Universidad Nacional de General Sarmiento

Trabajo Práctico Programación 1

Docentes: César Niveyro - Nahuel Sauma

Comisión: 07

Grupo: 4

Integrantes:

Ryan Reinoso (43.663.279)
Reinosoryan09@gmail.com

Martina Quevedo (46.915.452)
quevedomartina1705@gmail.com

Guadalupe Molinillo (46.942.084)
guadalupemolinillo@gmail.com

El Camino de Gondolf

Introducción

El trabajo “El camino de Gondolf” se trata de programar y diseñar un videojuego donde se simula un ambiente de época donde un mago llamado Gondolf debe eliminar a los murciélagos que salen de muchos lugares y que intentan acabar con él.
Gondolf tratará de eliminarlos con sus hechizos mágicos que cuando lo utiliza bajará su maná y también deberá tener cuidado de no chocarse con las piedras a su alrededor que interferirá en su camino.

Este juego no solo busca cumplir con una serie de requisitos técnicos sino que intenta proporcionar la mejor experiencia para el jugador.
Enfocándonos al desarrollo del juego, este contiene implementaciones de clases de diferentes puntos claves para el trabajo, creaciones del personaje, los enemigos, los hechizos, pociones, un jefe, etc. Además de ampliación de imágenes para estas y colisiones entre ellas.
Para poder ganar Gondolf deberá acabar con los enemigos y con el jefe final.

Algunos problemas que encontramos bajo nuestro trabajo fueron:

- Comprender y adaptarnos a los requerimientos del trabajo práctico, que en ciertos casos no resultaban del todo claros o intuitivos.

- Coordinar el trabajo grupal a través de GitHub, una herramienta que varios integrantes no habían utilizado previamente.

- Encontrar momentos adecuados para reunirnos y avanzar en el proyecto, teniendo en cuenta la carga de otras materias del curso.

- Incorporar imágenes adecuadas que aportaran coherencia visual y mejoraran la experiencia general del juego.

Descripción de las clases:

- **Clase Gondolf :** La clase `Gondolf` representa al personaje principal del juego, contiene todo los valores necesarios (Posición, imágenes, vida, mana, etc) para poder crearlo.

```java
public class Gondolf {
    private int x; // Posición x de Gondolf en la pantalla
    private int y; // Posición y de Gondolf en la pantalla
    private int alto; // Alto de Gondolf
    private int ancho; // Ancho de Gondolf
    private int vida; // Vida de Gondolf
    private int mana; // Mana de gondolf
    private Image imagenDerecha; // Imagen del personaje a la derecha
    private Image imagenIzquierda; // Imagen del personaje a la izquierda
    private Image imagenActual; // Imagen del personaje guardada
    private Image imagenArriba; // Imagen del personaje cuando va a arriba
    private Image imagenAbajo; // Imagen del personaje cuando va a abajo


    public Gondolf (int x , int y, int alto, int ancho, int vida, int mana) { // Inicializamos a Gondolf
        this.x = x;
        this.y = y;
        this.alto = alto;
        this.ancho = ancho;
        this.vida = vida;
        this.mana = mana;
        this.imagenDerecha = Toolkit.getDefaultToolkit().getImage("Imagenes/Gondolfder.png");
        this.imagenIzquierda = Toolkit.getDefaultToolkit().getImage("Imagenes/Gondolfizq.png");
        this.imagenAbajo = Toolkit.getDefaultToolkit().getImage("Imagenes/GondolfNormaloAbajo.png");
        this.imagenArriba = Toolkit.getDefaultToolkit().getImage("Imagenes/Gondolfarriba.png");


        this.imagenActual = imagenAbajo;   // Imagen inicial por defecto

    }

    public void dibujar(Entorno entorno) { // Metodo que dibuja a Gondolf en pantalla en la posición x,y y escala
        entorno.dibujarImagen(imagenActual, x, y, 0, 1.0);
    }
```

```
}

public void dibujar(Entorno entorno) { // Metodo que dibuja a Gondolf en pantalla en la posición x,y y escala
    entorno.dibujarImagen(imagenActual, x, y, 0, 1.0);
}


    public int restarvida(int v) { // Metodo que resta cantidad v de vida.
        if(vida > 0) {
        vida = vida - v;
        } else {
        }

    return vida;

}
    public int sumarvida() { // Metodo que suma cantidad 10 de vida.
        if(vida > 0 && vida < 100) {
        vida = vida + 10;
        }
    return vida;
}

    public int restarMana(int j) { // Metodo que resta cantidad J de Maná
        if(mana > 0) {
        mana = mana - j;

        }
    return mana;

}
    public int sumarMana() { // Metodo que suma cantidad 10 de Maná.
        if(mana >= 0 && mana <= 100) {
        mana = mana + 10;

        }
    return mana;
}

public void MoverIzq() { // Metodo que mueve a Gondfolf -3 en X .
    this.x -=3 ;
    this.imagenActual = imagenIzquierda;
}
```

```
public void MoverIzq() { // Metodo que mueve a Gondfolf -3 en X .
    this.x -=3 ;
    this.imagenActual = imagenIzquierda;
}
public void MoverDer() { // Metodo que mueve a Gondfolf +3 en X .
    this.x +=3 ;
    this.imagenActual = imagenDerecha;
}
public void MoverArriba() { // Metodo que mueve a Gondfolf -3 en Y .
    this.y -=3 ;
    this.imagenActual = imagenArriba; // Utiliza la imagen asignada
}
public void MoverAbajo() { // Metodo que mueve a Gondfolf +3 en Y .
    this.y +=3 ;
    this.imagenActual = imagenAbajo;
}

public boolean colisionaPorDerecha(Entorno entorno) { // Metodo boolean que retorna True si Gondolf toca el borde derecho - El menú.
    return this.x + this.ancho/2 >= entorno.ancho()-175;
}

public boolean colisionaPorIzquierda(Entorno entorno) { // Metodo boolean que retorna True si Gondolf toca el borde izquierdo
    return this.x - this.ancho/2 <= 0;
}

public boolean colisionaPorArriba(Entorno entorno) {  // Metodo boolean que retorna True si Gondolf toca el borde de Arriba
    return this.y - this.alto / 2 <= 5;
}

public boolean colisionaPorAbajo(Entorno entorno) { // Metodo boolean que retorna True si Gondolf toca el borde de Abajo
    return this.y + this.alto / 2 >= entorno.alto()-5;
}

public boolean colisionaConPiedra(int dx, int dy, Piedra[] piedras) {
    for (Piedra p :  piedras ) {
        // Calculamos dónde estaría Gondolf si se moviera
        int futuroX = getX() + dx;
        int futuroY = getY() + dy;

        // Verificamos si en esa posición estaría tocando una piedra
        if (Math.abs(futuroX - p.getX()) < (getAncho() / 2 + p.getAncho() / 2) &&
            Math.abs(futuroY - p.getY()) < (getAlto() / 2 + p.getAlto() / 2)) {
            return true; // Hay colisión
```

