TRABAJO PRÁCTICO – PROGRAMACIÓN 1

"EL CAMINO DE GONDOLF"

Integrantes:

1) Ulises Rodríguez
2) Agustín Gallicchio

Dni:

1) 46634940
2) 46089011

Mail:

1) uli.leo.rod@gmail.com
2) agustingallicchio2017@gmail.com

Docentes:

- Lucas Bidart Gauna
- Zoe Guadalupe Sacks

Comisión: 03

Introducción:

Este trabajo práctico consistió en el desarrollo de un videojuego 2D llamado *El Camino de Gondolf*. El juego está protagonizado por un mago que debe defenderse de oleadas de murciélagos enemigos usando diferentes hechizos mágicos, mientras intenta sobrevivir y completar su misión.

El objetivo del proyecto fue diseñar e implementar un juego interactivo con distintos elementos: un personaje principal controlable, enemigos con movimiento autónomo, obstáculos, una interfaz con información del jugador, y botones que permiten seleccionar y lanzar hechizos. Todo fue desarrollado con lógica de programación orientada a objetos (POO), utilizando estructuras claras y organizadas que permiten mantener el código ordenado y funcional.

A lo largo del informe se detallan las decisiones de diseño, el funcionamiento del juego y las principales funcionalidades implementadas.

Descripción:

Para llevar a cabo este trabajo práctico, el videojuego cuenta con clases principales como: **"Mago"**, que representa al jugador; **"Murciélago"**, que representa a los enemigos; **"Hechizos"**, para los distintos ataques mágicos; **"Roca"**, como obstáculo del escenario; **"PantallaFinJuego"**, para mostrar por pantalla un mensaje de cuando el jugador gana o pierde; **"Menu"**, que muestra información del jugador mago; **"Boton"**, que nos ayudará para seleccionar un hechizo; **"FuncionesUtiles"**, como herramienta auxiliar para facilitar las tareas comunes del juego; y por último **"Juego"**, que es la principal de todas, sin ella no podríamos hacer funcionar el proyecto.

A continuación, se describe cada clase detallando sus atributos principales, sus métodos más importantes, y su rol dentro del sistema. También se comentan los desafíos encontrados durante el desarrollo y las soluciones adoptadas para resolverlos.

. Clase Mago:

La clase Mago representa al personaje principal controlado por el jugador. Está diseñada para gestionar tanto los atributos del mago como su comportamiento y visualización dentro del entorno del juego.

Variables de instancia:

- X e Y: Coordenadas que definen la posición del mago en la pantalla.
- Ancho y Altura: Dimensiones del Personaje.
- Velocidad: Define que tan rápido se desplaza el mago.
- Vida y Mana: Recursos del mago que se consumen. La vida se irá perdiendo cada vez que un murciélago colisione con el mago; mientras que el maná se va gastando cada vez que lanzamos un Hechizo.

- Imagen: Es la imagen actual que representa visualmente al mago (Cambia según la animación).

- Variables de animación: (contadorFrames, velocidadFrames, frameAbajo, frameArriba, frameIzquierda, frameDerecha): utilizadas para manejar los ciclos de animación en cada dirección.

Métodos Principales:

- Constructor: Inicializa al mago en una posición centrada de la pantalla, asignándole dimensiones, velocidad, recursos y una imagen inicial.
- AnimarMovimiento: Método autilizado para cambiar el frame actual, logrando que la animación se vea de forma fluida al moverse.
- Getters y Setters: Para acceder y modificar las propiedades del mago.
- Dibujar / DibujarImagenMago: Permiten visualizar al mago, ya sea como figura geométrica o con una imagen animada.
- Luego tenemos los bordes (izquierda, derecha, arriba y abajo) del propio mago.
- ColisionaMagoRoca: Detecta colision entre mago y las rocas, utilizando lógica de colisión por suposición de bordes.
- Movimientos (moverDerecha, moverIzquierda, moverArriba, moverAbajo): Cambian la posición del pago y actualiza el frame correspondiente.
- estaMuerto: Verifica si la vida del mago llegó a cero.
- dibujarVida/dibujarMana: visualiza gráficamente la barra de vida y el maná.

Problemas/Decisiones:

- Durante el desarrollo del juego, surgió un incoveniente al posicionar al personaje Mago en el centro de la pantalla jugable. Inicialmente se tomaba como referencia el ancho total de la ventana del juego, lo que provocaba que se centrara en el centro absoluto sin considerar el área del menú. Para solucionarlo, en lugar de utilizar el ancho completo de la ventana, se tomó como referencia solo el ancho del área de juego (excluyendo el menú). Por ende, para el valor X, se suma el ancho del menú más el ancho del menú dividido por 2. En cuanto el valor Y, se resta la altura total de la ventana divido 2 menos el alto del mago dividido 2. Gracias a esto el mago puede centrarse de forma correcta.
- Además, uno de los desafíos fue organizar correctamente los frames de animación para que se sincronizaran con la dirección del movimiento. También fue necesario ajustar el retardo entre cuadros (velocidadFrames) para que la animación se vea fluida, sin hacer lento el juego. Para resolver esto, se implementó una lógica de contador que reinicia el ciclo de animación después de alcanzar el último frame.

. Clase Murcielago:

La clase Murciélago representa un enemigo dentro del juego. Controla la posición, tamaño, animación y dibujo del murciélago, permitiendo mostrar animaciones de movimiento hacia la izquierda o derecha según la posición relativa del mago.

Variables de instancia:

- X e Y: Coordenadas que definen la posición del murcielago en la pantalla.
- Ancho y Altura: Dimensiones del Murciélago
- Imagen: Es la imagen actual que representa visualmente del mago (Cambia según la animación).
- Variables de animación: (contadorFrames, velocidadFrames, frameActual): utilizadas para manejar las animaciones.
- movimientoIzquierda y movimientoDerecha: Arrays de imagenes para del movimiento izquierdo y del derecho.

Métodos Principales:

- Constructor: recibe la posición inicial en x e y. Recibe 2 arrays con los nombres de los frames para las animaciones de la derecha e izquierda. Definimos para nuestro murciélago los arrays de frames para cada lado; y cargamos las imágenes recibidas para cada lado.
- Getters y Setters: Para acceder y modificar las propiedades del murciélago en X e Y.
- ActualizarAnimacion: Controla el conjunto de frames según la posición relativa del mago. Si el murciélago está a la derecha del mago, usa los frames de movimiento hacia la izquierda. Lo mismo si está del lado contrario. Además, actualiza la imagen del siguiente frame.
- dibujarImagen: dibuja el murciélago
- Luego tenemos los bordes (izquierda, derecha, arriba y abajo) del propio murciélago.

Problemas/Decisiones:

- Inicialmente, el comportamiento del murciélago en nuestra implementación se basaba en movimientos predefinidos en ocho direcciones fijas: arriba, abajo, izquierda, derecha y sus diagonales. Sin embargo, esto resultó poco eficiente, ya que el mago podía desplazarse libremente en cualquier dirección, y el murciélago no lograba seguirlo de forma fluida. Por ende, decidimos modificarlo con una estructura de código diferente. Reestructuramos el código utilizando un cálculo de distancia basado en la posición relativa entre el mago y el murciélago. Mediante la normalización de un vector que apunta del murciélago hacia el mago, logramos un movimiento más suave, continuo y dinámico.

. Clase Hechizos:

La clase **Hechizos** representa un hechizo que puede ser lanzado en el juego, con atributos para su posición, tamaño, costo de maná, tipo (nombre), animación visual y trayectoria. Permite crear hechizos con animaciones específicas (Fuego o Hielo), calcular su movimiento en base a una trayectoria hacia un destino, y dibujarlos en pantalla con animación.

Variables de instancia:

- Área de Efecto: Tenemos la posición X e Y del Hechizo en el mapa; y su ancho y Altura, que determinan el tamaño del hechizo.
- costoMana: el precio del maná para lanzar el hechizo.
- dirección: dirección en la que se mueve el hechizo, representada como valor decimal.
- Vx y Vy: variables de velocidad de la trayectoria.
- Imagen: imagen actual a dibujar (frame actual).
- imagenesFuego e imagenesHielo: arrays con los frames de animación para cada tipo de Hechizo.
- contadorFrames y velocidadFrames: control de velocidad para cambio de frames.
- Frame y angulo: nos determinara el frame actual del movimiento hacia la dirección.
- movimientoHechizoFuego y movimientoHechizoHielo: conjunto de imágenes para animar el hechizo de Hielo y Fuego.

Métodos Principales:

- Constructor: Inicializa la posición inicial, el tamaño, el nombre, el costo de maná y el ángulo. Carga las imágenes correspondientes para fuego y hielo. Si tieneTrayectoria es true, se calcula la trayectoria desde el X e Y del mago, hacia el X e Y del puntero del mouse. Para luego establecer Vx y Vy.
- cargarImagenes: Carga y retorna un array de imágenes
- animarMovimiento: Controla la animación de los frames.
- getCostoMana: devuelve el costo de maná del hechizo.
- permitirDisparar: verifica si el puntero del mouse está en un área válida para permitir disparar.
- dibujarImagenFuego y dibujarImagenHielo: dibuja el hechizo hielo y fuego en pantalla.
- calcularTrayectoria: calcula la velocidad en X e Y para que el hechizo se mueva hacia el destino con una velocidad normal.
- moverFuego y moverHielo: actualiza la animación de los frames. Se tiene en cuenta la posición del hechizo fuego y hielo según Vx y Vy.
- Luego tenemos los bordes (izquierda, derecha, arriba y abajo) del propio Hechizo.

Problemas/Decisiones:

- Otro de los problemas que enfrentamos en el proyecto fue el tema de los hechizos. Al principio, optamos por una solución basada en arreglos (arrays): creamos un array para el hechizo de fuego y otro para el hechizo de hielo. Cada uno contenía múltiples instancias predefinidas de su respectivo hechizo (por ejemplo, el de fuego con 3 elementos y el de hielo con 10). Sin embargo, esto provocaba un problema importante: al lanzar un solo hechizo, se disparaban todos los elementos del array simultáneamente, lo cual era completamente indeseado. Es decir, si el mago realizaba un solo disparo de fuego, se lanzaban las 3 instancias del array en vez de solo una. Para resolverlo, replanteamos la

lógica desde cero. En lugar de mantener un array de hechizos activos, diseñamos el sistema para que cada disparo genere una nueva instancia del hechizo dinámicamente. Esta instancia es lanzada, y al colisionar con un murciélago o con los bordes de la pantalla, se elimina. De esta manera, el mago puede lanzar hechizos de forma controlada, uno por uno, lo que permite un comportamiento mucho más intuitivo y funcional.

- Respecto al hechizo de hielo, al principio intentamos aplicar el mismo enfoque, pero luego optamos por una mecánica diferente. En este caso, el hechizo se origina en la posición del puntero del mouse, generando una masa de hielo de gran tamaño. Esta área actúa como una trampa que elimina a todos los murciélagos que colisionan con ella. Para evitar que este hechizo sea excesivamente poderoso o se mantenga de forma indefinida, limitamos su duración y le asignamos un alto costo de maná, a diferencia del hechizo de fuego, que puede utilizarse libremente sin costo alguno.

