

Trabajo Práctico Integrador – Programación II

ROLGAR II – Juego de Exploración y Combate por Turnos

Integrantes:

- Juana de la Torre – Legajo: 113734
- Nicolas Agustín Feldman – Legajo: 113735
- Matías Ezequiel Pasteloff – Legajo: 113224
- Alejo Ilan Segal – Legajo: 113104

Profesor: Gustavo Adolfo Schmidt

Curso: CB100 Algoritmos y Estructura de datos (Curso 1 - Mie-Jue) - Schmidt

Fecha: 16/11/2025

Contenido

1. Informe del Trabajo	1
1.2 Objetivo General del Juego.....	1
1.3 Mecánicas Principales	1
1.4 Arquitectura del Proyecto	2
1.5 Decisiones de Diseño	3
2. Cuestionario del TP.....	3
3. Manual del Usuario	5
4. Manual del Programador.....	5
5. Conclusiones	6
6. Anexos	6
Respuestas al Cuestionario.....	7
Diagrama del Mapa / Tablero (Esquema Simplificado)	8
Explicaciones Adicionales.....	9

1. Informe del Trabajo

1.1 Introducción

ROLGAR II es un videojuego 2.5D de exploración y combate por turnos, desarrollado íntegramente en Java. El proyecto incluye un tablero tridimensional (X, Y, Z), múltiples jugadores, enemigos con IA, sistema de cartas, alianzas y una interfaz gráfica basada en sprites.

El trabajo integra programación orientada a objetos, estructuras de datos, modelado de entidades, manejo de eventos, carga de recursos multimedia y diseño modular.

1.2 Objetivo General del Juego

El objetivo principal es **eliminar a todos los enemigos del mapa** explorando distintos niveles conectados por rampas. Los jugadores avanzan cooperando o individualmente, gestionando recursos, enfrentando enemigos y utilizando cartas especiales.

La partida finaliza en: - **Victoria:** si no quedan enemigos vivos. - **Derrota:** si todos los jugadores mueren.

1.3 Mecánicas Principales

Exploración

- Tablero 3D con 10 niveles (coordenada Z).
- Habitaciones, pasillos y rampas generados por GeneradorDeMundo.
- Visibilidad limitada según rango de visión del personaje.

Movimiento por Turnos

En modo multijugador local, cada turno equivale exactamente a un solo movimiento por jugador.

Esto significa que, una vez que el jugador se desplaza a un casillero válido, el turno avanza automáticamente y pasa al siguiente jugador en el orden establecido.

Acciones especiales (como usar cartas, subir/bajar de nivel o iniciar combate) también consumen el turno si son acciones activas.

De esta forma, el flujo de juego se vuelve dinámico y equilibrado entre los participantes.

Combate 1v1

- Se inicia al entrar en la casilla de un enemigo o mediante ataque manual.
- Turnos alternados entre jugador y enemigo.
- Log de combate visible en la interfaz.

Inventario y Cartas

Cada jugador posee un inventario de 10 slots (hotbar). Cartas implementadas: - Ataque doble - Curación parcial - Curación total - Curación a aliado - Escudo - Esquivar daño - Invisibilidad - Doble movimiento - Aumento de vida - Teletransportación - Robo de carta

Alianzas entre Jugadores

Los jugadores pueden: - Proponer alianza - Aceptar / rechazar - Transferir cartas - Recibir ayuda de daño en combate - Romper la alianza (aleatoriamente o por distancia)

IA de Enemigos

- Ataques cuando un jugador está en la misma celda.
- Gestión unificada mediante AdministradorDeEnemigos.

Visión del jugador (5 x 5)

El rango de visión del jugador está configurado como un área de **5x5 casilleros** centrada en su posición.

En versiones anteriores se evaluó un rango 3x3, pero resultaba insuficiente para la escala de los mapas, por lo que se amplió a 5x5 para permitir mejor navegación y claridad visual. Solo las casillas dentro de este radio son visibles; el resto permanece oculto o marcado como visitado según correspondiera.

1.4 Arquitectura del Proyecto

Paquete com.entidades

- Personaje: Jugador controlable.
- Enemigo: Enemigos del mundo.
- Entidad: Clase base para cualquier ser vivo.
- Alianza: Gestión de relaciones entre jugadores.

Paquete com.items

- Carta (abstracta) + todas las cartas del juego.
- Inventario y lógica de administración de slots.

