio = true;
          }

          // Si no se presionó ninguna tecla, muestra la imagen
de frente
          if (!seMovio) {
              this.imagenActual = imagenFrente;
          }

          // Verifica si la nueva posición está dentro de los
límites de la pantalla
          boolean dentroDePantalla = nuevoX - ancho / 2 >= 0 &&
                                     nuevoX + ancho / 2 <= 600 &&
                                     nuevoY - alto / 2 >= 0 &&
                                     nuevoY + alto / 2 <= 600;

          if (dentroDePantalla) {
             // Crea una instancia simulada para verificar
colisiones
             Gondolf simulado = new Gondolf(nuevoX, nuevoY, magia,
vida);
```

```
            simulado.magoSimulado(ancho, alto);

            boolean colisiona = false;

            // Verifica si la posición colisiona con alguna roca
            for (int i = 0; i < rocas.length; i++) {
                Roca r = rocas[i];
                if (r != null && r.colisionaCon(simulado)) {
                    colisiona = true;
                    break;
                }
            }

            // Si no hay colisión, actualiza la posición del
personaje
            if (!colisiona) {
                this.x = nuevoX;
                this.y = nuevoY;
            }
        }
    }

    // Dibuja al personaje en pantalla con su imagen actual
    public void dibujar(Entorno entorno) {
         entorno.dibujarImagen(imagenActual, x, y, 0);
    }

    // Reduce la vida al recibir daño
    public void recibirDanio(int cantidad) {
         vida -= cantidad;
         if (vida < 0) {
              vida = 0;
         }
    }

    // Aumenta la vida al recibir una poción
    public void recibirVida(int cantidad) {
        vida += cantidad;
        if (vida > 100) { // Límite máximo
            vida = 100;
        }
    }

    // Verifica si tiene suficiente magia para lanzar un
hechizo
    public boolean tieneMagiaSuficiente(int costo) {
       return magia >= costo;
   }

    // Resta la cantidad de magia usada en un hechizo
    public void lanzarHechizo (int cantidad) {
         if (tieneMagiaSuficiente(cantidad)) {
              magia -= cantidad;
         }
    }
```

```
     // Aumenta la magia (por ejemplo, al matar enemigos)
     public void incrementaMagia(int cantidad) {
          magia += cantidad;
     }

     // Método para ajustar el tamaño del mago simulado (usado
en colisiones)
     public void magoSimulado(double ancho, double alto) {
         this.ancho = ancho;
         this.alto = alto;
     }

     // Incrementa vida y limita a 100 si se excede (duplicado
innecesario del método anterior)
     public void incrementaVida(int cantidad) {
         this.vida += cantidad;
         if (this.vida > 100) {
             this.vida = 100;
         }
     }

     // Métodos para obtener atributos del personaje
    public double getX() { return x; }
    public double getY() { return y; }
    public int getVida() { return vida; }
    public int getMagia() { return magia; }
    public double getAncho() { return ancho; }
    public double getAlto() { return alto; }
}
```

Hechizo.java: Representa los ataques que lanza el mago.

```
package juego; // Define el paquete al que pertenece esta clase

// Importación de clases necesarias para gráficos y entorno
import java.awt.Color;
import entorno.Entorno;
import java.awt.Image;
import javax.swing.ImageIcon;

public class Hechizo {
    // Posición actual del hechizo
    private double x;
    private double y;

    // Dirección de movimiento del hechizo (normalizada)
    private double direccionX;
    private double direccionY;
```

```
    // Velocidad del hechizo
    private double velocidad;

    // Tamaño del hechizo (radio para dibujo y colisión)
    private double radio;

    // Costo en magia para lanzar el hechizo
    private int costoMagia;

    // Color del hechizo (usado si no se carga imagen)
    private Color color;

    // Duración del hechizo en milisegundos
    private int duracion;

    // Tiempo (tick) en que se lanzó el hechizo
    private int tickInicial;

    // Imagen del hechizo
    private Image imagenHechizo;