. Clase Roca:

La clase `Roca` representa un objeto estático (una roca) dentro del juego. Cada roca tiene una posición en el plano (x,y), un tamaño fijo, y una imagen que la representa visualmente. Se usan métodos para dibujar la roca y obtener sus bordes, principalmente para la detección de colisiones.

Variables de instancia:

- X e Y: Coordenadas que definen la posición de la roca en la pantalla.
- Ancho y Altura: Dimensiones del Murciélago.
- Imagen: utilizado para cargar visualmente a la roca
- Imágenes: array utilizado para diferentes imágenes de roca.

Métodos Principales:

- Constructor: Inicializa las posiciones en X e Y. Recibe un índice llamado “tipo” para seleccionar la imagen de la roca dentro del array de imágenes.
- dibujarImagenRoca: Dibuja la imagen de la roca en la posición X e Y dentro del juego.
- Luego tenemos los bordes (izquierda, derecha, arriba y abajo) de la propia Roca.

Problemas/Decisiones:

- Desde el inicio, el mapa incluía al menos cinco rocas estáticas colocadas estratégicamente para dificultar el desplazamiento del jugador. El problema surgió con la primera implementación de la colisión, donde usábamos una lógica booleana como bandera: si se detectaba cualquier colisión entre el mago y una roca, simplemente se anulaba el movimiento, lo que hacía que el personaje no pudiera avanzar en ninguna dirección. Para mejorar esto, mantuvimos el uso de banderas booleanas, pero ahora su función se centra en detectar específicamente desde qué lado se produce la colisión**.** Por ejemplo, si el lado derecho del mago colisiona con el lado izquierdo de una roca**,** la bandera correspondiente se activa (`true`), y en lugar de bloquear por completo el movimiento, se impide solo

avanzar hacia la derecha, permitiendo el desplazamiento hacia izquierda, arriba o abajo. Esta lógica se aplica simétricamente a los otros lados: si la colisión es por la izquierda, se impide ir hacia ese lado, pero no hacia el resto, y así sucesivamente.

. Clase PantallaFinJuego:

La clase PantallaFinJuego se utiliza para mostrar una pantalla final en el juego cuando el jugador gana o pierde. Presenta una imagen que indica el resultado y permite salir del juego presionando la tecla ESC bajo ciertas condiciones.

Variables de instancia:

- X e Y: Coordenadas que son definidas para centrar la imagen en la pantalla.
- Ancho y Altura: Son las dimensiones que se mostrarán la imagen.
- imagenFin: Imagen que indica si el jugador ganó o perdió (pantalla de fin de Juego).
- EstadoFin: Estado del juego (1 == ganó; 2 == perdió).

Métodos Principales:

- Constructor: Para el ancho y la altura, reciben como valor los argumentos pasados en el parámetro. Luego para el X e Y, se dividen los mismos argumentos por 2; para X se toma el ancho dividido por 2 y la Y se toma la altura dividida por 2.
- salirJuego: Me permite cerrar el juego si el estadoFin es igual a 2. Si es así, podré presionar la tecla ESC.
- dibujarImagenFinJuego: Dibuja la imagen de fin en la posición X e Y centrada.
- Getters y Setters: Encontramos para las coordenadas X e Y. Obtenemos el set y get de ancho; pero solo se obtiene el get de altura.

Problemas/Decisiones:

- No presentó problemas. Solo se instanció con una imagen de victoria y otra de derrota, según si el jugador gana o pierde.

. Clase Menu:

La clase Menú representa un panel visual que se muestra en el costado derecho de la pantalla del juego. Contiene una imagen (menú gráfico) y está centrado verticalmente. Su función es meramente visual, aportando un componente estético y/o informativo a la interfaz del juego.

Variables de instancia:

- X e Y: Coordenadas que son definidas para centrar el menú en la pantalla.
- Ancho y Altura: Son las dimensiones que se mostrarán la imagen.
- Imagen: Es la Imagen del menú cargada.

Métodos Principales:

- Constructor: Recibe el ancho y la altura total de la ventana del juego. Genera un valor en ancho, altura, x e y haciendo operaciones con los valores recibidos en los parámetros.
- dibujarImagenMenu: dibuja el menú en la pantalla
- Getters y Setters: Se mantiene los getters y setters de X, Y, Ancho y Altura

Problemas/Decisiones:

- A diferencia de otras clases del proyecto, la implementación del menú no representó mayores dificultades técnicas. Desde un principio, se definió que el menú estaría ubicado en el extremo derecho de la ventana del juego, ocupando una cuarta parte del ancho total de la pantalla. Para lograr esto, simplemente se instanció como un objeto que se posicionaba y redimensionaba dinámicamente en función del tamaño de la ventana. Uno de los detalles que tuvimos que resolver fue la adaptabilidad del menú ante cambios en la resolución o el redimensionamiento de la ventana.

. Clase Boton:

La clase Boton representa un botón gráfico que puede detectar si el cursor del mouse está sobre él y cambia de apariencia según su estado (seleccionado o no). Se utiliza en el menú del juego para elegir opciones, como la dificultad, mediante la interacción del usuario.

Variables de instancia:

- X e Y: Coordenadas que son definidas para centrar el botón en él menú.
- Ancho y Altura: Son las dimensiones de los botones.
- Seleccionado: indica si el botón fue seleccionado. Para eso inicia en false.
- imagenS: imagen cuando el botón está seleccionado.
- imagenDS: imagen cuando el botón no está seleccionado.

Métodos Principales:

- Constructor: Inicializa con la posición en X e Y. Toma valores para las dimensiones como el ancho y la altura. Si el botón está deseleccionado, la imagen toma la imagenA; sino toma la imagenB.

- estaDentro: Funcion para saber si el mouse está dentro del botón. Se toman los valores del punteroX y el punteroY (propios del mouse) y se calcula con los bordes del objeto botón para verificar si está dentro o no.
- estadoActual: retorna el estado actual del botón.
- cambiarEstado: retorna el estado negado del botón. Si está en true, entonces lo negamos y pasará a false.
- dibujarnImagenDificultad: dibuja siempre la imagen del estado deseleccionado con una escala personalizada como parámetro.
- dibujarnImagenBoton: dibuja la imagen del botón dependiendo del estado actual del botón.

Problemas/Decisiones:

- La clase Botón fue diseñada inicialmente con el propósito de permitir la selección de hechizos en el juego. Cada hechizo (fuego y hielo**)** cuenta con su propio botón en pantalla. Hacia las etapas finales del proyecto, esta clase también se reutilizó para implementar los botones de selección de dificultad del juego (como “fácil” o “normal”), aunque su función principal desde el inicio estuvo enfocada en los hechizos. El funcionamiento de los botones está basado en una lógica que impide que ambos hechizos estén activos al mismo tiempo, ya que eso rompería el equilibrio del juego y podría generar errores en el código. Para que un hechizo se active, se deben cumplir dos condiciones: primero, que el estado actual del botón esté desactivado, y segundo, que el puntero del mouse, tanto en la coordenada X como en la Y, se encuentre dentro del área del botón al momento de hacer clic. Si se cumplen estas condiciones, entonces el botón se activa y habilita el disparo del hechizo correspondiente. Además, el mago podrá solamente lanzar un hechizo a la vez. Por ejemplo, si el hechizo de fuego ya está activado y el jugador hace clic sobre el botón del hechizo de hielo, automáticamente el hechizo de fuego se desactiva y se activa el de hielo.

. Clase Funciones_Utiles:

La clase Funciones_utiles actúa como un conjunto de funciones auxiliares o de utilidad que facilitan ciertas operaciones en el juego, específicamente la detección de colisiones entre el personaje principal (Mago) y enemigos (Murcielago).
Es una clase estática y no requiere instanciación, siendo accesible desde cualquier parte del programa.

Variables de instancia:

- No posee atributos. Por lo tanto, todos los métodos pueden utilizarse directamente como Funciones_utiles.metodo().

Métodos Principales:

- colisionMagoMurcielago: Recibe mago y un array de murciélagos como parámetros. Se inicializa dentro una variable llamada colisiones de tipo int. Se recorre el array de murciélagos. Si el elemento murciélago es diferente de null,

se evalúa si sus límites colisionan con los del mago. En caso de colisión: se elimina al murciélago (null), se incrementa el contador de colisiones, y se resta uno de vida al mago. Por último, devuelve colisiones.

- seguimientoMurcielago: es utilizado para que el murciélago persiga al mago. Primero calcula la distancia en los ejes x e y entre el mago y el murciélago. Luego calcula la distancia real usando el teorema de Pitágoras (Raíz cuadrada de la suma de cuadrados). Si la distancia es mayor a cero (es decir, si están superpuestos), calcula una velocidad normalizada para mover el murciélago hacia el mago con una escala fija de velocidad. Finalmente actualiza la posición del murciélago sumando esta velocidad a sus coordenadas actuales haciendo que el murciélago se desplace hacia el mago de manera fluida.
- obtenerSpritesIzquierdaPorRonda y obtenerSpritesDerechaPorRonda: Son 2 funciones para obtener las imágenes de los enemigos dependiendo del número de ronda para cada enemigo correspondiente. Al pasar de ronda se actualiza el Sprite. Se recibe el número de ronda y retorna un array de enemigos de su respectiva ronda.
- generarMurcielago: Genera un nuevo murciélago en el juego en una posición aleatoria de los bordes de la pantalla.
- actualizarMurcielagos: Actualiza la cantidad de murciélagos que se generan por pantalla. Cuando el mago colisiona con el murciélago, ya sea con su cuerpo o con el hechizo, esta función detecta que faltan murciélagos en pantalla. Por lo tanto, dependiendo de la cantidad que faltan en pantalla, se genera esa cantidad de murciélagos.

Problemas/Decisiones:

- La clase Funciones_Utiles fue creada con el propósito de contener métodos auxiliares que se utilizan a lo largo del juego, evitando sobrecargar la clase principal Juego. Esto facilita una mejor organización del código, haciéndolo más claro, legible y fácil de mantener. Muchas veces no sabíamos dónde ubicar ciertas funciones que eran necesarias, pero no encajaban directamente en ninguna clase específica del juego. Por eso, decidimos agruparlas en esta clase aparte. Desde Juego, simplemente llamamos a los métodos de Funciones_Utiles cuando los necesitamos, lo que también ayuda a mantener una mejor separación de responsabilidades dentro del proyecto.

. Clase Juego:

La clase **Juego** es el núcleo central que controla la lógica principal del videojuego. Administra el ciclo de vida completo del juego, incluyendo la pantalla de inicio, selección de dificultad, control de pausas, actualización del estado del jugador y enemigos, manejo de entradas, dibujo de la escena y verificación del estado del juego para determinar si continúa o finaliza.

Esta clase coordina la interacción entre los distintos elementos del juego como el mago (jugador), murciélagos (enemigos), hechizos, y la interfaz gráfica.

