Universidad Nacional de General Sarmiento

Programación I- Primer Semestre 2025
Trabajo Practico: El camino de Gondolf

COM-7

Profesores:

- Cesar Niveyro - cniveyro@campus.ungs.edu.ar
- Nahuel Sauma - nsauma@campus.ungs.edu.ar

Grupo 10

Alumnos:

- **Ponce,Dylan Alexander - 43.917.267 - dylan18ponce2323@gmail.com**
- **Carrizo,Nahuel - 47.014.559 - carrizonahue86@gmail.com**

Universidad Nacional de General Sarmiento

***1*. Introducción** En el presente informe se documenta el proceso de continuación y mejora del desarrollo de un videojuego. El proyecto originalmente se encontraba en fase base, y el equipo asumió la tarea de finalizarlo e implementar nuevas clases, y métodos. La inclusión de la clase Menú, Personaje, Enemigo, entre otras. Uso de métodos del entorno, para realizar ciertas acciones en el juego,uso de mouse, presión de teclas, además de la incorporación de imágenes para mejorar la estética del juego.

***2*. Descripción** Se implementaron 7 nuevas clases al proyecto, las cuales 6 son las más importantes para el funcionamiento del juego, ellas son, Personaje, Enemigo, Obstáculo, Hechizo, Menú, Botón. Nos encontramos con obstáculos a la hora del desarrollo del trabajo, tales como la división de responsabilidades entre las clases y diferentes métodos y cuestiones más específicas, tales como calcular la colisión entre un círculo y un rectángulo o trabajar con objetos nulos.

Clase Personaje - Recibe parámetros y atributos que definen al Personaje Mago, realiza las acciones indicadas que puede hacer, y dibuja al mismo.

```
public class Personaje {

    //Parametros
    private int x;
    private int y;
    private int ancho;
    private int alto;

    //Atributos
    private int velocidad;
    private int vida;
    private int mana;

    //Rectangulo Color
    private Color color;

    // Constructor del Personaje
    public Personaje(int x, int y, int ancho, int alto, int velocidad, int vida, int mana) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.velocidad = velocidad;
        this.vida = vida;
        this.mana = mana;
        this.color= Color.BLUE;
    }
}
```

Métodos de la Clase Personaje:

- *dibujar()* dibuja el Personaje, a partir de los parámetros pasados, dibuja un rectángulo en pantalla.
- *mover()* se encarga del movimiento del personaje.
- *colisionarPorDerecha/Izquierda/Arriba/Abajo()* retorna un booleano, con el objetivo de que el Personaje, no atraviese los límites del entorno, y sobrepase el Menú.

- *colisionaConRoca()* retorna un booleano, con el objetivo de que el Personaje, no atraviese las Rocas/Obstáculos del mapa.
- *recibirDanio()* cuando el personaje recibe daño, se encarga de bajarle la vida al mismo.
- *reducirMana()* cuando el personaje una un hechizo, que gaste mana, el mana se reduce.
- *recuperarVida()* cuando el personaje, pasa por una Poción de Vida, recuperar una cantidad de vida.
- *recuperarMana()* cuando el personaje, pasa por una Poción de Maná, recuperar una cantidad de maná.

Clase Enemigo - Recibe parámetros y atributos que definen al Enemigo Murciélago, realiza las acciones indicadas que puede hacer, y dibuja al mismo.

```java
public class Enemigo {

    //Parametros
    private double x;
    private double y;
    private double ancho;
    private double alto;

    //Atributos
    private double velocidad;
    private double danio;

    //Rectangulo Color
    private Color color = Color.GREEN;

    // Constructor del Enemigo
    public Enemigo(double x, double y, double ancho, double alto, double velocidad, double danio) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.velocidad = velocidad;
        this.danio = danio;
    }
}
```

Métodos de la Clase Enemigo:

- *dibujar()* dibuja el Enemigo, a partir de los parámetros pasados, dibuja un rectángulo en pantalla.
- *perseguir()* se encarga del movimiento del enemigo, a partir de las coordenadas de mi personaje, actualiza la dirección del enemigo hacia el personaje.
- *colisionaCon()* retorna un booleano, que define si mi enemigo colisionó con el personaje.
- *colsionaConHechizo()* retorna un booleano, que define si mi enemigo colisionó con el hechizo.