    // Constructor del hechizo
    public Hechizo(
        double xInicial, double yInicial,        // Posición
inicial del hechizo
        double destinoX, double destinoY,        // Coordenadas
destino (por ejemplo, mouse)
        double radio,                            // Radio del
hechizo
        int costoMagia,                          // Cuánto mana
gasta
        Color color,                             // Color del
hechizo
        int tickInicial,                         // Tiempo en que
se creó
        int duracion,                            // Cuánto dura
en pantalla
        String rutaImagen                            // Ruta de
la imagen a utilizar
    ) {
        // Inicializa variables básicas
        this.x = xInicial;
        this.y = yInicial;
        this.velocidad = 5.0;
        this.radio = radio;
        this.costoMagia = costoMagia;
        this.color = color;
        this.tickInicial = tickInicial;
        this.duracion = duracion;

        // Carga la imagen del hechizo y la escala al tamaño
adecuado
        this.imagenHechizo = new ImageIcon(rutaImagen)
            .getImage()
            .getScaledInstance((int)(radio * 2), (int)(radio *
```

```
2), Image.SCALE_SMOOTH);

        // Calcula la dirección del hechizo hacia el destino
        double dx = destinoX - xInicial;
        double dy = destinoY - yInicial;
        double distancia = Math.sqrt(dx * dx + dy * dy); //
Distancia entre origen y destino

        // Si la distancia es mayor que 0, normaliza la dirección
        if (distancia != 0) {
            this.direccionX = dx / distancia;
            this.direccionY = dy / distancia;
        } else {
            // Si la distancia es 0 (origen = destino), no se
mueve
            this.direccionX = 0;
            this.direccionY = 0;
        }
    }

    // Mueve el hechizo en la dirección previamente calculada
    public void mover() {
        x += direccionX * velocidad;
        y += direccionY * velocidad;
    }

    // Verifica si el hechizo sigue dentro de los límites de la
pantalla
    public boolean estaDentroDePantalla(int ancho, int alto) {
        return x >= 0 && x <= ancho && y >= 0 && y <= alto;
    }

    // Dibuja el hechizo en pantalla (imagen o círculo si la
imagen no está cargada)
    public void dibujar(Entorno entorno) {
        if (imagenHechizo != null) {
            entorno.dibujarImagen(imagenHechizo, x, y, 0);
        } else {
            entorno.dibujarCirculo(x, y, radio * 2, color); //
Dibuja un círculo si no hay imagen
        }
    }

    // Verifica si el hechizo todavía está activo en base al
tiempo actual
    public boolean estaActivo(int tiempoActual) {
        return (tiempoActual - tickInicial) <= duracion;
    }

    // Verifica si el hechizo está colisionando con un murciélago
    public boolean afectaA(Murcielago m) {
        double dx = m.getX() - x;
        double dy = m.getY() - y;
        double distancia = Math.sqrt(dx * dx + dy * dy);
        // Si la distancia entre los centros es menor que la suma
```

```
de los radios, hay colisión
        return distancia < (radio + m.getRadio());
    }

    // Métodos getter para acceder a variables privadas

    public int getCostoMagia() {
        return costoMagia;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public double getRadio() {
        return radio;
    }
}
```

BotonHechizo.java: Controla la interacción gráfica con hechizos.

```
package juego;

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

public class BotonHechizo {
    private String nombre;
    private int x;
    private int y;
    private Color color;
    private int costoMagia;
    private int ancho = 140;
    private int alto = 40;

    public BotonHechizo(String nombre, int x, int y, Color
color, int costoMagia) {
        this.nombre = nombre;
        this.x = x;
        this.y = y;
        this.color = color;
        this.costoMagia = costoMagia;
    }


    public void dibujar(Entorno entorno, boolean seleccionado) {
        Color fondo = seleccionado ? Color.YELLOW : color;
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, fondo);
```

```
        entorno.cambiarFont("Arial", 14, Color.BLACK);
        entorno.escribirTexto(nombre + " (" + costoMagia + ")",
x - ancho/2 + 5, y + 5);
    }

    // Verifica si el boton de hechizo fue presionado
    public boolean fuePresionado(int mouseX, int mouseY) {
        return mouseX >= x - ancho / 2 && mouseX <= x + ancho /
2 && mouseY >= y - alto / 2 && mouseY <= y + alto / 2;
    }

    // Getters
    public int getCostoMagia() {return costoMagia;}
    public String getNombre() {return nombre;}
    public int getX() { return x; }
    public int getY() { return y; }
}
```

Menu.java: Administra la parte visual del menú.