Variable de instancia:

- `int x,y,ancho,alto:` Coordenadas X e Y en pantalla, ancho y altura que tendrá el personaje.
- `int vida, mana:` Vida del personaje y mana del mismo.
- `Image Imagen:` Representa las imágenes que tendrá el personaje según a donde se esté moviendo.

Constructores

Public Gondolf (int x , int y, int alto, int ancho, int vida, int mana):
Le solicitamos esos valores para crear a gondolf.

Métodos:

`public void dibujar (Entorno entorno)`: Se encarga de dibujar a Gondolf en la pantalla

`public int restarvida(int v)`**:** Se encarga de restar vida al personaje

`public int sumarvida()`**:** Se encarga de sumar vida al personaje

`public int sumarMana()` Se encarga de sumar maná al personaje

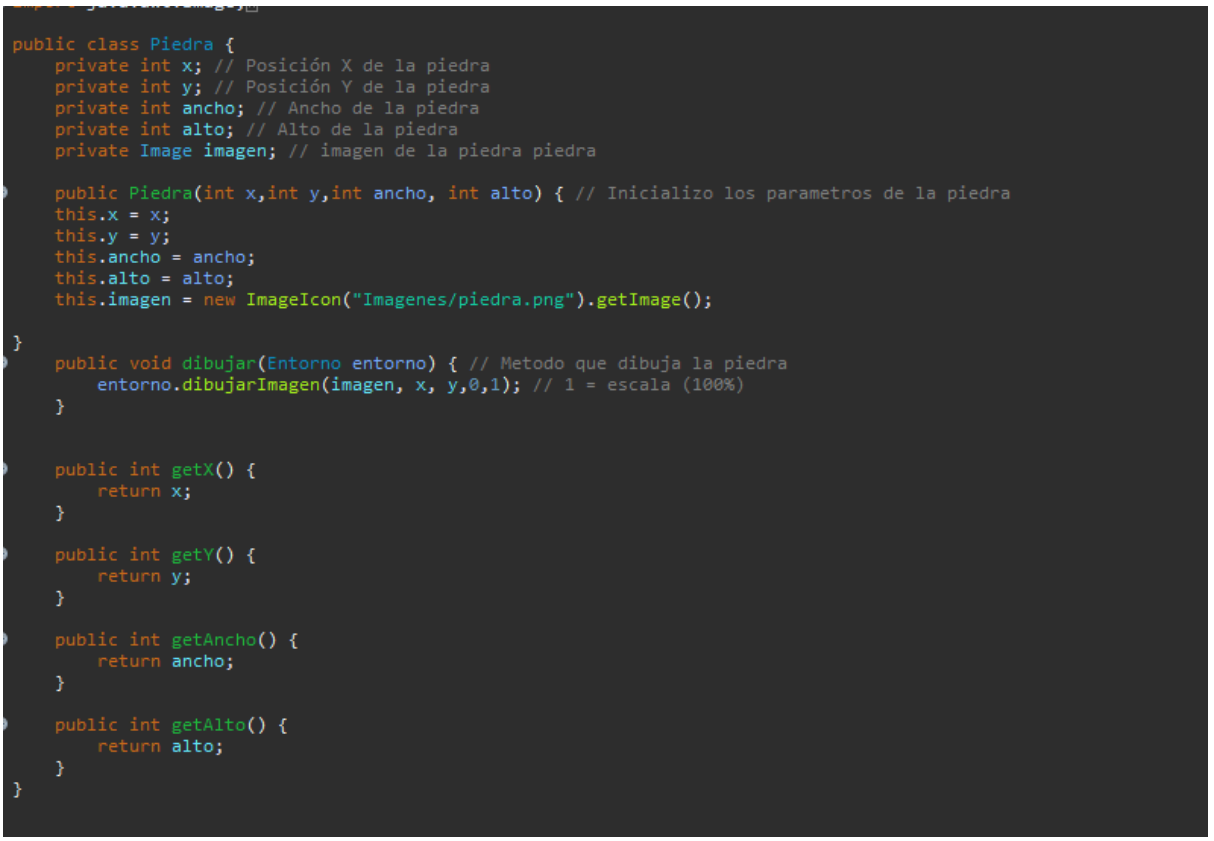
`public void Mover*()`: Mueve al personaje a la dirección que cambies el *.