Clase Obstáculo - Recibe parámetros que definen al Obstáculo, y dibuja al mismo.

```java
public class Obstaculo {

    //Parametros
    private int x;
    private int y;
    private int ancho;
    private int alto;

    //Constructor de Obstaculo
    public Obstaculo (int x, int y, int ancho, int alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
    }
}
```

Métodos de la Clase Obstáculo:

- *dibujar()* dibuja el Obstáculo, a partir de los parámetros pasados, dibuja un rectángulo en pantalla.

Clase Menú - Recibe parámetros que definen al Menú, lo dibuja en pantalla, además de imprimir los datos de vida, mana, y enemigos eliminados en pantalla.

```java
public class Menu {

    //Parametros
    private int x;
    private int y;
    private int ancho;
    private int alto;

    //Construcor del Menu
    public Menu(int x, int y, int ancho, int alto) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
    }
}
```

Métodos de la Clase Menú:

- *dibujar()* dibuja el Menú, a partir de los parámetros pasados, dibuja un rectángulo en pantalla, además de imprimir los datos datos de vida, mana, y enemigos eliminados.
- *getBordeIzquierdo()* retorna un valor entero, importante para obtener el borde derecho del mapa menos el menú. Obteniendo la coordenada x.

Clase Boton - Recibe parámetros que definen al Botón, lo dibuja en pantalla, además de darle un funcionamiento a los mismos, a partir de seleccionarlos o deseleccionarlos.

```java
public class Boton {
    private int x;
    private int y;
    private int ancho;
    private int alto;
    private String nombreHechizo;
    private boolean seleccionado = false;
    private Color colorOriginal;
    private Color colorSeleccionado = Color.GREEN;

    // Constructor clase boton
    public Boton(int x, int y, int ancho, int alto, Color color, String nombreHechizo) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.nombreHechizo = nombreHechizo;
        this.colorOriginal = color;
        this.nombreHechizo = nombreHechizo;
    }
}
```

Métodos de la Clase Botón:

- *seleccionar()* cambia la condición del boolean *seleccionado,* usado para seleccionar un botón.
- *deseleccionar()* cambia la condición del boolean *seleccionado,* usado para la deseleccion un botón.
- *estaSeleccionado()* es la condición actual del boolean *seleccionado,* si el botón está seleccionado.
- *dibujar()* dibuja el Botón, a partir de los parámetros pasados, dibuja un rectángulo en pantalla que tiene ciertas condiciones para el cambio de color si el mouse esta sobre (metodo *mouseSobre()*) , además de imprimir el nombre del hechizo.
- *fueClickeado()* verifica si fue o no clickeado el botón, retornando un booleano, a partir de las coordenadas del mouse, y las coordenadas del botón, si se apretó el botón izquierdo del mouse en esa posición.
- *mouseSobre()* verifica si el mouse está sobre él las coordenadas del botón, retornando un booleano, esto es importante para la clase dibujar, ya que si el puntero del mouse, esta sobre el boton, el mismo cambia de color.
- *mouseSobreJuego()* retorna un booleano, para verificar que una vez seleccionado el botón, esté active un hechizo, el cual debe ser lanzado en el mapa jugable, evitando conflictos si el clic fue sobre un botón.

Clase Hechizo - Recibe parámetros que lo definen,condiciones en la que se encuentra el mismo, activo o no, lo dibuja en pantalla, teniendo en cuenta el radio. Y si se usa, dependiendo del hechizo, descuenta o no el maná.

```
public class Hechizo {
    //Parametros
    private int radio;
    private Color color;
    private boolean activo;
    private int anchoJuego;
    private int anchoMenu;
    private int costoMana;
    private Personaje mago;

    //Constructor Hechizo
    public Hechizo(int anchoJuego, int anchoMenu, int costoMana, Personaje mago) {
        this.activo = false;
        this.anchoJuego = anchoJuego;
        this.anchoMenu = anchoMenu;
        this.radio = 0;
        this.color = Color.RED;
        this.costoMana = costoMana;
        this.mago=mago;
    }
}
```

Métodos de la Clase Hechizo:

- *activar()* define si el hechizo está activo o no, cambiando el valor del booleano *activo*, además de definir nuevos radios, dependiendo de los parámetros que entren (diferencias entre hechizos).
- *desactivar()* cambia el valor del booleano *activo,* para desactivar el hechizo que se está usando.
- *estaActivo()* es la condición actual del boolean *activo,* si el hechizo está activado.
- *dibujar()* dibuja el Hechizo, sí el mismo está activo, a partir de los parámetros pasados (entre ellos el radio), dibuja un círculo en pantalla.
- *descontarMana()* si el hechizo que se está usando, tiene un gasto de mana, llama al método *reducirMana()* para descontar el maná, dando un límite de uso al Personaje.
- *finalizarHechizo()* una vez que el hechizo fue utilizado, el mismo se debe desactivar, y se tienen que deseleccionar los botones.

EXTRAS

Clase Poción - Recibe parámetros que lo definen, se dibuja en el entorno, y dependiendo si colisiona o no con el mago, nos retorna un resultado.

```java
public class Pocion {
    //Parametros
    private double x;
    private double y;
    private double ancho;
    private double alto;
    private String tipo;

    //Imagen
    private Image imagenVida;
    private Image imagenMana;

    //Constructor de Pocion
    public Pocion(double x, double y,double ancho,double alto, String tipo) {
        this.x = x;
        this.y = y;
        this.ancho = ancho;
        this.alto = alto;
        this.tipo = tipo;

        //Imagenes
        imagenVida = Herramientas.cargarImagen("imagenes/Vida.png");
        imagenMana = Herramientas.cargarImagen("imagenes/Mana.png");
    }
}
```

Métodos de la Clase Poción:

- *dibujar()* a partir de los parámetros pasados, dibuja en el entorno la poción indicada.
- *colisionaCon()* retorna un booleano para saber si la poción colisiona con el mago en el juego.

No hubo tantos problemas en cuanto a la creación de las clases, en grupo desde un principio ya teníamos una idea de las clases necesarias para el desarrollo juego, y de algunos metodos de movimiento y interacción, pero creemos que se puede modificar aún más las clases y los métodos, para que estén mejor estructurados, y sean más entendibles para el usuario. Algunos métodos fueron complicados de desarrollar, métodos donde se usa la geometría de los objetos, pero gracias a los aportes de los profesores la tarea se hizo más sencilla.