```
package juego; // Declara que esta clase pertenece al paquete
'juego'

import java.awt.Color; // Importa la clase Color para usar
colores
import entorno.Entorno; // Importa la clase Entorno para
interactuar con el motor gráfico

public class Menu {
    // Posición X del menú (en la pantalla)
    private int x;

    // Ancho y alto del menú
    private int ancho;
    private int alto;

    // Arreglo de botones que representan hechizos
    private BotonHechizo[] botones;

    // Índice del hechizo actualmente seleccionado (-1 si no hay
ninguno)
    private int hechizoSeleccionado;

    // Constructor del menú
    public Menu(int x, int ancho, int alto) {
        // Inicializa la posición y dimensiones del menú
        this.x = x;
        this.ancho = ancho;
        this.alto = alto;

        // Inicializa el arreglo de botones con capacidad para 3
```

```
hechizos
        this.botones = new BotonHechizo[3];

        // Calcula el centro horizontal del menú
        int centroX = x + ancho / 2;

        // Crea los botones de hechizos con sus nombres,
posiciones, colores y costos de magia
        botones[0] = new BotonHechizo("Disparo Magico", centroX,
150, Color.CYAN, 0); // Hechizo gratis
        botones[1] = new BotonHechizo("Bola De Fuego", centroX,
220, Color.RED, 20); // Cuesta 20 de magia
        botones[2] = new BotonHechizo("Agujero Negro", centroX,
290, Color.DARK_GRAY, 50); // Cuesta 50 de magia

        // Por defecto, no hay hechizo seleccionado
        hechizoSeleccionado = -1;
    }

    // Dibuja el menú completo en la pantalla
    public void dibujar(Entorno entorno, int vida, int magia, int
kills) {
        // Dibuja el fondo del menú como un rectángulo oscuro
        entorno.dibujarRectangulo(x + ancho / 2, alto / 2, ancho,
alto, 0, new Color(30, 30, 30));

        // Cambia la fuente para escribir texto informativo
        entorno.cambiarFont("Arial", 16, Color.WHITE);

        // Muestra la vida del jugador
        entorno.escribirTexto("VIDA: " + vida, x + 10, 30);

        // Muestra la cantidad de magia disponible
        entorno.escribirTexto("MAGIA: " + magia, x + 10, 60);

        // Muestra la cantidad de enemigos eliminados
        entorno.escribirTexto("ELIMINADOS: " + kills, x + 10,
90);

        // Dibuja todos los botones de hechizo
        for (int i = 0; i < botones.length; i++) {
            // Verifica si este botón está seleccionado
            boolean seleccionado = (i == hechizoSeleccionado);

            // Dibuja el botón con el estilo correspondiente
(resaltado si está seleccionado)
            botones[i].dibujar(entorno, seleccionado);
        }
    }

    // Actualiza qué hechizo está seleccionado cuando el jugador
hace clic
    public void actualizarSeleccion(Entorno entorno) {
        // Verifica si se presionó el botón izquierdo del mouse
        if (entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
```

```
            // Obtiene la posición del mouse
            int mouseX = entorno.mouseX();
            int mouseY = entorno.mouseY();

            // Recorre los botones para ver si alguno fue
presionado
            for (int i = 0; i < botones.length; i++) {
                // Si el botón fue presionado con el mouse, lo
selecciona
                if (botones[i].fuePresionado(mouseX, mouseY)) {
                    hechizoSeleccionado = i;
                }
            }
        }
    }

    // Devuelve el índice del hechizo actualmente seleccionado
    public int getHechizoSeleccionado() {
        return hechizoSeleccionado;
    }

    // Deselecciona cualquier hechizo
    public void deseleccionar() {
        hechizoSeleccionado = -1;
    }

    // Devuelve el botón de hechizo en la posición indicada
    public BotonHechizo getBoton(int indice) {
        return botones[indice];
    }

    // Verifica si el mouse está dentro del área del menú
    public boolean mouseEstaEnMenu(int mouseX) {
        return mouseX >= x;
    }
}
```

Murcielago.java: Enemigo que aparece en el entorno.

