Universidad Nacional de General Sarmiento

Ciclo Lectivo 2025

Materia: Programación I

Integrantes:

Franco Román Pachao, pachaofranco@gmail.com

Matias Casiba, gabrielcasiba19@gmail.com

Lucas Rodriguez, akaxolnokis@gmail.com

Introducción

Este trabajo práctico consiste en el desarrollo de un videojuego basado en Java donde el jugador controla al mago Gondolf, quien debe defenderse de oleadas de murciélagos enviados por su enemigo Suramun. El juego implementa mecánicas de movimiento, combate con hechizos, gestión de recursos (vida/energía), y obstáculos ambientales. Se incluyen elementos avanzados como sistema de oleadas progresivas, pantallas de recompensas, efectos de estado (ralentización) y drops aleatorios de pociones.

Descripción de Clases

1. **Boton.java**

   Variables:

   x, y: Posición del botón
   ancho, alto: Dimensiones
   texto: Etiqueta del botón
   seleccionado: Estado de selección

   Métodos:

   dibujar(): Renderiza el botón (amarillo si está seleccionado).
   fueClickeado(): Detecta colisión con el clic del mouse.
   setSeleccionado(): Cambia estado de selección.

   Problemas/Soluciones:

   Problema: Los botones no indicaban visualmente su selección.
   Solución: Se implementó cambio de color al hacer clic (seleccionado ? Color.yellow).

2. **Enemigo.java**

   Variables:

   x, y, ancho, alto: Posición y tamaño
   ralentizado, tiempoRalentizado: Control de efecto de ralentización
   spriteSheet: Sprites para animación
   frameActual, ticks: Gestión de frames de animación

   Métodos:

   dibujar(): Renderiza la animación del murciélago.
   moverHaciaPersonaje(): Persigue al jugador con velocidad reducida si está ralentizado.
   ralentizar(): Aplica efecto de ralentización.

   Problemas/Soluciones:

   Problema: Movimiento poco fluido y sin animación.
   Solución: Implementación de spritesheet con cambio de frames controlado por velocidadAnimacion.

3. **Hechizo.java**

   Variables:

   nombre, costo, radio, color, duracionRalentizacion

   Métodos:

dibujar(): Muestra el área de efecto del hechizo.
enRango(): Detecta colisión con enemigos.
aplicarEfectoRalentizacion(): Ralentiza enemigos en el área.

Problemas/Soluciones:
Problema: El hechizo de ralentización no afectaba a los enemigos.
Solución: Se añadió duracionRalentizacion y lógica para aplicar el efecto.

4. Juego.java (Clase principal)
Variables clave:
oleadaActual, objetivosOleada: Control de oleadas
mostrandoRecompensas, botonesRecompensa: Sistema de recompensas
pociones: Gestión de ítems

Métodos principales:
tick(): Bucle principal del juego.
generarPocion(): Spawnea pociones al eliminar enemigos.
controlarFinDeOleada(): Avanza a la siguiente oleada o finaliza el juego.
prepararOleada(): Reinicia enemigos para una nueva oleada.

Problemas/Soluciones:
Problema: Pociones aparecían dentro de rocas.
Solución: Validación de posición con posicionValida usando colisiones.
Problema: Transición entre oleadas no reiniciaba enemigos.
Solución: Se implementó prepararOleada() para resetear el array de enemigos.
Problema: Hechizo "Transmutación Final" eliminaba enemigos.
Solución: Se creó un condicional para verificar el tipo de hechizo.

5. Menu.java
Variables:
fondoMenu: Imagen de fondo del menú lateral.

Métodos:
dibujar(): Renderiza el menú con título "Hechizos".

Problemas/Soluciones:
Problema: Diseño visual poco atractivo.
Solución: Se reemplazó el color plano por una textura de madera (fondoMadera.jpg).

6. Personaje.java
Variables:
velocidad: Velocidad base modificable (efecto de recompensa).
imagen: Sprite del mago.

Métodos:
aumentarVelocidad(): Aplica bonus de velocidad (recompensa).
reproducirSonidoDamage(): Efecto de sonido al recibir daño.

Problemas/Soluciones:
Problema: Movimiento no respondía fluidamente.
Solución: Se normalizó el sistema de colisiones con rocas usando colisionaConRocaAlMover().

7. Pocion.java
Variables:
tipo: "vida" o "mana".
cantidad: Puntos a restaurar.

Métodos:
colisionaConPersonaje(): Detecta recogida por el jugador.

8. ReproducirMusica.java
Métodos:
reproducirMusicaLoop(): Reproduce música en loop con Clip.LOOP_CONTINUOUSLY.

9. Roca.java
Variables:
imagen: Textura de roca.

Métodos:
dibujar(): Renderiza la roca escalando la imagen.

Implementación

El código fuente completo está disponible en GitHub, con las siguientes características destacadas:

1. **Sistema de Oleadas:**
   - Tres oleadas con objetivos crecientes (20, 40, 60 enemigos).
   - Velocidad y daño de enemigos aumentan por oleada (velocidadesOleada, danioPorOleada).
2. **Recompensas:**
   - Pantalla de selección post-oleada con tres opciones:
     - +20 vida.
     - +20 energía.
     - +velocidad.
3. **Hechizos:**
   - Bomba de Agua (costo 0, radio pequeño).
   - Tormenta de Fuego (costo 20, radio grande).
   - Transmutación Final (ralentización masiva).
4. **Items (Pociones):**
   - 30% de probabilidad de dropeo al eliminar enemigos.
   - Restauran vida o maná (+5 puntos).
5. **Animaciones:**
   - Spritesheet para animación de murciélagos (spriteBat.png).
   - Cambio de frames controlado por velocidadAnimacion.

Conclusiones

La gestión de colisiones (enemigos, rocas, pociones) requirió algoritmos de detección precisos. El sistema de oleadas implicó coordinar múltiples estados del juego (espera, recompensas, combate).

Se priorizó jugabilidad sobre realismo (ej: pociones aparecen en ubicaciones válidas). Los efectos de sonido y música mejoran inmersión (goldCobra.wav en loop).

Jugabilidad balanceada con dificultad progresiva. Implementación exitosa de requerimientos opcionales (oleadas, recompensas, pociones).

La modularización en clases facilitó el desarrollo en equipo. Reutilizar objetos (ej: enemigos null vs. nuevos) mejoró el rendimiento. Iterar con valores de variables (ej: probabilidadPocion) ajusta la experiencia.

El proyecto nos ayudó a trabajar como equipo y a practicar el diseño de algoritmos orientado a objetos en Java.