3. Implementación - Juego

```java
package juego;

import java.awt.Color;

public class Juego extends InterfaceJuego
{
    // El objeto Entorno que controla el tiempo y otros
    private Entorno entorno;
    private Personaje mago;
    private Menu menu;
    private Obstaculo [] roca;
    private Mapa mapa;

    //Boton y Hechizos
    private Boton boton1;
    private Boton boton2;
    private Hechizos hechizo;

    //Estado del Juego
    private boolean juegoFinalizado = false;
    private boolean victoria = false;

    //Enemigos
    private Enemigo[] enemigos;
    private int totalEnemigos = 50;
    private int enemigosActivos = 10;
    private int enemigosEliminados = 0;
    private int enemigosGenerados = 0;
    private Random random = new Random(); // Agregado como atributo de la clase Juego

    //Pociones
    private Pocion[] pociones = new Pocion[20]; // Puede haber hasta 20 pociones en el mapa

    //Imagenes
    private boolean juegoIniciado = false;
    private Image imgInicio;
    private Image imgVictoria;
    private Image imgDerrota;

    Juego()
    {
        // Inicializa el objeto entorno
        this.entorno = new Entorno (this, "El camino de Gondolf", 800, 600);
        this.menu = new Menu (entorno.ancho()-100,entorno.alto()/2,200,entorno.alto());  //Modificar el x del menu(no se uso anchoMenu)
        this.mapa = new Mapa(100); // Cada tile mide 100x100 píxeles

        //Mago
        int centroX = menu.getBordeIzquierdo() / 2;
        int centroY = entorno.alto() / 2;
        int esquivar = 40;
        this.mago = new Personaje(centroX, centroY + esquivar, 20, 35, 3, 100, 100);


        //Roca
        int distanciado = 160; // Distancia del centro

        this.roca = new Obstaculo[] {
                // Esquinas más alejadas
                new Obstaculo(centroX - distanciado, centroY - distanciado, 30, 30), // Arriba izquierda
                new Obstaculo(centroX + distanciado, centroY - distanciado, 30, 30), // Arriba derecha
                new Obstaculo(centroX - distanciado, centroY + distanciado, 30, 30), // Abajo izquierda
                new Obstaculo(centroX + distanciado, centroY + distanciado, 30, 30), // Abajo derecha
                new Obstaculo(centroX, centroY, 30, 30)                              // Centro
        };

        //Boton
        int menuX = menu.getX();
        int menuTop = menu.getY() - menu.getAlto() / 2;

        int botonAncho = 150;
        int botonAlto = 75;
        int separacion = 10;

        int margenSuperior = 100;

        this.boton1 = new Boton(menuX, menuTop + botonAlto / 2 + margenSuperior, botonAncho, botonAlto, Color.CYAN, "Rayo congelante");
        this.boton2 = new Boton(menuX, menuTop + 3 * botonAlto / 2 + separacion + margenSuperior, botonAncho, botonAlto, Color.ORANGE, "Tormenta de fuego");

        //Hechizo
        this.hechizo = new Hechizos(entorno.ancho(), entorno.ancho()-menu.getBordeIzquierdo());

        //Enemigos
        this.enemigos = new Enemigo[enemigosActivos];

        for (int i = 0; i < enemigosActivos; i++) {
            enemigos[i] = crearAleatorio(entorno.ancho(), entorno.alto(), entorno.ancho()-menu.getBordeIzquierdo());
            enemigosGenerados++;
        }

        this.imgInicio = Herramientas.cargarImagen("imagenes/inicio.gif");
        this.imgVictoria = Herramientas.cargarImagen("imagenes/victoria.jpg");
        this.imgDerrota = Herramientas.cargarImagen("imagenes/derrota.gif");

        this.entorno.iniciar();
    }

    public void tick()
    {
        if (!juegoIniciado) {
            mostrarPantallaInicio();
            return;
        }

        if (juegoFinalizado) {
            mostrarPantallaFinal();
            return;
        }
```

```
        //Dibujado
        mapa.dibujar(entorno);
        mago.dibujarAnimado(entorno);
        menu.dibujarMenu(entorno, mago, enemigosEliminados);
        mago.moverAnimado(entorno, menu , roca);
        boton1.dibujar(entorno);
        boton2.dibujar(entorno);
        //hechizo.dibujar(entorno);
        hechizo.dibujarAnimado(entorno, boton1.estaSeleccionado(), boton2.estaSeleccionado());

        for (Obstaculo rocas : roca) {
            rocas.dibujarRoca(entorno);
        }

        // Enemigos

        // For que verifica las colisiones y realiza acciónes sobre ella
        for (int i = 0; i < enemigos.length; i++) {
            Enemigo e = enemigos[i];
            if (e != null && e.colisionaCon(mago)) {
                mago.recibirDanio(e.getDanio());
                enemigos[i] = null;
                intentarGenerarPocion(e.getX(), e.getY());
                enemigosEliminados++;
                }
            }
        // For con responsabilidad de mover a los enemigos
        for (int i = 0; i < enemigos.length; i++) {
            Enemigo e = enemigos[i];
            if (enemigos[i] != null) {
                e.perseguir(mago);
                }
            }
        // For con la responsabilidad de dibujar a los enemigos
        for (int i = 0; i < enemigos.length; i++) {
            Enemigo e = enemigos[i];
            if (e != null) {
                e.dibujarMurcielago(entorno);
                }
            }
        // For que genera enemigos
        for (int i = 0; i < enemigos.length; i++) {
            Enemigo e = enemigos[i];
            if (e == null && enemigosGenerados < totalEnemigos) {
                enemigos[i] = crearAleatorio(entorno.ancho(), entorno.alto(), entorno.ancho()-menu.getBordeIzquierdo());
                enemigosGenerados++;
                }
            }
        // Verifica si el jugador perdió
        if (mago.getVida() <= 0) {
            juegoFinalizado = true;
            victoria = false;
            }
        // Verifica si el jugador ganó
        if (enemigosEliminados >= totalEnemigos) {
            juegoFinalizado = true;
            victoria = true;
            }
```

```
        // Selección de hechizos
        if (boton2.fueClickeado(entorno) && mago.puedeLanzarAroDeFuego(hechizo.getCostoMana())) {
            boton2.seleccionar();
            boton1.deseleccionar();
            hechizo.activar(40, Color.RED, 10); // Aro de fuego (rojo, grande)
            }
        if (boton1.fueClickeado(entorno)) {
            boton1.seleccionar();