Variables de instancia:

- El anchoVentana y alturaVentana: indican el ancho (900) y la altura (600) de la ventana.
- inicioJuego: se inicializa en false. Es una variable booleana para iniciar el juego
- mago: inicia el objeto mago.
- menu: inicia el objeto menu.
- PantallaFinJuegoGana y PantallaFinJuegoPierde: Instanciamos las variables para los objetos de pantalla. Tanto el que gana como el que pierde.
- rocas: Se inicializan 5 elementos de rocas, dándole coordenadas y el tipo.
- hudMago: carga la imagen del hud.
- imagenMago: carga la imagen del mago en el hud.
- imagenInicio: carga la imagen del inicio.
- punteroImagenes: Es un array de las imágenes de los punteros (fuego,hielo,normal).
- tipoPuntero: indicará que tipo de imagen se usará para el puntero.
- imagenPuntero: se carga la imagen del puntero seleccionado, utilizando el tipoPuntero dentro del array de punteroImagenes.
- imagenFondo: carga la imagen del mapa
- imagenDificultad: carga la imagen del selector de dificultad.
- imagenBtnFacil: carga la imagen del botón de dificultad fácil.
- imagenBtnNormal: carga la imagen del botón de dificultad normal.
- imagenBtnDificil: carga la imagen del botón de dificultad difícil.
- botonFacil: carga un botón de dificultad de fácil.
- botonNormal: carga un botón de dificultad de normal.
- botonDificil: carga un botón de dificultad de difícil.
- seleccionDificultad: booleano para seleccionar una sola vez la dificultad.
- rondaFinal: ronda final definida por la dificultad.
- imagenPausa: carga la imagen de la pausa.
- enPausa: variable iniciada en false para el estado de la pausa.
- esperaEnter: inicia en true cuando espera la tecla a que se toque el enter.
- cantMurcielagoPantalla: cantidad de murciélagos que hay en pantalla, que son 10.
- cantMurcielagoTotal: cantidad de murciélagos en total, que son 50.
- cantMurcielagoGenerados: variable de marca la cantidad generados en pantalla.
- cantMurcielagosEliminados: cantidad de murciélagos eliminados.
- murcielagos: se genera un array de murciélagos con la cantidad total.
- velocidadMurcielago: velocidad de murciélagos utilizado en las rondas siguientes.
- puntajeMurcielagos: Utilizado para marcar los puntos cada vez que eliminamos enemigos.
- hechizoFuego: Declaramos el Hechizo Fuego.
- hechizoHielo: Declaramos el Hechizo Hielo.
- punteroX y punteroY: Son las variables para el puntero.
- tiempo: variable de tiempo.
- tiempoInicio: se inicializa en -1
- temporizadorEjecutado: variable de ejecución de temporizador.

- tiempoFinal: variable para almacenar el tiempo que se mantuvo vivo el mago dentro del juego.
- puntajesGuardados: variable booleana que nos sirve para detener el tiempoFinal, ya que sino seguiría.
- salirCarga1: variable para salir de la ronda1.
- salirCarga2: variable para salir de la ronda 2.
- imagenRonda1: cargo la imagen de la ronda 1.
- imagenRonda2: cargo la imagen de la ronda 2.
- regenerarMana: variable para cargar el maná.
- expirarHielo: variable para marcar la cantidad de segundos que estará el hechizo Hielo.
- tiempoCarga: inicializa con 2 segundos ya que es el tiempo de espera para las pantallas de carga.
- numeroRonda: variable del número de ronda.
- botonHechizoFuego y botonHechizoHielo: se instancia el botón del hechizo fuego y de hielo.
- imagenBotonHieloS y imagenBotonHieloDs: carga las imágenes de los botones de hielo seleccionado y deseleccionado.
- imagenBotonFuegoS e imagenBotonFuegoDs: carga las imágenes de los botones de fuego seleccionado y deseleccionado.

Métodos Principales:

- dibujarinicio: dibuja como imagen el inicio del juego.
- dibujarDificultad: dibuja como imagen la dificultad.
- dibujarImagenHudMago: dibuja como imagen el hud del Mago.
- dibujarImagenPuntero: dibuja como imagen el puntero.
- esperaEnter: lo utilizamos para que el jugador presione enter.
- inicio: función utilizada para iniciar el juego.
- dibujarEstadoJuego: dibuja las rocas, el mago, y los murciélagos cuando estamos en pausa.
- rondaSiguiente: función utilizada para aumentar la dificultad de la siguiente ronda.
- colisionaHechizoMurcielago: función utilizada para colisionar los hechizos con los murciélagos. Es usado para hechizo fuego y de hielo.
- murcielagosEliminados: función booleana para determinar si todos los murciélagos fueron eliminados.
- actualizarHechizo: función utilizada para que los hechizos se puedan mover, y que no supere los bordes de la pantalla del juego incluyendo el menú.
- presionaEnter: función para indicar si se presionó el enter.
- manejarPausa: función booleana para indicar si se tocó la tecla espacio para pausar.
- salirJuego: función utilizada cuando ganamos o perdemos para salir del juego. Se presiona la tecla escape.
- resetearValores: función utilizada para resetear todos aquellos valores que van cambiando durante el juego.
- ReiniciarJuego: función utilizada para reiniciar el juego cuando se indica que se presione enter para reiniciar el juego.

- actualizarMovimientoMago: función utilizada para determinar los movimientos del mago. El mago no superará los bordes del juego. Además, colisiona con las rocas.
- procesarHechizos: función utilizada para indicar si el hechizo se lanzó, si colisiono o debería volverse null.
- esperarTiempo: función booleana utilizada para
- reiniciar: función utilizada para reiniciar el esperarTiempo.
- dibujarHechizos: función utilizada para dibujar el hechizo hielo y fuego.
- dibujarUI: función para dibujar la interfaz del juego. Dibuja lo siguiente: mago, menu, vida, mana, hud, texto de ronda, texto de cantidad de enemigos eliminados, dibujar el botón de los hechizos y del puntero.
- manejarPantallasDeCarga: función booleana para indicar el manejo de pantallas de carga entre ronda y ronda.
- actualizarPuntero: función utilizada para actualizar el puntero dependiendo del hechizo que nosotros escojamos.
- manejarEntradas: función utilizada para verificar si se presionaron el click izquierdo en los botones de los hechizos.
- actualizarTiempo: función para manejar el tiempo con el tick.
- dibujarEstadoMurcielago: función que se aplica para la persecución de los murciélagos hacia el mago; y se anima la animación de los murciélagos.
- dibujarEscena: función para dibujar el entorno del juego. Se dibuja: el mago, el menu, las rocas, los murciélagos, la interfaz de usuario y los hechizos.
- guardarTiempoPuntaje: función para guardar el tiempo final del juego.
- verificarFinJuego: función para verificar cuando el juego termino.
- controlarFinDeJuego: función para el control del fin del juego. Si el mago muere, quiere decir que no ganó, por lo tanto, se termina el juego.
- procesarMurcielagos: función utilizada para procesar toda la lógica de los murciélagos: colision, actualización de murciélagos.
- actualizarMana: función utilizada para actualizar y aumentar el maná cada vez que se gasta.
- dibujarSelectorDificultad: función para dibujar el selector de dificultad con los botones.
- manejarSelectorDificulad: función para seleccionar una dificultad para definir la cantidad de rondas a jugar.
- Juego(): Utilizado para inicializar la mayoría de los objetos importantes del proyecto.
- tick(): Utilizado para actualizar la lógica del juego en cada frame.
- main: Utilizado para que inicialice el juego.

Problemas/Decisiones:

- La clase Juego actúa como la columna vertebral del proyecto, y uno de los principales problemas que tuvimos fue su sobrecarga de código, especialmente en el método tick() y en la inicialización. Para solucionarlo, comenzamos a reorganizar y separar responsabilidades, delegando ciertas tareas a funciones internas específicas de la clase y otras a clases auxiliares como Funciones_Utiles. Así logramos una mayor limpieza, legibilidad y mantenibilidad del código. No fue un problema técnico complejo, sino más bien una cuestión de estructura y orden del contenido.

Implementación:

Todas las funciones y métodos utilizados en el desarrollo ya fueron explicados brevemente en la sección anterior. Por ende, simplemente se detallan las implementaciones correspondientes.

