Trabajo Práctico 1S 2025

“El camino de Gondolf”

Álvaro Adriano Ruiz Kuschill - 25060123311824 - adriano.rk06@gmail.com

Tomás Andrés Carrizo - 25060123473290 - tomascarrizo073@gmail.com

Franco Ezequiel Garré - 25032015424192 - Francogarre15@gmail.com

Gonzalo Sebastián Kinder - 25060200264948 - gonkin1301@gmail.com

Universidad Nacional General Sarmiento

1er semestre del 2025

Programación 1

Comisión 02

Grupo 08

Introducción

Este trabajo práctico consistió en desarrollar un videojuego en Java usando la librería Entorno. El jugador controla al mago Gondolf, atrapado en unas catacumbas y atacado por murciélagos enviados por su enemigo, Suramun. El objetivo es sobrevivir usando hechizos mágicos, esquivar obstáculos y derrotar enemigos. El juego incluye un menú lateral, control de vida y energía, generación de enemigos y distintos tipos de hechizos.

Descripción

Clase Juego.java

La clase Juego es la principal del proyecto. Extiende InterfaceJuego y contiene el loop principal del juego (tick()), donde se actualizan y dibujan todos los elementos en pantalla.

Variables:

- entorno: maneja la ventana del juego y los eventos del teclado/mouse.
- fondo: imagen de fondo.
- menu: instancia del menú lateral con las habilidades.
- gameOver: indica si el juego terminó.
- personaje: el jugador controlado.
- murcielagos: array de enemigos.
- rocas: obstáculos fijos en el mapa.
- bola, burbuja, luz: las tres habilidades disponibles.
- dañoBolaAplicado, dañoLuzAplicado: controlan que el daño en área no se aplique múltiples veces por una misma activación.
- enemigosEliminados: contador de enemigos derrotados.

Métodos principales:

- Juego(): constructor. Inicializa el entorno, los objetos del juego y empieza el ciclo del juego.
- tick(): loop del juego. Dibuja, actualiza enemigos, habilidades y verifica condiciones de victoria o derrota.
- dibujarEscenario(): muestra el fondo, el personaje y los obstáculos.
- dibujarUI(): muestra la interfaz con vida, energía y enemigos eliminados.
- mostrarGameOver(): pantalla de fin del juego.
- controlarMovimiento(): mueve al personaje si se presionan teclas WASD.
- actualizarEnemigos(): mueve, dibuja y verifica colisiones entre enemigos, habilidades y el personaje.
- generarMurcielagoAfuera(): genera enemigos en los bordes de la pantalla.
- controlarClicksYHabilidades(): gestiona los clicks del mouse y el uso de habilidades.
- dispararHabilidad(): activa la habilidad seleccionada.
- actualizarHabilidades(): mueve, actualiza y dibuja las habilidades en uso.
- aplicarDañoArea(): aplica daño en área a los enemigos cercanos.

Clase Personaje.java

Representa al mago jugador, con posición, vida, energía y sprites para cada dirección.

Variables:

- x, y: posición.
- magoFrente, magoArriba, magoDerecha, magoIzquierda: imágenes.

- dirección: hacia dónde mira.
- vida, vidaMax: salud.
- energia, energiaMax: energía para habilidades.
- energiaRecuperacion: cuánto se regenera por segundo.
- ultimoTickRecuperacion: controla el tiempo para regenerar energía.

Métodos principales:

- Movimiento (moverIzquierda, moverDerecha, etc.) con chequeo de colisiones.
- recibirDaño y estaMuerto para controlar vida.
- consumirEnergia y recuperarEnergia para manejo de energía.
- dibujar para mostrar el sprite según la dirección.

Clase Enemigo.java

Representa a un enemigo (un murciélago) que se mueve hacia el jugador, tiene animación, vida y puede atacar.

Variables:

- x, y: posición actual.
- velocidad: velocidad de movimiento constante (1.5).
- sprites: imágenes para la animación (4 frames).
- frameActual, contadorFrames: controlan la animación.
- ultimoGolpe, intervaloGolpe: controlan el tiempo entre ataques.
- vida, vidaMax: salud del enemigo.

Métodos principales:

- dibujar: muestra la animación y la barra de vida sobre el enemigo.
- moverHacia: mueve el enemigo hacia la posición del jugador usando vectores normales.
- puedePegar: verifica si puede atacar (considera protección y tiempo entre ataques).
- recibirDaño y estaMuerto: manejan la vida y el estado del enemigo.

Clase Obstáculo.java

Representa un obstáculo fijo en el juego (una roca) que impide el paso del personaje o enemigos.

Variables:

- x, y: posición del obstáculo en el mapa.
- imagen: imagen que representa visualmente la roca.
- alto, ancho: dimensiones del obstáculo (64x64 píxeles).

Métodos:

- Obstaculo(double x, double y): constructor que inicializa posición, tamaño e imagen.
- dibujar(Entorno entorno): dibuja la imagen en la posición x, y.
- Getters para x, y, ancho y alto para que otras clases puedan consultar posición y tamaño.