```
package juego;

import java.awt.Image;
import javax.swing.ImageIcon;
import entorno.Entorno;

// Clase que representa un murciélago enemigo
public class Murcielago {

     private double x;          // Posición horizontal del
murciélago
```

```
     private double y;          // Posición vertical del
murciélago
     private double velocidad;  // Velocidad de movimiento
     private double tamanio;    // Tamaño del murciélago (usado
para colisiones)
     private boolean vivo;      // Estado de vida: true = vivo,
false = muerto
     private Image imagen;      // Imagen visual del murciélago

     // Constructor: inicializa la posición, velocidad, estado y
carga la imagen
     public Murcielago(double xInicial, double yInicial, double
velocidad) {
          this.x = xInicial;
          this.y = yInicial;
          this.velocidad = velocidad;
          this.tamanio = 15;   // Define un tamaño base
          this.vivo = true;    // Al crearse, el murciélago está
vivo

          // Carga y escala la imagen del murciélago a 50x50
píxeles
          this.imagen = new ImageIcon("src/imagenes/murcielago
2.0.png")
                .getImage().getScaledInstance(50, 50,
Image.SCALE_SMOOTH);
     }

     // Método para dibujar el murciélago si está vivo
     public void dibujar(Entorno entorno) {
          if (vivo) {
               entorno.dibujarImagen(imagen, x, y, 0);
          }
     }

     // Método que hace que el murciélago se mueva hacia la
posición de Gondolf
     public void moverHacia(Gondolf gondolf) {
          if (!vivo) return; // Si está muerto, no se mueve

          // Calcula el vector de dirección desde el murciélago
hacia Gondolf
          double gondolfX = gondolf.getX() - x;
          double gondolfY = gondolf.getY() - y;

          // Calcula la distancia al personaje
          double distancia = Math.sqrt(gondolfX * gondolfX +
gondolfY * gondolfY);

          // Normaliza el vector y lo multiplica por la
velocidad para mover al murciélago
          if (distancia != 0) {
               x += (gondolfX / distancia) * velocidad;
               y += (gondolfY / distancia) * velocidad;
          }
```

```
	}

	// Verifica si colisiona con un punto dado (x, y) con cierto
radio (como un hechizo)
	public boolean colisionaCon(double x, double y, double
radio) {
		double dx = this.x - x;
		double dy = this.y - y;
		double distancia = Math.sqrt(dx * dx + dy * dy);
		return distancia < (this.tamanio / 2 + radio); //
Colisión circular
	}

	// Cambia el estado del murciélago a muerto
	public void morir() {
		this.vivo = false;
	}

	// Getters

	public boolean getVivo() {
		return vivo;
	}

	public double getX() {
		return x;
	}

	public double getY() {
		return y;
	}

	public double getRadio() {
		return tamanio / 2; // Retorna el radio, útil para
colisiones
	}
}
```

Pocion.java: Clase que define las pociones que sueltan los enemigos.

```
package juego;

// Importa clases para manejo de imágenes
import java.awt.Image;
import javax.swing.ImageIcon;

// Importa la clase Entorno
import entorno.Entorno;

public class Pocion {
```

```
    // Coordenadas de la poción en pantalla
    private double x;
    private double y;

    // Tiempo en que se creó la poción (en milisegundos)
    private int tiempoCreacion;

    // Indica si la poción es visible en pantalla
    private boolean visible;

    // Constante que define cuánto tiempo la poción permanece
visible (6 segundos)
    private final int DURACION_MS = 6000;

    // Radio de detección para saber si Gondolf tocó la poción
    private final int RADIO = 10;

    // Imagen de la poción
    private Image imagen;

    // Constructor: inicializa la posición, tiempo de creación,
visibilidad y la imagen
    public Pocion(double x, double y, int tiempoCreacion) {
        this.x = x;
        this.y = y;
        this.tiempoCreacion = tiempoCreacion;
        this.visible = true;

        // Carga la imagen desde archivo, la escala a 100x100
píxeles y la guarda
        this.imagen = new ImageIcon("src/imagenes/pocion de
vida.png")
                .getImage().getScaledInstance(100, 100,
Image.SCALE_SMOOTH);
    }