. Clase Juego:

```
//----------------Inicio------------------------

	public void dibujarinicio() {

		entorno.dibujarImagen( imagenInicio, anchoVentana / 2 + 10, alturaVentana / 2, 0, 0.77);

	}

	//----------------Dificultad------------------------

	public void dibujarDificultad() {

	   this.entorno.dibujarImagen(imagenDificultad, anchoVentana / 2 + 10, alturaVentana / 2, 0, 1.6);

	}

	//----------------Métodos propios para Hud del Mago----------------------

	public void dibujarImagenHudMago(Entorno entorno, int x, int y) {

		entorno.dibujarImagen(imagenMago,x, y, 0, 1);

	}

	//----------------Métodos propios para Puntero----------------------

	public void dibujarImagenPuntero(Entorno entorno, int x, int y) {

		entorno.dibujarImagen(imagenPuntero,x, y, 0, 1);

	}

	//----------------Metodo de dibujo de pausa--------------------

	public void dibujarImagenPausa(Entorno entorno) {

	   entorno.dibujarImagen(imagenPausa, anchoVentana / 2 + 10, alturaVentana / 3, 0, 1.7);

	}

	//

	public void esperaEnter() {
```

```
                esperaEnter = !esperaEnter;// Lo utilizamos para que el jugador presione enter
        }

//--------------------------Inicio de juego-------------------------

        private boolean inicio() {
                dibujarinicio();
                if (!inicioJuego) {//si ya se inicio el juego anteriormente, no pedira volver a esta pantalla
                   if (this.entorno.sePresiono(entorno.TECLA_ENTER) ) {//Presiona enter para iniciar
                        inicioJuego = !inicioJuego;
                            return inicioJuego;
                   }
                        return inicioJuego;
                }
                return true;

        }
//      ------------------------ Funcion de estado del Juego ----------------------------------
(Se retiro parametro enter para texto)

        public void dibujarEstadoJuego() {
           entorno.dibujarImagen(imagenFondo, anchoVentana / 2 + 10, alturaVentana / 2, 0, 1.7);
           // Dibujamos rocas
           for (int i = 0; i < rocas.length; i++) {
              if (rocas[i] != null) {
                 rocas[i].dibujarImagenRoca(entorno);
              }
           }
```

```
            // Dibujar Mago
            this.mago.dibujarImagenMago(entorno);

            // Dibujar Murcielago
            for (int i = 0; i < murcielagos.length; i++) {
                if (murcielagos[i] != null) {
                    murcielagos[i].actualizarAnimacion(this.mago.getX());
                    murcielagos[i].dibujarImagen(entorno);
                }
            }
        }
//        ------------------------ Funcion para aumentar la dificultad de la siguiente ronda ----------------------------------
        public void rondaSiguiente(boolean activar) {// Si recibe true como parametro, ejecutara la siguiente ronda(Se puede modificar)
            if (activar) {
                // Aumentar la dificultad
                this.cantMurcielagoPantalla += 5; // Aummentamos la cantidad de murcielagos en pantalla
                this.cantMurcielagoTotal += 10; // Aumentamos la cantidad total en la ronda
                this.velocidadMurcielago += 1; // Aumentamos la velocidad de los murciélagos
                                                                    //Mayor "DIFICULTAD"
                // Reiniciamos nuestro contador de generados y eliminados(y guardamos nuestro puntaje de enemigos eliminados)
                                this.puntajeMurcielagos += cantMurcielagosEliminados ; // Acumulador para el puntaje de enemigos por pantalla
                this.cantMurcielagoGenerados = 0;
                this.cantMurcielagosEliminados = 0;
                // Generamos un nuevo array de murciélagos con la nueva cantidad total
                this.murcielagos = new Murcielago[this.cantMurcielagoTotal];
```

```
            this.numeroRonda++;
        }
    }
//----------Métodos propios para Colision----------

//      Colision entre Hechizo y murcielago.
    public void colisionaHechizoMurcielago(Murcielago[] murcielagos , int n) {
        if(n == 1) { // Si es 1, hablamos de Hechizo Fuego
            for(int i =0; i < murcielagos.length;i++) { // Recorremos los murcielagos
                if(this.murcielagos[i] != null && this.hechizoFuego != null) { // Si ambos son diferentes de null
                    if(this.hechizoFuego.limiteIzquierdo() < this.murcielagos[i].limiteDerecho() && // Condición de colision
                        this.hechizoFuego.limiteSuperior() < this.murcielagos[i].limiteInferior() &&
                        this.hechizoFuego.limiteInferior() > this.murcielagos[i].limiteSuperior() &&
                        this.hechizoFuego.limiteDerecho() > this.murcielagos[i].limiteIzquierdo()) {
                        this.murcielagos[i] = null; // Que ambos sean null
                        this.hechizoFuego = null;
                        cantMurcielagosEliminados++; // Aumentamos nuestro contador
                    }
                }
            }
        }else { // Si es 2, hablamos de Hielo
            for(int i =0; i < murcielagos.length;i++) { // Recorremos los murcielagos
                if(this.murcielagos[i] != null && this.hechizoHielo != null) { // Si ambos son diferentes de null
```

```
                                        if(this.hechizoHielo.limiteIzquierdo() <
this.murcielagos[i].limiteDerecho() && // Condición de colision
                                                this.hechizoHielo.limiteSuperior() <
this.murcielagos[i].limiteInferior() &&
                                                this.hechizoHielo.limiteInferior() >
this.murcielagos[i].limiteSuperior() &&
                                                this.hechizoHielo.limiteDerecho() >
this.murcielagos[i].limiteIzquierdo()) {
                                                this.murcielagos[i] = null; // Que
ambos sean null
                                                cantMurcielagosEliminados++;//
Aumentamos nuestro contador
                                        }
                                }
                        }
                }
        }
//      ----------------- Murcielagos eliminados Total ----------------------
        private boolean MurcielagosEliminados() {
                for (int i = 0; i < murcielagos.length; i++) {
                        if (murcielagos[i] != null) {
                                return false; // Si almenos hay un solo murcielago, no
retorna nadfa
                        }
                }
                return true; // todos los murcielagos son nulos
        }

//      ----------------------------Generar Hechizo----------------------------------------
        public void actualizarHechizo(int n) {
                if(n == 1) { // Si hablamos de Hechizo Fuego
                        if(this.hechizoFuego != null) {
                                this.hechizoFuego.moverFuego();
```

```
                        if(this.hechizoFuego.limiteIzquierdo() < 0 || // Si supera el
izquierdo
                                this.hechizoFuego.limiteDerecho() > menu.getX() -
menu.getAncho() / 2 ||  //|| // Si supera el derecho
                                this.hechizoFuego.limiteSuperior() < 0 || // Si
supera arriba
                                this.hechizoFuego.limiteInferior() > alturaVentana)
{ // Si supera abajo
                                this.hechizoFuego = null;
                        }
                }
            }else { // Si hablamos de Hechizo Hielo
                if(this.hechizoHielo != null) {
                    this.hechizoHielo.moverHielo();
                    expirarHielo++;
                    if(this.hechizoHielo.limiteIzquierdo() < 0 || // Si supera el
izquierdo
                            this.hechizoHielo.limiteDerecho() > menu.getX() -
menu.getAncho() / 2 ||  //|| // Si supera el derecho
                            this.hechizoHielo.limiteSuperior() < 0 || // Si supera
arriba
                            this.hechizoHielo.limiteInferior() > alturaVentana)
{ // Si supera abajo
                            this.hechizoHielo = null;
                            expirarHielo =0;
                    }
                    if(expirarHielo >= 180) {
                        this.hechizoHielo = null;
                        expirarHielo = 0;
                    }
                }
            }
        }
```

```
//	Para indicar si se presiono enter
	public void PresionarEnter(boolean enter) {
		if(enter) {
			return;
		}
		else {
			enPausa = true;
			System.out.println("ENTER");
		}
	}
	//----------------Cambio de pausa-------------------------
	private boolean manejarPausa() {
	   if (this.entorno.sePresiono(entorno.TECLA_ESPACIO)) {
	      enPausa = !enPausa;
	   }
	   if (enPausa) {
	      if (this.entorno.sePresiono(entorno.TECLA_ESCAPE)) {//Reutilizamos el
reinicio para salir reiniciar desde el Menu
	         resetearValores();
	         System.out.println("reinicio");
	      }
	      dibujarEstadoJuego();
	      dibujarImagenPausa(entorno);
	          return true;

	   }
	   return false;
	}
	private void SalirJuego() {
		if(this.entorno.estaPresionada(entorno.TECLA_ESCAPE)) { //Tecla
Escape
```

```
                    entorno.dispose();
                }
            }
            public void resetearValores() {
                //reiniciamos las variables y objetos a sus valores por default
               this.numeroRonda = 1;
               this.enPausa = false;
               this.esperaEnter = true;
               this.tiempo = 0;
               this.tiempoInicio = -1;
               this.temporizadorEjecutado = false;
               this.tiempoFinal = 0;
               this.puntajesGuardados = false;
               this.salircarga1 = false;
               this.salircarga2 = false;
               this.cantMurcielagoPantalla = 10;
               this.cantMurcielagoTotal = 50;
               this.cantMurcielagoGenerados = 0;
               this.cantMurcielagosEliminados = 0;
               this.puntajeMurcielagos = 0;
               this.hechizoFuego = null;
               this.hechizoHielo = null;
               this.expirarHielo = 0;
               this.murcielagos = new Murcielago[this.cantMurcielagoTotal];

               //Restablecemos los apartados visuales del jugador
                    this.mago = new Mago(this.menu.getAncho(),alturaVentana);
                    this.botonHechizoFuego = new Boton(790,150, 120,
67,imagenBotonHieloS, imagenBotonHieloDS); // Hechizo Fuego
                    this.botonHechizoHielo = new Boton (790,250, 120,
67,imagenBotonFuegoS, imagenBotonFuegoDS); // Hechizo Hielo
```

```
                //variables de Hechizo y Mana
                regenerarMana = 0;
                expirarHielo = 0;
                this.entorno.iniciar(); // Inicia el juego


        }
    private void ReiniciarJuego() {
       if (this.entorno.estaPresionada(entorno.TECLA_ENTER)) {//Si se presiona Enter en la pantalla final del juegop. Podra Reiniciarlo
          resetearValores();
          System.out.println("reinicio");
       }
    }


        //-------------------- Movimiento y Colision Mago-Rocas ----------------------
        private void actualizarMovimientoMago() {

        if(this.entorno.estaPresionada(entorno.TECLA_IZQUIERDA)) { //Movimiento izquierdo
                                        if(mago.getX() - mago.getAncho() / 2 > 0 ) {
                                                this.mago.moverIzquierda();
                                                if(this.mago.colisionaMagoRoca(rocas)) {
                                                        this.mago.moverDerecha();
                                                }
                                        }
                                }

        if(this.entorno.estaPresionada(entorno.TECLA_DERECHA)) { //Movimiento Derecho
                                        if(mago.getX() + mago.getAncho() / 2 < menu.getX() - menu.getAncho() / 2) {
                                                this.mago.moverDerecha();
```

```
                                        if(this.mago.colisionaMagoRoca(rocas)) {
                                                this.mago.moverIzquierda();

                                        }
                                }
                        }

        if(this.entorno.estaPresionada(entorno.TECLA_ARRIBA)) { //Movimiento
Ascendente
                                if(mago.getY() - mago.getAltura() / 2 > 0) {
                                        this.mago.moverArriba();
                                        if(this.mago.colisionaMagoRoca(rocas)) {
                                                this.mago.moverAbajo();
                                        }
                                }
                        }
                        if(this.entorno.estaPresionada(entorno.TECLA_ABAJO))
{ //Movimiento Descendente
                                if(mago.getY() + mago.getAltura() / 2 < 600 ) {
                                        this.mago.moverAbajo();
                                        if(this.mago.colisionaMagoRoca(rocas)) {
                                                this.mago.moverArriba();
                                        }
                                }
                        }



        }
        //-----------------------Metodo para procesar los Hechizos----------------------------
------
        //Indica si el Hechizo se lanzo, si colisiono o si debera volverse null
        private void procesarHechizos() {
```

```
            //-----------------------------------Hechizos---------------------------------

        // Hechizo Fuego
        if (this.botonHechizoFuego.estadoActual()) {
            if (this.entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
                if (Hechizos.permitirDisparar(punteroX, menu)) {
                    if (this.hechizoFuego == null) {
                        this.hechizoFuego = new Hechizos(mago.getX(), mago.getY(),
punteroX, punteroY, 25, 25, "Fuego", 0, 1, true);
                        this.mago.setMana(mago.getMana() -
this.hechizoFuego.getCostoMana());
                    }
                }
            }
        }

        // Hechizo Hielo
        if (this.botonHechizoHielo.estadoActual()) {
            if (this.entorno.sePresionoBoton(entorno.BOTON_IZQUIERDO)) {
                if (Hechizos.permitirDisparar(punteroX, menu)) {
                    if (this.hechizoHielo == null && this.mago.getMana() > 5) {
                        this.hechizoHielo = new Hechizos(punteroX, punteroY, punteroX,
punteroY, 200, 200, "Hielo", 8, 1, false);
                        this.mago.setMana(mago.getMana() -
hechizoHielo.getCostoMana());
                        expirarHielo = 0;
                    }
                }
            }

        }
```

```
            //Sacamos estos metodos de los condicionales, ya que si los dejamos dentro de estos
            //Al lanzar un hechizo y deselecionar el boton de este, el hechizo se congela
        actualizarHechizo(2);
        colisionaHechizoMurcielago(murcielagos, 2);
        actualizarHechizo(1);
        colisionaHechizoMurcielago(murcielagos, 1);
           }
           //-----------------------Metodo para generar una espera-------------------------------
    public boolean esperarTiempo(int tiempoEspera, int tiempoActual) {
           if (temporizadorEjecutado){  // Variable para que no se vuelva a aplicar la funcion a no ser que la reiniciemos
                                                                    // Reiniciar definido mas abajo
                                return false;
                        }
        if (tiempoInicio == -1) {
           tiempoInicio = tiempoActual; // Se guarda el tiempo cuando se llama por primera vez
        }
        //Si mi "tiempoActual" es mayor o igual a el "tiempoInicio"  + el tiempo esperado "tiempoEspera" retornara true
        if (tiempoActual >= tiempoInicio + tiempoEspera) {
           temporizadorEjecutado = true;
           return true;
        }

        return false; // Si no se cumplio el tiempo, sigue pasando el tiempo sin pasar por el primer condicional
    }
    //-----------------------Metodo para reiniciar "esperarTiempo"-------------------------------
```

```
    public void reiniciar() {
        //reinicia tiempo inicio a -1  y el booleano para ejecutar el temporizador
        tiempoInicio = -1;
        temporizadorEjecutado = false;

    }
    //----------------------Metodo para detener el tiempo interno del juego----------
---------

    //----------------------Metodo especifico para dibujar los hechizos----------------
------------
    private void dibujarHechizos() {
        if (this.hechizoFuego != null) {
            this.hechizoFuego.dibujarImagenFuego(entorno);
        }
        if (this.hechizoHielo != null) {
            this.hechizoHielo.dibujarImagenHielo(entorno);
        }
    }
    //----------------------METODO PARA DIBUJAR INTERFAZ DEL JUEGO----------
-----------------------------------
            private void dibujarUI() {

                    this.menu.dibujarImagenMenu(entorno); //Dibujamos el menú

                    this.mago.dibujarVida(entorno, 5+this.menu.getX(),
this.menu.getY()+86); // Dibujamos la vida del Mago
                    this.mago.dibujarMana(entorno, 5+this.menu.getX(),
this.menu.getY()+100);// Dibujamos el manadel Mago
                    this.dibujarImagenHudMago(entorno, 5+this.menu.getX(),
this.menu.getY()+150);// Dibujamos el Hud del Mago
```

```
        entorno.cambiarFont("OCR A Extended", 22, Color.BLACK, entorno.NEGRITA);// Le cambiamos la fuente por una mas estetica

        entorno.escribirTexto("Ronda: " + this.numeroRonda, this.menu.getX()-40, this.menu.getY() + 20);//Mostramos la ronda actual en el Menu


        entorno.cambiarFont("OCR A Extended", 18, Color.BLACK, entorno.NEGRITA);// Le cambiamos la fuente por una mas estetica

        entorno.escribirTexto("" + cantMurcielagosEliminados, this.menu.getX()+ 20, this.menu.getY() + 70);//Mostramos los murcielagos eliminados por pantallas de todas las rondas


    // Dibujar el boton del Hechizo Fuego
                    if(this.botonHechizoFuego.estadoActual()) {
                            this.botonHechizoFuego.dibujarnImagenBoton(entorno);
                    }else {
                            this.botonHechizoFuego.dibujarnImagenBoton(entorno);
                    }


    // Dibujar el boton del Hechizo Hielo
                    if(this.botonHechizoHielo.estadoActual()) {
                            this.botonHechizoHielo.dibujarnImagenBoton(entorno);
                    }else {
                            this.botonHechizoHielo.dibujarnImagenBoton(entorno);
                    }
            // Dibujar Puntero
               dibujarImagenPuntero(entorno, entorno.mouseX(), entorno.mouseY());


            }
            //------------------- Pantallas de Carga Entre Ronda y Ronda --------------------
            private boolean manejarPantallasDeCarga() {
               if (numeroRonda == 1 && !salircarga1) {
                  if (esperarTiempo(tiempoCarga, tiempo)) {// Tiempo de finalizacion de pantalla de carga
```

```
                salircarga1 = true;
                reiniciar();
            } else {
                dibujarEstadoJuego();
                this.entorno.dibujarImagen(imagenRonda1, anchoVentana / 2 + 10,
alturaVentana / 2, 0, 1.7);
                return true; // Sigue mostrando pantalla de carga
            }
        }

        if (numeroRonda == 2 && !salircarga2) {
            if (esperarTiempo(tiempoCarga, tiempo)) {
                salircarga2 = true;
            } else {
                dibujarEstadoJuego();
                this.entorno.dibujarImagen(imagenRonda2, anchoVentana / 2 + 10,
alturaVentana / 2, 0, 1.7);
                return true; // Sigue mostrando pantalla de carga
            }
        }

        return false; // No hay pantalla de carga activa
    }
    //---------------------------------------- PUNTERO Y HUD ----------------------------
-------------
    private void actualizarPuntero() {// Puntero "Fuego"
        if (botonHechizoFuego.estadoActual()) {
            imagenPuntero = Herramientas.cargarImagen(punteroImagenes[1]);
            tipopuntero = 1;
        } else if (botonHechizoHielo.estadoActual()) {// Puntero "Hielo"
            imagenPuntero = Herramientas.cargarImagen(punteroImagenes[2]);
            tipopuntero = 2;
```

```
        } else {// Puntero normal
            imagenPuntero = Herramientas.cargarImagen(punteroImagenes[0]);
            tipopuntero = 0;
        }
    }
    //---------------------------------------- PUNTERO Y HUD ----------------------------
--------------
    private void manejarEntradas() {//Metodo para verificar cuando se presionaron
los botones del Mouse
            // Obtenemos los valores del punteroX y el PunteroY propios del mouse
        punteroX = entorno.mouseX();
        punteroY = entorno.mouseY();

        if (entorno.seLevantoBoton(entorno.BOTON_IZQUIERDO)) {//Condicional
al levantar el Boton

            if (botonHechizoFuego.estaDentro(punteroX, punteroY)) {//Si esta dentro
del area del boton(Boton Fuego)
                botonHechizoFuego.cambiarEstado();
                if (botonHechizoHielo.estadoActual())
botonHechizoHielo.cambiarEstado();
            } else if (botonHechizoHielo.estaDentro(punteroX, punteroY)) {//Si esta
dentro del area del boton(Boton Hielo)
                botonHechizoHielo.cambiarEstado();
                if (botonHechizoFuego.estadoActual())
botonHechizoFuego.cambiarEstado();
            }
        }

        actualizarPuntero();//llamamos a nuestro metodo para modificar el puntero
    }
    //---------------------------------------- TIEMPO DEL JUEGO ------------------------
-----------------
```

```
	private void actualizarTiempo() {

		//Manejamos el tiempo con el tick ya que el tiempó no es preciso(por lo que vi,, es por los bucles internos tambien)

		if(this.entorno.numeroDeTick() % 100 == 1) {// Dividimos la cantidad de ticks y cuanto el resto sea exactamente 1 aumenta el "tiempo"

			tiempo++;

			System.out.println(tiempo);

		}

	}

	//---------------------------------------- METODO PARA DIBUJAR EL ESTADO ACTUAL DEL MURCIELAGO----------------------------------------

	private void dibujarEstadoMurcielago() {

		//	-------------------------Persecusión Murcielago----------------------------

			for (int i = 0; i < murcielagos.length; i++) {

				if(this.murcielagos[i] != null) { // Si el murcielago es diferente de null, lo persigue

					Funciones_utiles.seguimientoMurcielago(mago, murcielagos[i]); // Actualiza posición del murcielago actual



this.murcielagos[i].actualizarAnimacion(this.mago.getX()); // Actualizamos la animacion del murcielago

			this.murcielagos[i].dibujarImagen(entorno); // Dibujamos al Murcielago

				}

			}

	}

	//---------------------------------------- METODO PARA DIBUJAR EL ENTORNO DEL JUEGO----------------------------------------

	 private void dibujarEscena() {

			this.entorno.dibujarImagen(imagenFondo, anchoVentana / 2 + 10, alturaVentana / 2, 0, 1.7);
```

```
                        //------------------------------ Mago -------------------------
                                this.mago.dibujarImagenMago(entorno);
//dibujamos al mago

                        //------------------------------ Menu -------------------------
                                this.menu.dibujarImagenMenu(entorno);
//Dibujamos el menú


                        //------------------------------ Rocas -------------------------
                                for (int i = 0; i < rocas.length;i++) { //Dibujamos a
las Rocas
                                        rocas[i].dibujarImagenRoca(entorno);
                                }
                        //------------------------------ Murcielagos-------------------------
                                dibujarEstadoMurcielago();
                        //------------------------------ Interfaz de Usuario-----------------------
--
                                dibujarUI();
                        //------------------------------ Hechizos-------------------------
                                dibujarHechizos();


        }
         //---------------------------------------- METODO PARA GUARDAR EL
TIEMPO FINAL DEL JUEGO-----------------------------------------
        private void guardarTiempoPuntaje() {
                if (!puntajesGuardados) {      //Sin este condicional, sigue aumentando el
tiempoFinal
                        tiempoFinal = tiempo; // Guardamos el tiempo para mostralo al
final
                        puntajesGuardados = true;
                        puntajeMurcielagos += cantMurcielagosEliminados ;
                        System.out.println("Tiempo Guardado");
                }
```

```
	}

	 //--------------------------------------- METODO PARA VERIFICAR QUE TERMINO EL JUEGO----------------------------------------

	private void verificarFinJuego() {

		//-----------------Condicion para finalizar la Ronda/Juego-----------------------

		if (MurcielagosEliminados() && cantMurcielagoGenerados == cantMurcielagoTotal)

		{

			if (numeroRonda==rondaFinal) {//Indicamos la ronda final

				guardarTiempoPuntaje();//Guardamos el tiempo y la cantidad de murcielaghos eliminados total

				dibujarEstadoJuego();

				PantallaFinJuegoGana.dibujarImagenFinJuego(entorno);

			entorno.cambiarFont("Arial", 38, Color.BLACK, entorno.NEGRITA);// Le cambiamos la fuente por una mas estetica

			entorno.escribirTexto("" + tiempoFinal, this.PantallaFinJuegoGana.getX()+ 200, this.PantallaFinJuegoGana.getY() - 30);

			entorno.escribirTexto("" + puntajeMurcielagos, this.PantallaFinJuegoGana.getX()+ 200, this.PantallaFinJuegoGana.getY() + 15);

			SalirJuego();

			ReiniciarJuego();

			    return;

			}

			else {

				rondaSiguiente(true); // Si no es la ronda final, aumenta la dificultad

			}

		}

	}

	//---------------Funcion para el control del juego-PAUSA-FIN DEL JUEGO(GANAR/PERDER)-RONDAS

	private void controlarFinDeJuego() {
```

```
                if (mago.estaMuerto()) { //si el mago esta sin vida, mostramos pantalla fin de juego perdido

                        guardarTiempoPuntaje();//Guardamos el tiempo y la cantidad de murcielaghos eliminados total

                    dibujarEstadoJuego();

                    PantallaFinJuegoPierde.dibujarImagenFinJuego(entorno);

                entorno.cambiarFont("Arial", 38, Color.BLACK, entorno.NEGRITA);// Le cambiamos la fuente por una mas estetica

                this.entorno.escribirTexto("" + tiempoFinal, this.PantallaFinJuegoPierde.getX()+ 200, this.PantallaFinJuegoPierde.getY() - 30);//Mostramos los murcielagos eliminados por pantallas de todas las rondas

                this.entorno.escribirTexto("" + puntajeMurcielagos, this.PantallaFinJuegoPierde.getX()+ 200, this.PantallaFinJuegoPierde.getY() + 25);

                SalirJuego();

                ReiniciarJuego();

                return;

                   }

        }

         //---------------------------------------PROCESA TODA LA LOGICA DE LOS MURCIELAGOS----------------------------------------

         //Esto realiza: Colision de Mago/murcielago, Actualiza posicion y Estado Murcielago, Generacion de Murcielagos, Contador total

        private void procesarMurcielagos() {

                        //----------------Colision entre Mago y Murciélago----------------------------

                        int ColisionesMurcielagos=Funciones_utiles.colisionMagoMurcielago(mago, murcielagos); // Esto va indicando si el murcielago colisiona con mago

                        // Si es así, entonces el murcielago será null

                        //                  ----------------Restablecer Murcielagos----------------------

                        cantMurcielagoGenerados = Funciones_utiles.actualizarMurcielagos(

                                        murcielagos, // Array de murcielagos

                                        cantMurcielagoPantalla, // Cantidad en pantalla
```

```
                                cantMurcielagoGenerados, // Cantidad de generados
                                cantMurcielagoTotal, // Cantidad total
                                anchoVentana, // Ancho de ventana del juego
                                alturaVentana, // Alto de ventana del juego
                                velocidadMurcielago, // Velocidad de murcielago
                                menu, // Y el propio menú
                                numeroRonda//valor de nuestra ronda
                            );

                    //-----------------Colision entre Mago y Murciélago = Perder vida----------
                    for(int i = 0; i < murcielagos.length;i++) { // recorremos los murcielagos
                            if(this.murcielagos[i] != null) { //  Preguntamos si son distintos de null
                                    Funciones_utiles.colisionMagoMurcielago(mago, murcielagos); // nos vamos a la colision
                            }// Ademas de que el elemento murcielago sea null, el mago perderá vida
                    }
//                  //-----------------Contador de murcielagos totales eliminados----------
                                    cantMurcielagosEliminados += ColisionesMurcielagos;//Contador que definira si se termina la ronda/juego
                            }
        //---------------------------------------ACTUALIZA EL MANA DEL JUGADOR----------------------------------------

        public void actualizarMana() {
                // Regeneramos el mana
                regenerarMana ++; // Constantemente irá subiendo
                if(regenerarMana >= 60) { // Hasta que se llegue a 60 ticks --> 1 segundo
                        if(this.mago.getMana() < 10) { // Si el mana del mago es menor a 10
```

```
                                this.mago.setMana(this.mago.getMana()+1); // Se aumenta el mana +1
                        }
                        regenerarMana = 0; // Si no es así, vuelve a 0
                }
        }
        //Metodo para  dibujar el selector de Dificultad con los botones
        public void dibujarSelectorDificultad() {
                dibujarDificultad();
                this.botonFacil.dibujarnImagenDificultad(entorno, 1.5);
           this.botonNormal.dibujarnImagenDificultad(entorno, 1.5);
           this.botonDificil.dibujarnImagenDificultad(entorno, 1.5);
        }
         //----------------------------------------LOGICA DEL SELECTOR DE DIFICULTAD----------------------------------------
        public void manejarSelectorDificultad() {//Se seleciona una dificultad para definir la cantidad de rondas a jugar
           int mouseX = entorno.mouseX();
           int mouseY = entorno.mouseY();
                //
           if (entorno.seLevantoBoton(entorno.BOTON_IZQUIERDO)) {//Condicional al levantar el Boton
              if (botonFacil.estaDentro(mouseX, mouseY)) {
                 rondaFinal = 1;
                 seleccionDificultad = true;
                 System.out.println("Dificultad Fácil");
              } else if (botonNormal.estaDentro(mouseX, mouseY)) {
                 seleccionDificultad = true;
                 rondaFinal = 2;
                 System.out.println("Dificultad Normal");
             // } else if (botonDificil.estaDentro(mouseX, mouseY)) {//Modo Dificil "bloqueado"
```

```
                //     seleccionDificultad = true;
                //rondaFinal = 3;
                //System.out.println("Dificultad Difícil");
            }
        }
    }

    Juego()
    {
        //instanciamos los botones de dificultad
        this.botonFacil = new Boton(450, 250, 200, 50, imagenBtnFacil, imagenBtnFacil);
        this.botonNormal = new Boton(450, 330, 200, 50, imagenBtnNormal, imagenBtnNormal);
        this.botonDificil = new Boton(450, 410, 200, 50, ImagenBtnDificil, ImagenBtnDificil);
        // Inicializa el objeto entorno
        this.entorno = new Entorno(this, "Proyecto para TP", anchoVentana, alturaVentana);

        // Inicializamos el menu
        this.menu = new Menu(anchoVentana,alturaVentana);

        // Inicializamos al mago
        this.mago = new Mago(this.menu.getAncho(),alturaVentana);

        // Inicializamos las pantallas
        this.PantallaFinJuegoGana = new PantallaFinJuego(anchoVentana, alturaVentana, 1); // Pantalla Juego Gana
        this.PantallaFinJuegoPierde = new PantallaFinJuego(anchoVentana, alturaVentana, 2); // Pantalla Juego Pierde

        // Inicializamos los botones del Hechizo
```

```
		this.botonHechizoFuego = new Boton(790,150, 120, 67, imagenBotonHieloS, imagenBotonHieloDS); // Hechizo Fuego
		this.botonHechizoHielo = new Boton (790,250, 120, 67,imagenBotonFuegoS, imagenBotonFuegoDS); // Hechizo Hielo

		// Regenerar Mana
		regenerarMana = 0;

		// Expirar el Hechizo Hielo
		expirarHielo = 0;



		// Inicia el juego!
		this.entorno.iniciar();
	}

	// Metodo tick para cada instante del juego
	public void tick()
	{
		// Funcion de inicio de juego para ejecutarse una sola vez(Solo muestra pantalla de inicio y espera ENTER)
		if (!inicio()) {
		   return;
		}
		// Funcion para seleccionar dificultad despues del inicio
		if (!seleccionDificultad) {
			dibujarSelectorDificultad();
		   manejarSelectorDificultad();

			System.out.println("select D");
		   return;
		}
```

```
            //Funcion para aplicar pausa(Tecla ESPACIO)
            if (manejarPausa()) {
                    return;
            }
            //Inicia el juego  con el contador de tiempo
        actualizarTiempo();
            //Pantalla de cargas entre Rondas
            if (manejarPantallasDeCarga()) {
               return; // se está mostrando la pantalla de ronda
            }
            //Controla el si el jugador a Perdido el juego
        controlarFinDeJuego();
        if (mago.estaMuerto()) {
               return;
            }
        //Realizara el nuevo dibujo del mago al moverse
            actualizarMovimientoMago();
            //Funcion para actualizar el mana que utilizo el jugador
            actualizarMana();
            //Funcion para procesar el estado y contador de Murcielagos
            procesarMurcielagos();
            //Funcion para el manejo de entrada del click dek jugador
        manejarEntradas();
        //Funcion de dibujo de objetos estaticos Y dinamicos, como la la interfaz(UI)
y los Hechizos
        dibujarEscena();
        //Proocesamos la logica de los hechizos y su estado
            procesarHechizos();
            //Verificamos si se cumplen las condiciones para finalizar la
Ronda/Juego
            verificarFinJuego();
```

```
    }
```