`public boolean colisionaPor*(Entorno entorno)`**:** Devuelve True si el personaje colisiona con la posición * que pongas.

`public boolean colisionaConPiedra(int dx, int dy, Piedra[] piedras)`**:** Devuelve True si el personaje colisiona con la piedra.

- **Clase Piedra:** La clase `Piedra` representa a las piedras que se dibujan en la pantalla, contiene sólo los parámetros de X e Y, un alto y un ancho.

-



```java
public class Piedra {
    private int x; // Posición X de la piedra
    private int y; // Posición Y de la piedra
    private int ancho; // Ancho de la piedra
    private int alto; // Alto de la piedra
    private Image imagen; // imagen de la piedra piedra

    public Piedra(int x,int y,int ancho, int alto) { // Inicializo los parametros de la piedra
    this.x = x;
    this.y = y;
    this.ancho = ancho;
    this.alto = alto;
    this.imagen = new ImageIcon("Imagenes/piedra.png").getImage();

}
    public void dibujar(Entorno entorno) { // Metodo que dibuja la piedra
        entorno.dibujarImagen(imagen, x, y,0,1); // 1 = escala (100%)
    }


    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getAncho() {
        return ancho;
    }

    public int getAlto() {
        return alto;
    }
}
```

Variable de instancia:

- `int x,y,ancho,alto:` Coordenadas X e Y en pantalla, ancho y altura que tendrá la piedra

  .**Constructor :**

  public Piedra(int x,int y,int ancho, int alto): Le otorgamos lo anteriormente dicho.

  Método:

  public void dibujar(Entorno entorno): Método que dibuja la piedra en pantalla.

- **Clase Jefe:** La clase `Jefe` representa al jefe final del juego. Contiene toda la información necesaria para crearlo, moverlo y detectar colisiones.

```java
public class Jefe {
    private int x; // Posición x del jefe
    private int y; // Posición y del jefe
    private int ancho; // Ancho del Jefe
    private int alto; // Alto del personaje
    private int direccioninicial; // Dirección donde comienza a moverse el jefe
    private Image imagen; //Imagen del Jefe
    private int vida; // Vida el Jefe


    public Jefe(int x, int y, int ancho, int alto) { // Inicializamos al Jefe
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.direccioninicial = 1;
        this.imagen = new ImageIcon("Imagenes/golbat.gif").getImage(); // Asignamos la imagen al jefe
        this.vida = 100; // Asignamos la cantidad de vida al jefe


    }

    public void dibujar(Entorno entorno) { // Metodo para dibujar al jefe
        entorno.dibujarImagen(imagen, x, y, 0);
    }

    public void mover(Entorno entorno) { // Metodo para mover al jefe
        if (this.y < 50) {
            this.y += 2; // Primero baja hasta y == 50
        } else {
            // Cuando ya bajó, se mueve en la dirección actual
            this.x += 3 * direccioninicial;

            // Revisa colisiones con los bordes
            if (colisionaPorDerecha(entorno)) {
                direccioninicial = -1; // Cambia a izquierda
            } else if (colisionaPorIzquierda(entorno)) {
                direccioninicial = 1; // Cambia a derecha
            }
        }
    }
```

```java
    public boolean colisionaPorDerecha(Entorno entorno) { //Metodo que revisa que colisione por derecha - el menu
        return this.x + this.ancho/2 >= entorno.ancho()-175;
    }

    public boolean colisionaPorIzquierda(Entorno entorno) { //Metodo que revisa que colisione por izquierda
        return this.x - this.ancho/2 <= 0;
    }

    public boolean colisionaCon(int otroX, int otroY, int margen) { // Metodo que revisa que colisione con algun objeto
        return Math.abs(this.x - otroX) < margen && Math.abs(this.y - otroY) < margen;
    }

    public int restarvida(int a) { // se le resta la cantidad a de vida al jefe
        if(vida > 0) {
        vida = vida - a;
        }

    return vida;
```

Variable de instancia:

- `int x,y,ancho,alto:` Coordenadas en pantalla y dimensiones del jefe.