Paquete com.tablero

- Tablero, Casillero, Coordenada, TipoCasillero.
- Mapa tridimensional.

Paquete com.logica

- Core del juego: turnos, combates, IA, alianzas.

Paquete com.ui

- Renderizadores: mundo, HUD, combate, menú.
 - SpriteManager: carga y fallback de bitmaps.
 - PanelJuego, VentanaJuego.
-

1.5 Decisiones de Diseño

1. **Mapa tridimensional real (X, Y, Z):** Permite exploración profunda y cumple el requisito de niveles.
 2. **Sprites bitmap en PNG:** Fácil edición y compatibilidad con Java.
 3. **Cartas polimórficas:** Facilita agregar nuevas cartas sin cambiar la lógica.
 4. **Turnos centralizados:** Garantiza orden y control en multijugador.
 5. **Renderizadores separados:** Menor acoplamiento entre lógica y presentación.
 6. **Alianzas dinámicas:** Favorecen la estrategia y cooperación.
-

2. Cuestionario del TP

1) ¿Qué es un SVN?

SVN (Subversion) es un sistema de control de versiones centralizado.

Permite que múltiples desarrolladores trabajen sobre los mismos archivos almacenados en un servidor central, registrando cambios, versiones y historial del proyecto.

Cada usuario obtiene una copia de los archivos, los modifica y luego sube los cambios nuevamente al servidor.

Características clave:

- Repositorio centralizado.
- Historial completo de versiones.
- Permite revertir cambios, comparar versiones y colaborar en equipo.

2) ¿Qué es una “Ruta absoluta” o una “Ruta relativa”?

- Ruta absoluta

Describe la ubicación completa de un archivo o carpeta desde la raíz del sistema.

No depende de dónde se encuentre el programa o el usuario.

Ejemplos:

En Windows: C:\Usuarios\Juan\Documentos\archivo.txt

En Linux: /home/juan/documentos/archivo.txt

Siempre es válida sin importar dónde estés ubicado.

- Ruta relativa:

Describe la ubicación a partir de la carpeta actual (directorio de trabajo).

Depende de dónde está parado el programa.

Ejemplos:

..../imagenes/bicho.png → subir un nivel y entrar a "imagenes"

sprites/enemigo.png → carpeta sprites relativa al proyecto

Es más flexible para proyectos que pueden moverse de carpeta.

3) ¿Qué es Git?

Git es un sistema de control de versiones distribuido, utilizado para gestionar el historial y la colaboración en proyectos de software.

A diferencia de SVN:

- Cada desarrollador tiene una copia completa del repositorio (no solo una copia de trabajo).
- Permite trabajar sin conexión.
- Es más rápido, seguro y flexible.

Conceptos clave:

- Commit: registrar un cambio
- Branch: rama de trabajo independiente
- Merge: combinar cambios
- Push / pull: subir o traer cambios del repositorio remoto (por ejemplo GitHub)

Git es actualmente el estándar de la industria para desarrollo colaborativo.

3. Manual del Usuario

3.1 Requisitos

- Tener instalado Java 17 o superior.
- Compilar y ejecutar el programa desde consola o un IDE (Eclipse, IntelliJ, etc.).

3.2 Inicio del Juego

Al iniciar, el menú permite elegir entre 1 y 4 jugadores.

3.3 Controles

Movimiento: - W / A / S / D → direcciones - Q / E / Z / C → diagonales

Acciones: - **F** → atacar jugador adyacente - **1..0** → usar carta del slot - **T** → iniciar transferencia de carta - **L** → proponer alianza - **Y** → aceptar alianza - **N** → rechazar alianza - **ENTER** → finalizar turno - **ESC** → pausar

3.4 HUD y UI

- Mapa visible según visión del jugador.
 - Hotbar de cartas.
 - Log de combate.
 - Sprite del jugador con nombre debajo.
-

4. Manual del Programador

4.1 Requisitos Técnicos

- Java 17 o superior.
- No usa librerías externas.

4.2 Estructura del Proyecto

Explicación de paquetes (ver sección arquitectura).

4.3 Cómo Agregar un Nuevo Enemigo

1. Crear sprite PNG en /sprites.
2. Crear clase nueva heredando de Enemigo o instanciar uno en el mundo.
3. Registrar nombre en SpriteManager.