. Clase Mago:

```
public Mago(int anchoMenu, int alturaVentana) {
                this.ancho = 30;
                this.altura = 30;
                this.x = anchoMenu + anchoMenu/2 ; // 225 + 112.5 = 338 aprox
                this.y = (alturaVentana/2) - this.altura/2; // (600/2) - (30/2) --> 300 - 15 =
285
                this.velocidad = 5;
      this.imagen = Herramientas.cargarImagen("imagenes\\\\caminar_abajo2.png");
      this.vida = 10;
      this.mana = 10;
        }

//  -----------Metodo de animacion ---------(Reutilizar en otros objetos como murcielagos
para darle animacion)
// tener en cuenta variables: int contadorFrames, int velocidadFrames, int
frameDireccion, String[] movimientDireccionMago

      private int animarMovimiento(String[] frames, int frameActual) {
          contadorFrames++; // // aumentamos un contador de frames para generar
delay entre frame y frame

          if (contadorFrames >= velocidadFrames) {// cuando llegue a 5 se aplicara el
siguiente frame
              this.imagen = Herramientas.cargarImagen(frames[frameActual]);
//cargamos el frame desde el array de imagenes "movimientoAbajoMago"
              frameActual++; // aumenta frame
```

```
            if (frameActual >= frames.length) {// si el Frame es  mayor o igual a el tamaño total de nuestro array de frames, volvera a 0
                frameActual = 0; // reiniciamos nuestro frame actual para volver al inicio(un bucle)
            }

            contadorFrames = 0; // reiniciamos el delay entre frame y frame
        }

        return frameActual; // nos devolvera el frame actual
    }

//---------------------- GETTERS Y SETTERS ----------------------------
    // X
    public int getX() {
        return x;
    }
    public void setX(int x) {
        this.x = x;
    }

    // Y
    public int getY() {
        return y;
    }
    public void setY(int y) {
        this.y = y;
    }

    // Ancho
    public int getAncho() {
```

```
        return ancho;
    }
    public void setAncho(int ancho) {
        this.ancho = ancho;
    }

    // Altura
    public int getAltura() {
        return altura;
    }
    public void setAltura(int altura) {
        this.altura = altura;
    }
    // Vida
    public int getVida() {
        return vida;
    }
    public void setVida(int vida) {
        this.vida = vida;
    }

    // Mana
    public int getMana() {
        return mana;
    }
    public void setMana(int mana) {
        this.mana = mana;
    }

//------------------------          DIBUJAR MAGO --------------------
```

```
	public void dibujar(Entorno entorno) {
		entorno.dibujarRectangulo(this.x, this.y, this.ancho, this.altura,0,
Color.BLUE);
	}
	// Dibujar imagen Mago
	public void dibujarImagenMago(Entorno entorno) {
		entorno.dibujarImagen(this.imagen, this.x, this.y, 0, 3);
	}

//---------------------- BORDES DEL MAGO ----------------------------
	public int limiteSuperior() {
		return this.y - this.altura/2 ;
	}
	public int limiteInferior() {
		return this.y + this.altura/2;
	}
	public int limiteIzquierdo() {
		return this.x - this.ancho/2;
	}
	public int limiteDerecho() {
		return this.x + this.ancho/2;
	}

// ---------------------------- COLISION MAGO ROCA --------------------------
	public boolean colisionaMagoRoca(Roca[] rocas) {
		for(int i =0; i < rocas.length;i++) {
			if (this.limiteIzquierdo() < rocas[i].limiteDerecho() &&
				this.limiteSuperior() < rocas[i].limiteInferior() &&
				this.limiteInferior() > rocas[i].limiteSuperior() &&
				this.limiteDerecho() > rocas[i].limiteIzquierdo()) {
				return true; // Si colisiona es true
```

```
                }
            }
            return false; // sino, no colisiona
        }

// -------------------------- MOVIMIENTOS ----------------------------
        // En cada movimiento, llamamos a nuestro animarMovimiento para cambiar de frame.

        public void moverDerecha() { // Movimiento Derecha
            this.x = x + velocidad;
           frameDerecha = animarMovimiento(movimientoDerechaMago, frameDerecha); //animacion derecha
        }

        public void moverIzquierda() { // Movimiento izquierda
            this.x = x - velocidad;
           frameIzquierda = animarMovimiento(movimientoIzquierdaMago, frameIzquierda); //animacion derecha
        }

        public void moverArriba() {  // Movimiento Arriba
            this.y = y - velocidad;
           frameArriba = animarMovimiento(movimientoArribaMago, frameArriba); //animacion arriba
        }

        public void moverAbajo() { // Movimiento Abajo
           this.y = y + velocidad;
           frameAbajo = animarMovimiento(movimientoAbajoMago, frameAbajo);//animacion arriba
        }
```

```
// ---------------------------------- VIDA Y MANA ----------------------------------------
        // Detecta si murio el Mago
        public boolean estaMuerto() {
                if(getVida() <= 0) {
                        return true;
                }else {
                        return false;
                }
        }

        // Dibuja la vida
        public void dibujarVida (Entorno entorno, int x , int y) {
                entorno.dibujarRectangulo(x, y, vida*12, 10, 0, Color.RED);
        }



// Dibujar el mana
        public void dibujarMana (Entorno entorno, int x, int y) {
                entorno.dibujarRectangulo(x, y, mana*12, 10, 0, Color.BLUE);
        }
}
```