- `int direccioninicial`**:** Determina la dirección en la que comienza a moverse el jefe (1: derecha, -1: izquierda).

- `Image imagen`**:** Imagen del jefe que se muestra en pantalla.

- `int vida`**:** Vida actual del jefe.

  Constructor:

  `public Jefe(int x, int y, int ancho, int alto)` : Inicializa al jefe con su posición, tamaño, imagen y vida predeterminada (100 puntos). También establece que inicialmente se moverá hacia la derecha.

  Métodos:

  `public void dibujar(Entorno entorno)`: Se encarga de mostrar al jefe en pantalla usando su imagen.

  `public void mover(Entorno entorno)`: Mueve al jefe. Primero baja hasta una cierta posición (y = 50), y luego se desplaza lateralmente. Si choca con los bordes izquierdo o derecho de la pantalla, cambia de dirección.

  `public boolean colisionaPorDerecha(Entorno entorno)`: Devuelve `true` si el jefe colisiona con el borde derecho de la pantalla.

  `public boolean colisionaPorIzquierda(Entorno entorno)`: Devuelve `true` si el jefe colisiona con el borde izquierdo de la pantalla.

  `public boolean colisionaCon(int otroX, int otroY, int margen)`: Devuelve `true` si el jefe colisiona con otro objeto, usando una tolerancia definida por el margen.

  `public int restarvida(int a)`: Resta una cantidad `a` de vida al jefe si su vida es mayor a cero. Devuelve el valor actualizado de la vida.

- **Clase Disparo:** La clase `Disparo` representa los disparos lanzados por el jefe. Contiene la información necesaria para posicionarlos, moverlos, dibujarlos y detectar colisiones.

```java
public class Disparo {
    private int x; // Posición x del Disparo en la pantalla
    private int y; // Posición Y del Disparo en la pantalla
    private int alto; // Alto del Disparo
    private int ancho; // Ancho del disparo
    private Image image; // Importa una imagen

    public Disparo(int x,int y, int alto, int ancho) {  //Parametros para inicializar el disparo
        this.x = x;
        this.y = y;
        this.alto = alto;
        this.ancho = ancho;
        this.image = Toolkit.getDefaultToolkit().getImage("Imagenes/Disparo.png");
    }


    public void mover() { // Metodo que mueve el disparo
        this.y += 3;

    }
     public void dibujar (Entorno entorno) { // Metodo que dibuja el Disparo en la pantalla
         entorno.dibujarImagen(image, x, y, 0, 1);
     }

     public boolean ColisionaCon(int otroX, int otroY, int margen) { // Metodo para verificar si colisiona con otro objeto
            return Math.abs(this.x - otroX) < margen && Math.abs(this.y - otroY) < margen;
        }

    public int getX() {
        return x;
    }


    public int getY() {
        return y;
    }


    public int getAlto() {
        return alto;
```

Variables de instancia:

- **`int x, y`:** Coordenadas del disparo en la pantalla.
- **`int alto, ancho`:** Dimensiones del disparo.
- **`Image image`:** Imagen que representa visualmente al disparo.

Constructor:

public Disparo(int x, int y, int alto, int ancho): Inicializa un disparo con su posición y tamaño, y le asigna una imagen cargada desde la carpeta correspondiente.

Métodos:

`public void mover()`: Mueve el disparo hacia abajo aumentando su posición `y`.

`public void dibujar(Entorno entorno)`: Dibuja el disparo en pantalla utilizando su imagen y posición actual.

`public boolean ColisionaCon(int otroX, int otroY, int margen)`: Devuelve `true` si el disparo colisiona con otro objeto dentro de un margen de tolerancia.

`public int getX()`: Devuelve la coordenada `x` del disparo.

`public int getY()`: Devuelve la coordenada `y` del disparo.

`public int getAlto()`: Devuelve la altura del disparo.