            boton2.deseleccionar();
            hechizo.activar(20, Color.BLUE, 0);; // Rayo congelante (azul, pequeño)
            }
        // Desactivar hechizo si hago clic en el mapa
        // Lanzamiento del hechizo
        int mouseX = entorno.mouseX();
        int mouseY = entorno.mouseY();
        if (Boton.mouseSobreJuego(mouseY, mouseX, entorno.ancho(), entorno.ancho()-menu.getBordeIzquierdo(), boton1, boton2, entorno)
                && entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO) && hechizo.estaActivo()) {
                    // Reduce el mana si clickeo el boton 2
                    if (boton2.estaSeleccionado()) {
                        mago.reducirMana(hechizo.getCostoMana());
                    }
                    // Elimina enemigos en el área del clic
                    for (int i = 0; i < enemigos.length; i++) {
                        Enemigo e = enemigos[i];
                        if (e != null) {
                            int dx = (int) (e.getX() - mouseX);
                            int dy = (int) (e.getY() - mouseY);
                            // Verifica si el hechizo colisiona con el enemigo
                            if (e.colisionaConHechizo(dx, dy, hechizo)) {
                                enemigos[i] = null;
                                intentarGenerarPocion(e.getX(), e.getY());
                                enemigosEliminados++;
                            }
                        }
                    }
                    // Desactivar hechizo y botones
                    Hechizos.finalizarHechizo(hechizo, boton1, boton2);
                }
        dibujarYRecolectarPociones();

    }
```

```java
    //Estados del Juego
    private void reiniciarJuego() {
        int centroX = menu.getBordeIzquierdo() / 2;
        int centroY = entorno.alto() / 2;
        int esquivar = 40;
        this.mago = new Personaje(centroX, centroY + esquivar, 20, 35, 3, 100, 100);
        this.enemigosEliminados = 0;
        this.enemigosGenerados = 0;
        this.enemigos = new Enemigo[enemigosActivos];
        this.pociones = new Pocion[20];

        for (int i = 0; i < enemigosActivos; i++) {
            enemigos[i] = crearAleatorio(entorno.ancho(), entorno.alto(), entorno.ancho()-menu.getBordeIzquierdo());
            enemigosGenerados++;
            }
        this.juegoFinalizado = false;
        this.victoria = false;
        }

    private void mostrarPantallaFinal() {
        if (victoria) {
            entorno.dibujarImagen(imgVictoria, entorno.ancho() / 2, entorno.alto() / 2, 0);
            entorno.cambiarFont("Arial", 20, Color.GREEN);
            entorno.escribirTexto("¡GANASTE!", entorno.ancho() / 2 - 80, entorno.alto() - 70);
        } else {
            entorno.dibujarImagen(imgDerrota, entorno.ancho() / 2, entorno.alto() / 2, 0);
            entorno.cambiarFont("Arial", 20, Color.RED);
            entorno.escribirTexto("¡DERROTA!", entorno.ancho() / 2 - 80, entorno.alto() - 70);
        }

        entorno.cambiarFont("Arial", 24, Color.WHITE);
        String textoReinicio = "Presiona ENTER para reiniciar";
        int xTexto = entorno.ancho() / 2 - textoReinicio.length() * 6;
        entorno.escribirTexto(textoReinicio, xTexto, entorno.alto() - 40);

        if (entorno.sePresiono(entorno.TECLA_ENTER)) {
            reiniciarJuego();
            juegoIniciado = true;
        }
    }

    private void mostrarPantallaInicio() {
        entorno.dibujarImagen(imgInicio, entorno.ancho() / 2, entorno.alto() / 2, 0);
        entorno.cambiarFont("Arial", 24, Color.WHITE);
        String textoInicio = "Presiona ENTER para comenzar";
        int xInicio = entorno.ancho() / 2 - textoInicio.length() * 6;
        entorno.escribirTexto(textoInicio, xInicio, entorno.alto() - 40);

        if (entorno.sePresiono(entorno.TECLA_ENTER)) {
            juegoIniciado = true;
        }
    }
```

```java
    //Generacion de Enemigos
    public Enemigo crearAleatorio(int anchoJuego, int altoJuego, int anchoMenu) {
        double ancho = 15;
        double alto = 15;
        double velocidad = 1.6;
        double danio = 5;
        double x = 0;
        double y = 0;

        int margen = 25;
        int lado = random.nextInt(4); // 0: arriba, 1: abajo, 2: izquierda, 3: derecha

        switch (lado) {
            case 0: // Arriba
                x = random.nextInt(anchoJuego - anchoMenu - margen * 2) + margen;
                y = 0;
                break;
            case 1: // Abajo
                x = random.nextInt(anchoJuego - anchoMenu - margen * 2) + margen;
                y = altoJuego;
                break;
            case 2: // Izquierda
                x = 0;
                y = random.nextInt(altoJuego - margen * 2) + margen;
                break;
            case 3: // Derecha (antes del menú)
                x = anchoJuego - anchoMenu - 1;
                y = random.nextInt(altoJuego - margen * 2) + margen;
                break;
            }
        return new Enemigo(x, y, ancho, alto, velocidad, danio);
        }

    //Dibujar Pocion
    private void dibujarYRecolectarPociones() {
        for (int i = 0; i < pociones.length; i++) {
            if (pociones[i] != null) {
                pociones[i].dibujar(entorno);
                if (pociones[i].colisionaCon(mago)) {
                    if (pociones[i].getTipo().equals("VIDA")) {
                        mago.recuperarVida(20);
                    } else {
                        mago.recuperarMana(30);
                    }
                    pociones[i] = null;
                }
            }
        }
    }
```

```
320     //Generar Pocion
321     private void intentarGenerarPocion(double x, double y) {
322         if (random.nextDouble() < 0.1) {
323             String tipo = random.nextBoolean() ? "VIDA" : "MANA";
324             for (int i = 0; i < pociones.length; i++) {
325                 if (pociones[i] == null) {
326                     pociones[i] = new Pocion(x, y,10,10, tipo);
327                     break;
328                 }
329             }
330         }
331     }
332
333     @SuppressWarnings("unused")
334     public static void main(String[] args)
335     {
336         Juego juego = new Juego();
337     }
338 }
339
```