. Clase Murcielago:

```
        // Constructor
        public Murcielago(int x, int y,  String[] framesDerecha, String[]
framesIzquierda) {
                this.x = x;
                this.y = y;
                this.ancho = 20;
                this.altura = 20;
```

```
            //definimos para nuestro murcielago los array bde frames para cada lado
            this.movimientoIzquierdaMurcielago = framesIzquierda;
            this.movimientoDerechaMurcielago = framesDerecha;

            //cargamos las imagenes rercibidas para cada lado
            this.imagen = Herramientas.cargarImagen(framesDerecha[0]);
            this.imagen = Herramientas.cargarImagen(framesIzquierda[0]);
        }

//--------------------------- GETTERS Y SETTERS ---------------------------
        // X
        public int getX() {
            return x;
        }
        public void setX(int x) {
            this.x = x;
        }

        // Y
        public int getY() {
            return y;
        }
        public void setY(int y) {
            this.y = y;
        }

//------------------ Actualizar Animacion ----------------------------------
        public void actualizarAnimacion(int posicionMagoX) {
           String[] frames;  // Nuestro array de imagenes
```

```
                if (this.x > posicionMagoX) { // Si está a la derecha el mago, mira para la
izquierda
                        frames = movimientoIzquierdaMurcielago;
                } else { // Si está a la izquierda el mago, mira para la derecha
                        frames = movimientoDerechaMurcielago;
                }
                contadorFrames++; // Aumentamos frames
                if (contadorFrames >= velocidadFrames) { // Si el contador es mayor a la
velocidad
                        this.imagen = Herramientas.cargarImagen(frames[frameActual]);
// entonces que cargue la imagen
                        frameActual++; // Aumentamos los frames
                        if (frameActual >= frames.length) {
                           frameActual = 0;
                        }
              contadorFrames = 0;  // reiniciamos los frames
                }
        }

        // Método de dibujo para el murcielago
        public void dibujarImagen(Entorno entorno) {
           entorno.dibujarImagen(this.imagen, this.x, this.y, 0,2);
        }

// ---------------------- BORDES DEL MURCIELAGO -----------------------
        public int limiteSuperior() {
                return this.y - this.altura/2 ;
        }
        public int limiteInferior() {
                return this.y + this.altura/2;
        }
        public int limiteIzquierdo() {
```

```
            return this.x - this.ancho/2;
        }
        public int limiteDerecho() {
            return this.x + this.ancho/2;
        }
}
```