`public int getAncho()`: Devuelve el ancho del disparo

- **Clase Boton:** La clase `Boton` representa un botón gráfico que se puede dibujar en pantalla. Tiene atributos para definir su posición, tamaño, colores y texto. Permite modificar su apariencia, marcarlo como seleccionado y detectar si el usuario hizo clic sobre él.

-
```java
public class Boton {

    // Coordenadas del centro del botón
    private int x;
    private int y;
 // Dimensiones del botón
    private int ancho;
    private int alto;

    private Color colorFondo;
    private String texto;
    private String fuente;
    private int tamañoFuente;
    private Color colorTexto;
    private boolean seleccionado;
    private static final Color GRIS_OSCURO = new Color(64, 64, 64); // Color gris oscuro usado cuando el botón está seleccionado

    // Constructor: inicializa posición, tamaño, color y valores por defecto del texto
    public Boton(int x, int y, int ancho, int alto, Color colorFondo) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.colorFondo = colorFondo;
        this.texto = ""; // Texto vacío por defecto
        this.fuente = "Impact";
        this.tamañoFuente = 17;
        this.colorTexto = Color.WHITE;
        this.seleccionado = false; // No seleccionado al iniciar
    }

 // Método para cambiar el texto del botón
    public void setTexto(String texto) {
        this.texto = texto;
```

-
```java
    }

  // Método para cambiar fuente, tamaño y color del texto
    public void setFuente(String fuente, int tamaño, Color colorTexto) {
        this.fuente = fuente;
        this.tamañoFuente = tamaño;
        this.colorTexto = colorTexto;
    }

  // Método para marcar o desmarcar el botón como seleccionado
    public void setSeleccionado(boolean sel) {
        this.seleccionado = sel;
    }

  // Devuelve si el botón está seleccionado o no
    public boolean isSeleccionado() {
        return this.seleccionado;
    }

  // Dibuja el botón en pantalla
    public void dibujar(Entorno entorno) {
        Color colorParaDibujar = seleccionado ? GRIS_OSCURO : colorFondo; // Usa gris oscuro si está seleccionado, si no, su color de fondo normal
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, colorParaDibujar); // Dibuja el rectángulo del botón
        entorno.cambiarFont(fuente, tamañoFuente, colorTexto);  // Cambia la fuente para el texto

     // Calcula la posición horizontal aproximada del texto para centrarlo
        int textoAnchoAprox = texto.length() * (tamañoFuente / 2);
        int textoX = x - textoAnchoAprox / 2;
        int textoY = y + (tamañoFuente / 3); // Calcula la posición vertical del texto

        entorno.escribirTexto(texto, textoX, textoY); // Dibuja el texto en el botón
    }

  // Verifica si un punto (px, py) está dentro del área del botón
    public boolean estaDentro(int px, int py) {
        // Coordenadas de los bordes del botón
        int left = x - ancho / 2;
        int right = x + ancho / 2;
        int top = y - alto / 2;
        int bottom = y + alto / 2;
```

-

Variables de instancia:

- `int x`: Coordenada X del centro del botón.
- `int y`: Coordenada Y del centro del botón.
- `int ancho`: Ancho del botón.
- `int alto`: Alto del botón.

- `Color colorFondo`: Color de fondo del botón cuando no está seleccionado.
- `String texto`: Texto que se muestra dentro del botón.
- `String fuente`: Nombre de la fuente usada para el texto del botón.
- `int tamañoFuente`: Tamaño de la fuente del texto.
- `Color colorTexto`: Color del texto que se muestra en el botón.
- `boolean seleccionado`: Indica si el botón está seleccionado (resaltado o activo).
- `static final Color GRIS_OSCURO`: Color constante usado como fondo cuando el botón está seleccionado.

Métodos:

- `Boton(int x, int y, int ancho, int alto, Color colorFondo)`: Constructor que inicializa la posición, tamaño, color de fondo y configura valores por defecto para el texto y la fuente.
- `void setTexto(String texto)`: Permite establecer el texto que se mostrará en el botón.
- `void setFuente(String fuente, int tamaño, Color colorTexto)`: Permite cambiar el tipo de fuente, su tamaño y el color del texto.
- `void setSeleccionado(boolean sel)`: Establece si el botón está seleccionado o no.
- `boolean isSeleccionado()`: Devuelve `true` si el botón está actualmente seleccionado, `false` en caso contrario.
- `void dibujar(Entorno entorno)`: Dibuja el botón en pantalla con su color, texto y estilo correspondiente. Si el botón está seleccionado,

se usa un color más oscuro como fondo.

- `boolean estaDentro(int px, int py)`: Verifica si un punto (por ejemplo, las coordenadas del mouse) está dentro del área del botón.

- `int getX()`: Devuelve la coordenada X del centro del botón.

- `int getY()`: Devuelve la coordenada Y del centro del botón.