Clase MenuLateral.java

Esta clase representa el menú lateral del juego donde el jugador puede seleccionar entre varias habilidades (Bola de fuego, Explosión de luz y Burbuja protectora) y también salir del juego. Además, muestra barras visuales de vida y energía, y el ícono de la habilidad activa.

Variables:

- iconoBurbuja, iconoBola, iconoExplosion: imágenes que representan los íconos de cada habilidad.
- habilidadActiva: enum que indica qué habilidad está seleccionada actualmente (puede ser NINGUNA, BOLA, EXPLOSION o BURBUJA).

Métodos principales:

- MenuLateral(): constructor que carga las imágenes y pone la habilidad activa en NINGUNA.
- getHabilidadActiva() y setHabilidadActiva(Habilidad hab): para obtener o cambiar la habilidad activa.
- manejarClick(double mouseX, double mouseY): detecta si el jugador hizo click sobre alguno de los botones de habilidades o el botón salir, y actúa en consecuencia.
- dibujar(Entorno entorno, int vida, int vidaMax, int energia, int energiaMax): dibuja el menú lateral con los botones, barras de vida y energía, y el ícono de la habilidad activa resaltando el botón correspondiente.
- resaltarBoton(Entorno entorno, int y, String texto, int xTexto, int yTexto): método privado para pintar un botón seleccionado resaltado con color naranja.
- manejarEntrada(Entorno entorno): permite salir con la tecla ‘n’.

Clase Constantes

Esta clase define constantes estáticas que se usan en todo el juego para mantener valores fijos y evitar "números mágicos" repartidos por el código. Facilita modificar valores globales de forma centralizada.

Variables:

No tiene variables de instancia, sino variables estáticas y finales que representan constantes:

- VENTANA_JUEGO_ANCHO y VENTANA_JUEGO_ALTO: tamaño (ancho y alto) de la ventana del juego en píxeles (600x600).
- PERSONAJE_ANCHO y PERSONAJE_ALTO: dimensiones estándar para el sprite o hitbox del personaje (59x94).
- VELOCIDAD: valor fijo para la velocidad de movimiento, igual a 2.

Clase Habilidad

Habilidad es una clase abstracta que sirve como base para todas las habilidades que el jugador puede usar en el juego (por ejemplo, Bola de Fuego, Burbuja Protectora, etc.). Controla la animación, movimiento y estado de activación de cada habilidad.

Variables:

- sprites: arreglo de imágenes que representa las distintas frames o estados de animación de la habilidad.
- frameActual: índice del frame que se muestra actualmente para la animación.
- contadorFrames: contador para controlar el cambio de frame y velocidad de la animación.
- activa: booleano que indica si la habilidad está activa o no (visible y moviéndose).

- x, y: posición actual de la habilidad en el espacio del juego.
- dx, dy: velocidad en cada eje (dirección normalizada y multiplicada por velocidad).
- velocidad: velocidad constante a la que se mueve la habilidad (5 unidades).

Métodos principales:

- cargarSprites(): método abstracto que las subclases deben implementar para cargar sus imágenes específicas.
- activar(origenX, origenY, destinoX, destinoY): inicializa la habilidad en la posición de origen, calcula su dirección hacia el destino, y la activa para que comience a moverse y animarse.
- mover(): mueve la habilidad según su velocidad y dirección si está activa.
- actualizarAnimacion(): actualiza el frame de la animación periódicamente para dar efecto visual.
- dibujar(Entorno): dibuja la habilidad en pantalla solo si está activa.
- estaActiva(): devuelve si la habilidad está actualmente activa.
- desactivar(): desactiva la habilidad (por ejemplo, al terminar el efecto o impactar).

Clase BolaDeFuego

BolaDeFuego es una subclase concreta que hereda de Habilidad y representa la habilidad de lanzar una bola de fuego que se mueve hacia un destino, explota al llegar y muestra una animación tanto de desplazamiento como de explosión.

Variables:

- spritesExplosión: arreglo de imágenes para la animación de la explosión una vez que la bola llega al destino.
- destinoX, destinoY: coordenadas del punto objetivo donde la bola debe explotar.
- explotando: booleano que indica si la bola está en estado de explosión (detonando la animación de explosión).
- angulo: ángulo de rotación usado para dibujar la bola orientada según su dirección de movimiento.

Métodos principales:

- cargarSprites(): carga los sprites para la bola de fuego en movimiento y los de la animación de explosión desde archivos de imagen.
- activar(origenX, origenY, destinoX, destinoY): además de activar y posicionar la bola, guarda el destino y calcula el ángulo para la rotación gráfica. Reinicia el estado de explosión.
- mover(): actualiza la posición de la bola solo si está activa y no explotando. Detecta si la bola llegó cerca del destino (distancia < 10) y cambia el estado a explotando para iniciar la animación.
- estaExplotando(): devuelve si la bola está en la fase de explosión.
- actualizarAnimacion(): maneja el avance de los frames de animación tanto para la bola en movimiento como para la explosión. Al terminar la explosión, desactiva la habilidad.
- dibujar(Entorno): dibuja la bola o la explosión con la rotación y escala apropiadas según el estado.
- Getters: getX(), getY() para obtener la posición actual.
- desactivar(): desactiva la habilidad (por ejemplo, al terminar la explosión).
- getRadio(): devuelve un valor fijo para el radio del área de daño de la explosión.