Clase Juego - Implementación de todos los métodos y llamados a todas las clases creadas, para darle un funcionamiento al juego.

Se agregaron algunos métodos para dar una condición al juego.

- *reiniciarJuego()* dependiendo en la condición en la que se encuentre el juego, sea derrota o victoria, me dan la oportunidad de volver a jugar. Por lo que los parámetros de juego, se deben reiniciar.
- *mostrarPantallaFinal()* y *mostrarPantallaInicio()* ambas, tienen un objetivo casi similar, que es el de mostrar un mensaje y una imagen. La pantalla inicial es una carta de presentación al juego,mientras que la pantalla final, depende de la condición en la que finalicemos el juego.
- *crearAleatorio()* es importante para generar un enemigo nuevo en el mapa, dado que el mismo spawnea en distintas ubicaciones, de los bordes de la zona jugable.
- *dibujarYRecolectarPociones()* y *intentarGenerarPocion()* ambas son importantes pero con objetivos distintos, por un lado, necesitamos ver la probabilidad de generar una poción, y su tipo. Por otro lado,necesitamos dibujar la poción, y si el mago colisiona con ella, dependiendo del tipo de poción, recuperar un valor de vida o mana.

4. **Conclusiones** Sin lugar a duda este trabajo nos ha hecho mejorar en varios aspectos en cuanto a nuestra lógica de programación, y también nos ha preparado para la resolución de problemas más complejos. Un aprendizaje que nos llevamos es la mejora que hemos desarrollado al trabajar en equipo, entendiendo mejor cómo funciona el flujo de trabajo en proyectos en equipo.