- **Clase Menu:** La clase `Menu` representa un componente gráfico rectangular que actúa, en este caso, como la visualización de los botones del menú dentro de la pantalla del juego. Permite definir su posición, dimensiones, color de fondo y texto personalizado.

```java
public class Menu {

    // Coordenadas del centro del menú en pantalla
    private int x;
    private int y;
    // Dimensiones del menú
    private int ancho;
    private int alto;

    private Color color;
    private String texto;
    private String fuente;
    private int tamañoFuente;
    private Color colorTexto;
 // Colores predefinidos para usar en los menús (tipo madera)
    private static final Color brown = new Color(90, 45, 15);
    private static final Color light_brown = new Color (205, 133, 63);

 // Constructor: recibe posición, tamaño y color. Inicializa valores por defecto del texto.
    public Menu (int x , int y, int alto, int ancho, Color color) {
        this.x = x; // Posición horizontal del centro del menú
        this.y = y; // Posición vertical del centro del menú
        this.alto = alto; // Altura del menú (botones)
        this.ancho = ancho; // Ancho del menú (botones)
        this.color = color; // Color de fondo del menú (botones)
        this.texto = ""; // Texto vacío por defecto
        this.fuente = "Showcard Gothic"; // Fuente predeterminada
        this.tamañoFuente = 17; // Tamaño de fuente por defecto
        this.colorTexto = Color.WHITE; // Texto blanco por defecto

    }
    // Setter para cambiar el texto que se muestra
    public void setTexto(String texto) {
        this.texto= texto;
    }
    // Setter para cambiar la fuente, tamaño y color del texto
    public void setFuente(String fuente, int tamaño, Color colorTexto) {
        this.fuente = fuente;
        this.tamañoFuente = tamaño;
```

```
    // Getter estático para obtener el color marrón oscuro
    public static Color getBrown() {
        return brown;
    }
    // Getter estático para obtener el marrón claro
    public static Color getlight_brown() {
        return light_brown;
    }
    // Dibuja el menú en pantalla
    public void dibujar(Entorno entorno) {
        entorno.dibujarRectangulo(x, y, ancho, alto, 0, color); // Dibuja el rectángulo del menú (botones)
        entorno.cambiarFont(fuente, tamañoFuente, colorTexto); // Configura la fuente y color del texto
        // Calcula la posición horizontal del texto para centrarlo
        int textoAnchoAprox = texto.length() * (tamañoFuente / 2);
        int textoX = x - textoAnchoAprox / 2;
     // Calcula la posición vertical del texto
        int textoY = y + (tamañoFuente / 3); // Escribe el texto en pantalla

        entorno.escribirTexto(texto, textoX, textoY);
    }
// Getters para acceder a la posición y tamaño del menú
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
    public int getAlto() {
        return alto;
    }

    public int getAncho() {
        return ancho;
    }
```

Variables de instancia:

- `int x`: Coordenada X del centro del menú.
- `int y`: Coordenada Y del centro del menú.
- `int ancho`: Ancho total del menú.
- `int alto`: Alto total del menú.
- `Color color`: Color de fondo del menú.
- `String texto`: Texto que se muestra en el menú.
- `String fuente`: Fuente del texto.
- `int tamañoFuente`: Tamaño del texto.
- `Color colorTexto`: Color del texto.

- `static final Color brown`: Color marrón oscuro predefinido..
- `static final Color light_brown`: Color marrón claro predefinido.

Métodos:

- `Menu(int x, int y, int alto, int ancho, Color color)`: Constructor. Inicializa la posición, dimensiones y color del menú. También configura valores por defecto para el texto, la fuente y el color del texto.
- `void setTexto(String texto)`: Permite establecer el texto que se mostrará en el menú.
- `void setFuente(String fuente, int tamaño, Color colorTexto)`: Cambia la fuente del texto, su tamaño y color.
- `static Color getBrown()`: Devuelve el color marrón oscuro predefinido.
- `static Color getlight_brown()`: Devuelve el color marrón claro predefinido.
- `void dibujar(Entorno entorno)`: Dibuja el menú como un rectángulo con el texto centrado en su interior, utilizando las propiedades actuales de fuente, tamaño y color.
- `int getX()`: Devuelve la coordenada X del menú.
- `int getY()`: Devuelve la coordenada Y del menú.
- `int getAlto()`: Devuelve la altura del menú.
- `int getAncho()`: Devuelve el ancho del menú.

- **Clase Hechizos:** La clase `Hechizos` representa un proyectil mágico lanzado por el jugador. Está definido por su posición, tamaño (diámetro) y color. Puede dibujarse en pantalla como un círculo y tiene métodos para detectar colisiones con enemigos comunes o con el jefe final. También permite modificar sus propiedades mediante getters y setters.

```java
public class Hechizos {
    // Posición del centro del hechizo
    private double x;
    private double y;

    private double diametro;
    private Color color;

 // Constructor: recibe origen y destino, diámetro y color
    public Hechizos(int origenX, int origenY, int destinoX, int destinoY, int diametro, Color color) {
        this.x = origenX; // Se guarda la coordenada X del punto de origen
        this.y = origenY; // Se guarda la coordenada Y del punto de origen
        this.diametro = diametro; // Se guarda el diámetro del hechizo
        this.color = color; // Se guarda el color del hechizo

    }
 // Dibuja el hechizo como un círculo en pantalla usando la librería Entorno
    public void dibujar(Entorno entorno) {
        entorno.dibujarCirculo(x, y, diametro, color);
    }
 // Verifica si el hechizo colisiona con un enemigo común
    public boolean colisionaCon(Enemigo enemigo) {
        // Calcula distancia entre el hechizo y el enemigo
        double dx = this.x - enemigo.getX();
        double dy = this.y - enemigo.getY();
        double distancia = Math.sqrt(dx * dx + dy * dy);
     // Retorna true si la distancia entre centros es menor que la suma de los radios
        return distancia < this.diametro / 2 + 20; // 20 es un radio aproximado del murciélago
    }
 // Verifica si el hechizo colisiona con el jefe
    public boolean colisionaCon(Jefe jefe) {
        // Calcula distancia entre el hechizo y el jefe
        double dx = this.x - jefe.getX();
        double dy = this.y - jefe.getY();
        double distancia = Math.sqrt(dx * dx + dy * dy);
     // Retorna true si hay colisión con el jefe
        return distancia < this.diametro / 2 + 20; // 20 es un radio aproximado del jefe
    }
```

-

Variables de instancia:

- `double x`: Coordenada X del centro del hechizo.
- `double y`: Coordenada Y del centro del hechizo.
- `double diametro`: Diámetro del hechizo (tamaño del círculo).
- `Color color`: Color visual del hechizo.