Clase ExplosionDeLuz

ExplosionDeLuz es una subclase de Habilidad que representa una habilidad que muestra una animación de explosión luminosa en una posición fija (destino) durante un tiempo limitado. A diferencia de otras habilidades que se mueven, esta permanece estática y desaparece luego de un cierto número de frames.

Variables:

- duracion: número entero que indica la cantidad máxima de frames que la explosión permanece visible (30 frames).
- frameVida: contador que lleva la cantidad de frames que la habilidad lleva activa para controlar cuándo debe desactivarse.

Además, hereda de Habilidad variables como:

- sprites: arreglo de imágenes para la animación de la explosión.
- x, y: posición donde se dibuja la explosión.
- activa: estado de activación de la habilidad.
- frameActual y contadorFrames: controlan la animación.

Clase BurbujaProtectora

BurbujaProtectora es una subclase de Habilidad que representa una burbuja de protección que sigue al personaje mientras está activa y luego explota tras cierto tiempo. Esta habilidad tiene dos fases: una fase de protección móvil y una fase de explosión animada fija.

Variables:

- explosionSprites: arreglo de imágenes que forman la animación de la explosión cuando la burbuja se destruye.
- frameExplosion: índice del frame actual de la animación de la explosión.
- explotando: booleano que indica si la burbuja está en la fase de explosión.
- tiempoActivacion: almacena el tiempo en milisegundos cuando la burbuja fue activada para controlar su duración.

Además hereda variables de la clase Habilidad como sprites, x, y, activa, frameActual, y contadorFrames.

Métodos principales:

- cargarSprites(): carga la imagen estática de la burbuja protectora.
- cargarExplosionSprites(): carga las imágenes de la animación de explosión.
- activar(origenX, origenY, destinoX, destinoY): activa la burbuja posicionándola inicialmente en la posición del personaje, reiniciando contadores y el estado de explosión, y registra el tiempo de activación.
- seguirPersonaje(jugadorX, jugadorY): actualiza la posición de la burbuja para que siga al jugador mientras está activa. Controla el tiempo activo; si supera 5 segundos, desactiva la burbuja y comienza la fase de explosión.
- actualizarAnimacion(): si la burbuja está activa, actualiza la animación estándar (heredada). Si está explotando, avanza la animación de explosión y finaliza la explosión cuando se terminan los frames.
- dibujar(Entorno): dibuja la burbuja si está activa, o la animación de explosión si está en esa fase.
- protege(): devuelve si la burbuja sigue protegiendo (está activa).
- estaExplotando(): indica si la burbuja está en la fase de explosión.

Problemas encontrados y soluciones:

- **Problema:** Distinguir claramente qué habilidad está activa y mostrarlo visualmente para que el jugador no se confunda.
  Solución: Se decidió dibujar un recuadro naranja que resalta el botón seleccionado y mostrar el ícono grande de la habilidad en un cuadro arriba.
- **Problema:** Necesidad de usar valores uniformes y fáciles de modificar para tamaño de ventana y personaje, y velocidad, sin hardcodear números en cada clase.
  Solución: Se creó esta clase para centralizar las constantes, facilitando mantenimiento y evitando errores por usar diferentes valores en distintas partes.
- **Problema:** Necesidad de un comportamiento común para todas las habilidades (movimiento, animación, estado activo) sin repetir código.
  Solución: Usar una clase abstracta que defina la estructura base y que cada habilidad implemente detalles específicos (como cargar sus sprites).

Conclusiones

El desarrollo de este proyecto permitió profundizar en la implementación y manejo de clases abstractas y herencia en Java, aplicándolas a un contexto práctico de un videojuego con habilidades dinámicas. Una de las principales lecciones aprendidas fue la importancia de estructurar correctamente la jerarquía de clases para facilitar la reutilización de código y mantener una buena organización, como se evidenció en la clase base `Habilidad` y sus subclases específicas.

Además, el manejo de animaciones y estados (activo, explotando, etc.) presentó un desafío interesante que requirió diseñar mecanismos claros para controlar el flujo de las habilidades, evitando comportamientos erráticos o bugs visuales. El uso de temporizadores y control de frames fue fundamental para conseguir transiciones suaves entre animaciones y efectos.

También se valoró la necesidad de anticipar y manejar problemas como la sincronización entre el estado lógico y la representación gráfica, lo que mejoró la experiencia visual y la lógica del juego.

En resumen, el trabajo no solo cumplió con los objetivos planteados, sino que también fortaleció habilidades en diseño orientado a objetos, programación gráfica y gestión de estados en sistemas interactivos. Estos aprendizajes serán muy valiosos para futuros proyectos que requieran estructuras complejas y animaciones controladas.