. Clase Menu:

```
//      Constructor
        public Menu(int anchoVentana, int alturaVentana){ //AnchoVentana = 900 ; alturaVentana = 600
            this.ancho = (anchoVentana - anchoVentana/2) / 2; // (900-450) / 2 = 225
            this.altura = alturaVentana; // 600
            this.x = anchoVentana - this.ancho /2 ; // sería 900 - (225 /2) = 900-112.5 = 788
            this.y = alturaVentana / 2; // sería 600/2 = 300
      this.imagen = Herramientas.cargarImagen("imagenes\\\\menu.png");
        }

//      Dibujar imagen
        public void dibujarImagenMenu(Entorno entorno) {
            entorno.dibujarImagen(this.imagen,this.x, this.y, 0, 3);
        }

//------------------------- GETTERS Y SETTERS ----------------------
        // X
        public int getX() {
            return x;
        }
        public void setX(int x) {
```

```
		this.x = x;
	}

	// Y
	public int getY() {
		return y;
	}
	public void setY(int y) {
		this.y = y;
	}

	// Altura
	public int getAltura() {
		return altura;
	}
	public void setAltura(int altura) {
		this.altura = altura;
	}

	// Ancho
	public int getAncho() {
		return ancho;
	}
	public void setAncho(int ancho) {
		this.ancho = ancho;
	}
}
```

. Clase Boton:

```
	// Constructor
	public Boton(int x, int y,int ancho, int altura, Image imagenA,Image imagenB )
{
```

```
            this.x = x;
            this.y = y;
            this.ancho = ancho;
            this.altura = altura;
            this.seleccionado = false;
            this.imagenS = imagenA;
            this.imagenDS = imagenB;
        }

        // Funcion para saber si el Mouse está dentro del botón
        public boolean estaDentro(int mouseX, int mouseY) {
            return mouseX >= x - ancho / 2 &&
                    mouseX <= x + ancho / 2 &&
                    mouseY >= y - altura / 2 &&
                    mouseY <= y + altura / 2;
        }

//-------------------------- ESTADO DEL BOTÓN ----------------------
        public boolean estadoActual() {
            return this.seleccionado;
        }
        public void cambiarEstado() {
           this.seleccionado = !this.seleccionado;
        }
        public void dibujarnImagenDificultad(Entorno entorno, double escala) {
                    entorno.dibujarImagen(this.imagenDS, this.x, this.y, 0, escala);
        }
//-------------------- DIBUJAR IMAGEN DEL BOTÓN ----------------------------
        public void dibujarnImagenBoton(Entorno entorno) {
            if(this.estadoActual()){
```

```
                entorno.dibujarImagen(this.imagenDS, this.x, this.y, 0, 2.5);
            }
            else {
                entorno.dibujarImagen(this.imagenS, this.x, this.y, 0, 2.7);
            }
        }
}
```

. Clase Hechizos:

```
public Hechizos(int xInicio, int yInicio,
                                    int xDestino, int yDestino, int ancho,int altura,
                                    String nombre, int costoMana, int angulo, boolean
tieneTrayectoria) {

            this.x = xInicio;
        this.y = yInicio;
        this.ancho = ancho;
        this.altura = altura;
            this.nombre = nombre;
            this.costoMana = costoMana;
            this.angulo = angulo;

        // Cargamos las imagenes a nuestra clasecon "cargarImagenes" con el array de
movimientoHechizo
        this.imagenesFuego = cargarImagenes(movimientoHechizoFuego);
        this.imagenesHielo = cargarImagenes(movimientoHechizoHielo);

        if (tieneTrayectoria) {
            calcularTrayectoria(xInicio, yInicio, xDestino, yDestino);
        }
```

```
        // Verifica que el contendido concida con la clase y le asigna las imagenes corrspondientes
        if (nombre.equals("Fuego")) {
            this.imagen = imagenesFuego[0];
        } else if (nombre.equals("Hielo")) {
            this.imagen = imagenesHielo[0];
        }


        }

        // Cargar Imagenes
        private Image[] cargarImagenes(String[] nombresImagenes) {
                //creamos un array de imagenes con el mismo tamaño que nuestreas imagenes
            Image[] imagenes = new Image[nombresImagenes.length];
            //reccoremos el array
            for (int i = 0; i < nombresImagenes.length; i++) {
                //cargamos cada imagen en nuestro nuevo array
               imagenes[i] = Herramientas.cargarImagen(nombresImagenes[i]);
            }
            //retornamos un nuevo array con las imagenes cargadas
            return imagenes;
        }

        // Animar Movimiento de Hechizos
        private int animarMovimiento(Image [] frames, int frameActual) {
            contadorFrames++; // // aumentamos un contador de frames para generar delay entre frame y frame

            if (contadorFrames >= velocidadFrames) {// cuando llegue a 5 se aplicara el siguiente frame
```

```
            this.imagen = frames[frameActual]; //cargamos el frame desde el array de imagenes "movimientoAbajoMago"
            frameActual++; // aumenta frame

            if (frameActual >= frames.length) {// si el Frame abajo es  mayor o igual a el tamaño total de nuestro array de frames, volvera a 0
                frameActual = 0; // reiniciamos nuestro frame actual para volver al inicio(un bucle)
            }
            contadorFrames = 0; // reiniciamos el delay entre frame y frame
        }
        return frameActual; // nos devolvera el frame actual
    }

    // Get costo Mana
    public int getCostoMana() {
            return costoMana;
    }

    // Permite disparar si el mouse se encuentra dentro de la pantalla del juego, sino no puede disparar
    public static boolean permitirDisparar(int xPuntero, Menu menu) {
            return xPuntero < menu.getX() - menu.getAncho() / 2;
    }

    //      Dibujar imagen Hechizo Fuego
    public void dibujarImagenFuego(Entorno entorno) {
            entorno.dibujarImagen(this.imagen, this.x, this.y, this.angulo, 2);
    }

    //      Dibujar imagen Hechizo Hielo
    public void dibujarImagenHielo(Entorno entorno) {
```

```
                entorno.dibujarImagen(this.imagen, this.x, this.y, 0, 2);
        }

//-------------------- TRAYECTORIA DE LOS HECHIZOS ----------------------
        private void calcularTrayectoria(int xInicio, int yInicio, int xFin, int yFin) {
    int dx = xFin - xInicio;
    int dy = yFin - yInicio;
    int distanciaCuadrada = dx * dx + dy * dy;
    if (distanciaCuadrada > 0) {
        int distancia = (int) Math.sqrt(distanciaCuadrada);
        int escala = 10; // medida para normalizar la velocidad en los 2 ejes
        this.vx = dx * escala / distancia;
        this.vy = dy * escala / distancia;

            //------CODIGO PARA ANIMACION DE SPRITES------
            // toma 2 puntos, calcula los grados (de rotación) entre X y el vector(dx, dy)
            // y pasamos los valores a condicionales para cada valor de rotar imagen

        // se calcula el ángulo en radianes desde el eje "X" y el punto definido por (dx, dy).
        // de esta forma sacamos el angulo que representa la direccion desde el mago hacia el mouse
        double anguloRad = Math.atan2(dy, dx);

        // convertimos radianes a grados , lo caul nos deja valores decimales
        double anguloEnGrados= Math.toDegrees(anguloRad);

        // Redondear a entero más cercano y castearlo a int
        int anguloGrados = (int) Math.round(anguloEnGrados);
```

```
        anguloGrados = (anguloGrados + 359) % 359;  //pasamos los valores en negativos(sumamos 360 a el valor "negativo", y solo lo que quede del resto sera el valor)


        // grados posibles tanteados con condicionales: 0°, 22.5°, 45°, 67.5°, 90°, 112.5°, 135°, 157.5°, 180°, 202.5°, 225°, 247.5°, 270°, 292.5°, 315°, 337.5°
        // sumamos 11 a nuestros grados apra redondear , ya que cada sector va de 22.5 en 22.5 osea  + 11
        // dividimos por el tamaño de nuestro sector ( / 22)
        //redondeamos el valor y quitamos decimales obteniendo el resto %16
        int indiceSector = ((anguloGrados + 11) / 22) % 16;

        double[] direcciones = {4.75, 5.0, 5.25, 0.25, 0, 0.75, 1.0, 1.25,
                1.5, 1.75, 2.25, 2.5, 3.25, 3.75, 4.0, 4.5};



        this.direccion = direcciones[indiceSector];
        this.angulo = direccion;
    }
}

//---------- MOVER HECHIZO ----------
public void moverFuego() {
    this.x += vx;
    this.y += vy;

    frame = animarMovimiento(imagenesFuego, frame);
}

public void moverHielo() {
    this.x += vx;
    this.y += vy;
```

```
        frame = animarMovimiento(imagenesHielo, frame);
    }

// ---------------------- BORDES DEL HECHIZO ----------------------
    public int limiteSuperior() {
                return this.y - this.altura/2 ;
        }
        public int limiteInferior() {
                return this.y + this.altura/2;
        }
        public int limiteIzquierdo() {
                return this.x - this.ancho/2;
        }
        public int limiteDerecho() {
                return this.x + this.ancho/2;
        }
}
```