Constructor:

- `Hechizos(int origenX, int origenY, int destinoX, int destinoY, int diametro, Color color)`:

Constructor. Inicializa la posición de origen, el diámetro y el color del hechizo.

- **Métodos:**

- `void dibujar(Entorno entorno)`:
  Dibuja el hechizo como un círculo en pantalla usando las coordenadas, tamaño y color actuales.

- `boolean colisionaCon(Enemigo enemigo)`:
  Verifica si el hechizo colisiona con un enemigo común. Calcula la distancia entre los centros y devuelve `true` si es menor a la suma de los radios.

- `boolean colisionaCon(Jefe jefe)`:
  Verifica si el hechizo colisiona con el jefe final, usando la misma lógica de distancia que con el enemigo común.

- `double getX()` / `void setX(double x)`:
  Getter y setter para la coordenada X del hechizo.

- `double getY()` / `void setY(double y)`:
  Getter y setter para la coordenada Y del hechizo.

- `double getDiametro()` / `void setDiametro(double diametro)`:
  Getter y setter para el diámetro del hechizo.

- `Color getColor()` / `void setColor(Color color)`:
  Getter y setter para el color del hechizo.

- **Clase Juego:** Contiene todo lo necesario para que el juego funcione, es el main del proyecto y donde todos los objetos son creados.
  En esta clase existe un metodo Tick que se va actualizando por segundo en pantalla para que se pueda ver el movimiento de todos los objetos interactuando entre ellos.

- **Clase Enemigo:**

```java
public class Enemigo {
    private int x;
    private int y;

    private Image imagenes;

    public Enemigo(int x, int y, int ancho, int alto) {
        this.x = x;
        this.y = y;
        this.imagenes = new ImageIcon("Imagenes/mmurcielago.gif").getImage();

    }

    // Movimiento hacia el objetivo (Gondolf)
    public void moverHacia(int objetivoX, int objetivoY) {
        double dx = objetivoX - this.x;
        double dy = objetivoY - this.y;
        double distancia = Math.sqrt(dx * dx + dy * dy); // Distancia total al objetivo

        if (distancia > 0) {
            this.x += (int)(dx / distancia * 2); // si distancia es mayor a 0 se dirige hacia el
            this.y += (int)(dy / distancia * 2);
        }
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imagenes, x, y, 0);
    }


    // Método para detectar si colisiona con el jugador (Gondolf)
    public boolean colisionaCon(int otroX, int otroY, int margen) {
        return Math.abs(this.x - otroX) < margen && Math.abs(this.y - otroY) < margen; // Colisión por proximidad (radio)
    }

 // Getters para obtener la posición actual del murciélago
    public int getX() { return x; }
    public int getY() { return y; }

    public static Enemigo generarMurcielagoAleatorio() {
        int lado = (int)(Math.random() * 4); // Elige un borde al azar (0 a 3)
        int x = 0;
        int y = 0;
```

```java
    public static Enemigo generarMurcielagoAleatorio() {
        int lado = (int)(Math.random() * 4); // Elige un borde al azar (0 a 3)
        int x = 0;
        int y = 0;

      // Define coordenadas dependiendo del lado
        switch (lado) {
            case 0: x = (int)(Math.random() * 625); y = -20; break;       // arriba
            case 1: x = 620; y = (int)(Math.random() * 600); break;       // derecha
            case 2: x = (int)(Math.random() * 625); y = 620; break;       // abajo
            case 3: x = -20; y = (int)(Math.random() * 600); break;       // izquierda
        }

        return new Enemigo(x, y, 20, 20);
    }


}
```

-

Esta clase representa a un enemigo del juego, específicamente un murciélago, que se mueve en dirección al personaje principal (`Gondolf`) y puede detectar colisiones con él. También se encarga de su aparición en pantalla y su posicionamiento inicial.

Variables de instancia

- **`private int x;`**
  Representa la posición horizontal (coordenada X) del enemigo en la

pantalla.

- **`private int y;`**
  Representa la posición vertical (coordenada Y) del enemigo en la pantalla.

- **`private Image imagenes;`**
  Almacena la imagen gráfica del murciélago que se usará para dibujarlo en pantalla. Se carga desde un archivo GIF ubicado en `"Imagenes/mmurcielago.gif"`.

Constructores

- **`public Enemigo(int x, int y, int ancho, int alto)`**
  Inicializa un nuevo enemigo en la posición `(x, y)` y carga su imagen.

Métodos

- **`public void moverHacia(int objetivoX, int objetivoY)`**
  Mueve al enemigo en dirección a un objetivo (usualmente el jugador). Calcula la dirección usando el vector entre el enemigo y el objetivo, y lo traslada con una velocidad constante de 2 píxeles por fotograma.