    // Dibuja la imagen de la poción si está visible
    public void dibujar(Entorno entorno) {
        if (visible) {
            entorno.dibujarImagen(imagen, x, y, 0); // 0 es la
rotación
        }
    }

    // Método que se llama constantemente para actualizar el
estado de la poción
    public void actualizar(int tiempoActual, Gondolf gondolf) {
        // Si no está visible, no hace nada
        if (!visible) return;

        // Si pasó más tiempo del permitido, la poción desaparece
        if (tiempoActual - tiempoCreacion > DURACION_MS) {
            visible = false;
            return;
        }
```

```
        // Calcula la distancia entre Gondolf y la poción
        double dx = gondolf.getX() - x;
        double dy = gondolf.getY() - y;
        double distancia = Math.sqrt(dx * dx + dy * dy);

        // Si Gondolf está suficientemente cerca (colisión), le
da vida
        if (distancia < RADIO + 20) {
            int vida = gondolf.getVida();     // Vida actual de
Gondolf
            int maxVida = 100;                // Vida máxima
posible
            int incremento = (int)(maxVida * 0.1); // 10% de vida
máxima

            // Si Gondolf no tiene vida completa, le aumenta la
vida
            if (vida < maxVida) {
                gondolf.incrementaVida(incremento);
            }

            // Una vez recogida, la poción desaparece
            visible = false;
        }
    }

    // Devuelve si la poción está visible o no (útil para control
desde otras clases)
    public boolean estaVisible() {
        return visible;
    }
}
///La clase Pocion se encarga de mostrar una poción en el juego
por un tiempo limitado.
///Si Gondolf se acerca, la poción le restaura vida y luego
desaparece.
```

Roca.java: Obstáculo en el escenario.

```
package juego;

import java.awt.Image;

import entorno.Entorno;
import entorno.Herramientas;

public class Roca {
    private double x;
    private double y;
```

```
    private double ancho;
    private double alto;
    private Image Rocas;

    public Roca(double x, double y) {
        this.x = x;
        this.y = y;
        this.ancho = 30;
        this.alto = 50;
        this.Rocas =
Herramientas.cargarImagen("Imagenes/piedras.png");
    }

    // Dibuja la roca en pantalla
    public void dibujar(Entorno entorno) {
      entorno.dibujarImagen(Rocas, x, y, 0, 0.1);

    }

    // Verifica si Gondolf choca con esta roca
    public boolean colisionaCon(Gondolf gondolf) {
        double gx = gondolf.getX();
        double gy = gondolf.getY();
        double gancho = gondolf.getAncho();
        double galto = gondolf.getAlto();

        // Verificamos colisión de rectángulo con rectángulo
        return Math.abs(this.x - gx) < (this.ancho + gancho) / 2
&&
               Math.abs(this.y - gy) < (this.alto + galto) / 2;
    }

    // Getters útiles si los necesitás
    public double getX() { return x; }
    public double getY() { return y; }
    public double getAncho() { return ancho; }
    public double getAlto() { return alto; }
}
```

- **Conclusiones:**

  El trabajo fue un gran desafío para organizarnos como grupo, por suerte todos nos pusimos de acuerdo en un meet que hicimos antes de arrancar en el cual decidimos cómo encarar el trabajo, esto nos sirvió para entender cómo optimizar el tiempo de cada uno y también para no pisarnos entre nosotros.
  Una lección que aprendimos como grupo es que no solo es importante hacer bien la parte del código sino que comunicarse con los compañeros a la hora de subir algo o modificar algo para poder intercambiar opiniones o consejos mutuamente.
  Otra conclusión fue que es muy importante comenzar desde una buena base tanto en la comunicación del grupo, como en la creación del código. Ya que si no se

comienza por una buena base, al avanzar en el código aparecen errores inesperados que atrasan todo el trabajo.
También hablando del resultado en sí, estamos muy contentos por el trabajo que obtuvimos y como pudimos resolver las situaciones que se nos pusieron en el camino. Por ejemplo, tuvimos un error de un “git force” que lo resolvimos en grupo con una copia local de otro compañero.