. Clase Funciones_Utiles:

```
public class Funciones_utiles {


//--------------------------- COLISION MAGO MURCIELAGO ------------------------------
----
        public static int  colisionMagoMurcielago(Mago mago, Murcielago[]
murcielagos) {
            int colisiones = 0;
                for (int i = 0; i < murcielagos.length; i++) {
              if (murcielagos[i] != null) {
                if (mago.limiteIzquierdo() < murcielagos[i].limiteDerecho() &&
                   mago.limiteSuperior() < murcielagos[i].limiteInferior() &&
                   mago.limiteInferior() > murcielagos[i].limiteSuperior() &&
```

```
                    mago.limiteDerecho() > murcielagos[i].limiteIzquierdo()) {
                    murcielagos[i] = null;  // Convierte el elemento murcielago en null
                    mago.setVida(mago.getVida()-1);
                    colisiones++;
                }
            }
        }
    return colisiones; // Salimos después de la primera colisión, si solo querés borrar uno
    }
//------------------------------ CALCULA SEGUIMIENTO DE MURCIELAGO ------------------------------
    public static void seguimientoMurcielago (Mago mago, Murcielago murcielagos) {
            if (murcielagos != null) { // Si es diferente de null
                    int dx = mago.getX() - murcielagos.getX(); // Obtenemos distancia en X entre el Mago y Murcielago
                    int dy = mago.getY() - murcielagos.getY(); // Obtenemos distancia en Y entre el Mago y Murcielago
                    int distanciaCuadrada = dx * dx + dy * dy; // Elevamos al cuadrado las distancias
                    if (distanciaCuadrada > 0) {   // Condicional para verificar que la distancia no sea 0.
                                                                // Si fuera 0 significa que el murcielago está encima del mago.
                                                                //
Posible uso aplicable a daño recibido al mago

                            int distancia = (int) Math.sqrt(distanciaCuadrada); // Ahora sacamos la raiz de la variable distanciaCuadrada
                            int escalaM = 4; // Con la escala de movimiento definimos la velocidad a la que se movera el Murcielago
                                                        // Toma como la relación de distancia entre 2 ejes, si uno aumenta el otro también.
```

```
                                // Normalizamos la velocidad multiplicandola por nuestra escala y dividiendola por distancia ("Hipotenusa")
                                int vx = dx * escalaM / distancia;// Obtenemos la velociad en X.
                                int vy = dy * escalaM/ distancia;// Obtenemos la velociad en Y.

                                // Actualizamos la posicion de nuestro murcielago con getters y setters de X e Y
                                murcielagos.setX(murcielagos.getX() + vx);
                                murcielagos.setY(murcielagos.getY() + vy);
                        }
                }
        }
//---------------------------------- OBTENER IMAGENES ----------------------------------------
/* Funcion para obtener las imagenes de los enemigos dependiendo del numero de ronda para cada enemigo correspondiente.
 * Al pasar de ronda se actualiza el sprite.
 */

        // Recibe el numero de ronda y retorna un array de enemigos de su respectiva ronda.
        public static String[] obtenerSpritesDerechaPorRonda(int ronda) {
                if (ronda >= 2) {
                        return new String[]{
                                "imagenes\\\\ojobatDerecha1.png",
                                "imagenes\\\\ojobatDerecha2.png",
                                "imagenes\\\\ojobatDerecha3.png",
                                "imagenes\\\\ojobatDerecha4.png"
                        };
                } else {
                        return new String[]{
```

```
                "imagenes\\\\Murcielago_derecha1.png",
                "imagenes\\\\Murcielago_derecha2.png",
                "imagenes\\\\Murcielago_derecha3.png",
                "imagenes\\\\Murcielago_derecha4.png"
            };
        }
    }

    public static String[] obtenerSpritesIzquierdaPorRonda(int ronda) {
        if (ronda >= 2) {
            return new String[]{
                "imagenes\\\\ojobatIzquierda1.png",
                "imagenes\\\\ojobatIzquierda2.png",
                "imagenes\\\\ojobatIzquierda3.png",
                "imagenes\\\\ojobatIzquierda4.png"
            };
        } else {
            return new String[]{
                "imagenes\\\\Murcielago_izquierda1.png",
                "imagenes\\\\Murcielago_izquierda2.png",
                "imagenes\\\\Murcielago_izquierda3.png",
                "imagenes\\\\Murcielago_izquierda4.png"
            };
        }
    }
//------------------------------ CANTIDAD MURCIELAGOS ----------------------------------
------
    public static int generarMurcielago(
       int k,
       Murcielago[] murcielagos,
       int cantMurcielagoGenerados,
```

```
            int cantMurcielagoTotal,
            int anchoVentana,
            int alturaVentana,

            String [] framesDerecha, // ahora al generar murcielagos es encesario agregar los sprites correspoondientes
            String [] framesIzquierda,
            Menu menu) {

            if (cantMurcielagoGenerados >= cantMurcielagoTotal) {
                return cantMurcielagoGenerados; // No generar más, retorno la misma cantidad
            }
            Random random = new Random();
            int borde = random.nextInt(4);
            int x = 0;
            int y = 0;

            if (borde == 0) { // Lado izquierdo
                x = 0;
                y = random.nextInt(alturaVentana);
            } else if (borde == 1) { // Lado derecho
                x = anchoVentana - menu.getAncho();
                y = random.nextInt(alturaVentana);
            } else if (borde == 2) { // Lado superior
                x = random.nextInt(anchoVentana - menu.getAncho());
                y = 0;
            } else { // Lado inferior
                x = random.nextInt(anchoVentana - menu.getAncho());
                y = alturaVentana;
            }
```

```
            // Los nuevos murcielagos recibiran el sprite correspondiente a la ronda
            murcielagos[k] = new Murcielago(x, y, framesDerecha, framesIzquierda);
            return cantMurcielagoGenerados + 1; // Incremento la cantidad generada
        }


//------------------------------------ ACTUALIZAR MURCIELAGOS ----------------------
-------------
        public static int actualizarMurcielagos(Murcielago[] murcielagos,
                int cantMurcielagoPantalla,
                int cantMurcielagoGenerados,
                int cantMurcielagoTotal,
                int anchoVentana,
                int alturaVentana,
                int velocidad,
                Menu menu,
                int ronda) { // el numero de ronda es necesario ya que se lo pasara como
parametro a "generarMurcielago"

                // Llamamos a obtenerSprites para cada lado con su valor ronda para el
enemigo
                String[] framesIzquierda = obtenerSpritesIzquierdaPorRonda(ronda);
                String[] framesDerecha = obtenerSpritesDerechaPorRonda(ronda);

                // -------------------Restablecer Murcielagos----------------------------------
                int vivosActuales = 0;   // Nos indicará la cantidad de murcielagos vivos
que están actualmente
                for (int i = 0; i < murcielagos.length; i++) {
                        if (murcielagos[i] != null) {
                                vivosActuales++;
                        } //Recorremos y marcamos la cantidad de vivos
                }
```

```
		/* La variable "faltan" nos sirve para indicar cuantos enemigos faltan en pantalla
		* Esto es para que se regeneren los que faltan en pantalla.
		*/
		int faltan = cantMurcielagoPantalla - vivosActuales;
		for (int i = 0; i < murcielagos.length; i++) {
			if (murcielagos[i] == null && faltan > 0 && cantMurcielagoGenerados < cantMurcielagoTotal) {
				cantMurcielagoGenerados = Funciones_utiles.generarMurcielago(
				i, murcielagos, cantMurcielagoGenerados, cantMurcielagoTotal,
			anchoVentana, alturaVentana, framesDerecha, framesIzquierda, menu);
					faltan--;
				vivosActuales++;
			}
		}
		return cantMurcielagoGenerados; // Retornamos el nuevo contador actualizado
	}
}
```

. Clase PantallaFinJuego:

```
	// Constructor
	public PantallaFinJuego(int anchoVentana, int alturaVentana, int finDeJuego) {
		this.ancho = anchoVentana;
		this.altura = alturaVentana;
		this.x = anchoVentana / 2;
		this.y = alturaVentana / 2;
		this.estadoFin = finDeJuego;
```

```
		if (estadoFin == 1) {
			this.imagenFin = Herramientas.cargarImagen("imagenes\\\\pantalla_ganaste.png");
		} else {
			this.imagenFin = Herramientas.cargarImagen("imagenes\\\\pantalla_perdiste.png");
		}
	}

	// Para salir del Juego
	public void salirJuego(Entorno entorno) {
		if(estadoFin == 2) {
			if(entorno.sePresiono(entorno.TECLA_ESCAPE)) {
				System.exit(0);
			}
		}
	}

	// Dibujamos la imagen del Fin Juego
	public void dibujarImagenFinJuego(Entorno entorno) {
		entorno.dibujarImagen(this.imagenFin, this.x, this.y, 0, 1.5);
	}

	// X
	public int getX() {
		return x;
	}
	public void setX(int x) {
		this.x = x;
	}
	// Y
```

```
	public int getY() {
		return y;
	}
	public void setY(int y) {
		this.y = y;
	}

	// Ancho
	public int getAncho() {
		return ancho;
	}
	public void setAncho(int ancho) {
		this.ancho = ancho;
	}
	// Altura
	public int getAltura() {
		return altura;
	}

}
```

. Clase Roca:

```
//	Constructor
	public Roca(int x, int y, int tipo) {
		this.x = x;
		this.y = y;
		this.ancho = 50;
		this.altura = 50;
      this.imagen = Herramientas.cargarImagen(imagenes[tipo]);
	}
```

```
	// Dibujar imagen
	public void dibujarImagenRoca(Entorno entorno) {
		entorno.dibujarImagen(this.imagen, this.x, this.y, 0, 3.5);
	}

//---------------------- BORDES DE LAS ROCAS ----------------------
	public int limiteSuperior() {
		return this.y - this.altura/2 ;
	}
	public int limiteInferior() {
		return this.y + this.altura/2;
	}
	public int limiteIzquierdo() {
		return this.x - this.ancho/2;
	}
	public int limiteDerecho() {
		return this.x + this.ancho/2;
	}
}
```

Conclusión:

Este proyecto nos permitió aplicar y consolidar nuestros conocimientos sobre Programación Orientada a Objetos (POO), mejorando notablemente nuestra lógica de programación y habilidades para diseñar estructuras organizadas y reutilizables en Java.

Además, nos brindó una valiosa experiencia en la planificación, implementación y mantenimiento de un proyecto completo, lo cual es fundamental para enfrentarnos a situaciones reales en el ámbito laboral. Esta práctica no solo refuerza lo aprendido en teoría, sino que también nos prepara para futuras tareas profesionales.

En resumen, este proyecto fue una oportunidad enriquecedora tanto en lo técnico como en lo formativo. Además, esto nos ayudó a entender mucho mejor el manejo completo de métodos y clases en Java.