- **`public void dibujar(Entorno entorno)`**
  Dibuja al enemigo en pantalla usando su imagen, en la posición actual `(x, y)`, con una rotación de 0 grados.

- **`public boolean colisionaCon(int otroX, int otroY, int margen)`**
  Verifica si el enemigo está lo suficientemente cerca del jugador como para considerarse una colisión. Esto se determina comparando las distancias entre las coordenadas `x` e `y` del enemigo y del jugador con un margen dado.

- **`public int getX()`**
  Devuelve la posición horizontal actual del enemigo.

- **`public int getY()`**
  Devuelve la posición vertical actual del enemigo.

- **`public static Enemigo generarMurcielagoAleatorio()`**
  Método estático que genera un murciélago en una posición aleatoria en alguno de los cuatro bordes de la pantalla (arriba, derecha, abajo o izquierda). Esto permite que los enemigos aparezcan desde fuera de la pantalla y se dirijan hacia el jugador.

- **Clase Poción:**
  Esta clase representa una poción en el juego, que puede ser recogida por el jugador para recuperar vida. La poción tiene una duración limitada en pantalla (6 segundos) y puede desaparecer si no se recoge a tiempo.

```java
public class Pocion {
    private int x;
    private int y;
    private Image imagen;
    private long tiempoCreacion; // Momento en el que fue creada la poción (en milisegundos)

    public Pocion(int x, int y) { // Constructor: crea una poción en la posición (x, y)
        this.x = x;
        this.y = y;
        this.imagen = new ImageIcon("Imagenes/pocion.png").getImage();
        this.tiempoCreacion = System.currentTimeMillis();
    }

    public void dibujar(Entorno entorno) {
        entorno.dibujarImagen(imagen, x, y, 0);
    }

    public boolean colisionaCon(int x, int y, int radio) { //colision con otro objeto
        return Math.hypot(this.x - x, this.y - y) < radio;
    }

    public int getX() { return x; } // Devuelve la coordenada x de la poción
    public int getY() { return y; }

    public boolean expirada() {
        return System.currentTimeMillis() - tiempoCreacion >= 6000; // 6 segundos
    }

}
```

Variables de instancia

- **`private int x;`**
  Indica la posición horizontal (coordenada X) de la poción en el entorno del juego.

- **`private int y;`**
  Indica la posición vertical (coordenada Y) de la poción.

- **`private Image imagen;`**
  Contiene la imagen gráfica de la poción, cargada desde un archivo PNG (`"Imagenes/pocion.png"`).

- **`private long tiempoCreacion;`**
  Almacena el momento exacto en que la poción fue creada, en milisegundos. Se usa para determinar cuánto tiempo ha estado la poción en pantalla.

Constructor

- **`public Pocion(int x, int y)`**
  Crea una nueva poción en la posición `(x, y)`, carga su imagen y registra el instante de su creación con `System.currentTimeMillis()`.

Métodos

- **`public void dibujar(Entorno entorno)`**
  Dibuja la imagen de la poción en el entorno gráfico en su posición actual `(x, y)`, sin rotación.

- **`public boolean colisionaCon(int x, int y, int radio)`**
  Verifica si hay colisión con otro objeto (por ejemplo, el jugador). Usa `Math.hypot` para comprobar si está dentro de un radio de proximidad determinado.

- **`public int getX()`**
  Devuelve la coordenada horizontal actual de la poción.

- **`public int getY()`**
  Devuelve la coordenada vertical actual de la poción.

- **`public boolean expirada()`**
  Indica si la poción ha estado en pantalla por más de 6 segundos. Si es así, retorna `true`, y puede ser eliminada del juego.

Conclusión

A lo largo del proyecto, se logró implementar un juego funcional en Java utilizando la librería Entorno, en el que múltiples elementos , como el jugador, enemigos, proyectiles, hechizos, pociones y un jefe final, interactúan de forma coherente y dinámica.

Durante el proceso, se aprendió la importancia de mantener un código modular y organizado, utilizando clases bien definidas con responsabilidades claras. Además, se valoró la necesidad de realizar pruebas frecuentes y ajustar tanto la lógica como los aspectos visuales del juego.

Uno de los aprendizajes más valiosos fue la gestión del tiempo y la planificación de funcionalidades. Integrar elementos visuales, como imágenes y animaciones, implicó entender cómo cargar recursos externos y manejar correctamente sus posiciones y comportamientos en el entorno gráfico.

En cuanto a los resultados obtenidos, se logró cumplir con los objetivos planteados: el personaje puede moverse, atacar, usar hechizos, enfrentarse a enemigos con comportamientos propios y recolectar pociones. El juego presenta una dinámica entretenida y responde correctamente a las acciones del jugador, lo que demuestra la efectividad del diseño y la implementación.

En resumen, este proyecto no solo permitió consolidar conocimientos técnicos, sino también desarrollar habilidades prácticas como el trabajo con clases interrelacionadas, la detección de colisiones, y el manejo de eventos y condiciones. Fue una experiencia enriquecedora que fortaleció tanto el pensamiento lógico como la creatividad en la construcción de sistemas interactivos.