4.4 Cómo Agregar una Carta Nueva

1. Extender Carta.
2. Implementar aplicarEfecto.
3. Agregar sprite.

4.5 IA y Combate

- AdministradorDeEnemigos gestiona turnos.
- AdministradorDeCombate maneja flujo 1v1.

4.6 Turnos y Multijugador

- Clase Turno rota índices.
 - AdministradorDeJuego decide cuándo termina la partida.
-

5. Conclusiones

El trabajo permitió integrar programación orientada a objetos, estructuras de datos, arquitectura de software y diseño de videojuegos en un proyecto completo. La modularidad lograda permite expansión futura del juego con relativa facilidad.

6. Anexos

- Sprites PNG.
 - Mapa del mundo.
 - Audio del juego.
 - Código fuente completo.
-

FIN DEL INFORME

Respuestas al Cuestionario

1. ¿Cuál es la finalidad del trabajo?

El trabajo tiene como finalidad integrar conceptos de programación orientada a objetos, estructuras de datos, manejo de estados, diseño modular y renderizado gráfico, desarrollando un videojuego completo con interacción, combate, inventario, enemigos y multijugador local.

2. ¿Qué estructura de datos se eligió y por qué?

Se utilizaron: - **Arreglos y ArrayList** para manejar listas dinámicas de jugadores, enemigos y cartas. - **Matriz 3D** para el tablero (X, Y, Z), lo cual permite múltiples niveles. - **Map<String, BufferedImage>** para sprites, por su acceso rápido por clave. - **Enums** para tipos de casillero. La elección priorizó eficiencia de acceso y claridad de implementación.

3. ¿Qué patrones de diseño se usaron?

- **Singleton** (SpriteManager) para centralizar recursos visuales.
- **Strategy** para efectos de cartas (cada carta implementa su propia lógica).
- **State Machine** en el ControladorJuego (RUNNING, MENU, PAUSED, COMBAT, etc.).
- **Modularización por responsabilidad** (RenderizadorMundo, Combate, HUD...).

4. ¿Cómo se organizó el código?

Por paquetes según dominio: - entidades → personajes, enemigos, alianzas. - items → cartas. - logica → reglas del juego, combates, turnos. - ui → renderizado, sprites, ventanas. - tablero → casillero, coordenada, tipos.

5. ¿Qué dificultades se presentaron?

- Manejo de turnos entre múltiples jugadores.
- Sincronizar renderizado y lógica.
- Evitar conflictos de merge.
- Implementar combate y uso de cartas dentro y fuera del combate.
- Controlar visibilidad por jugador.

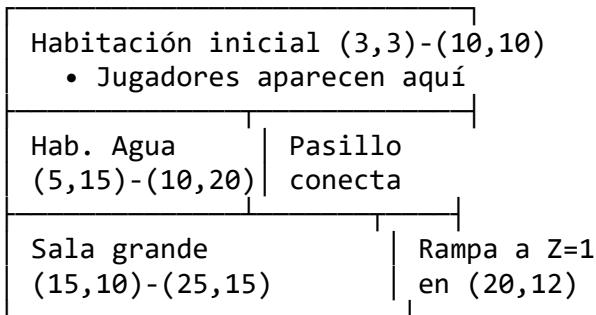
6. ¿Qué se aprendió del proyecto?

- Programación modular a gran escala.
- Trabajo con gráficos 2D.
- Gestión de inventarios y estados.
- Manejo de listas dinámicas.

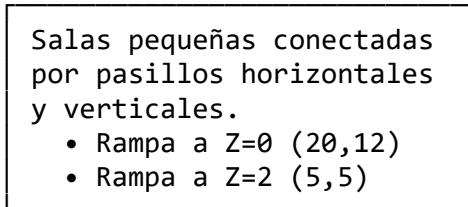
- Depuración en proyectos grandes.
-

Diagrama del Mapa / Tablero (Esquema Simplificado)

Nivel Z = 0 (Inicio)



Nivel Z = 1 (Cruce)



Z = 2–8 (Salas temáticas)

- Cavernas
- Prisión
- Salón grande
- Laberinto
- Nido

Cada nivel tiene:

- Habitaciones talladas
- Pasillos
- Enemigos específicos
- Cartas distribuidas

Nivel Z = 9 (Jefe Final)

Sala del Trono (10,15)-(30,23)
• REY MAGO (Jefe)
• Cartas de curación

Aclaración importante:

Actualmente los enemigos se colocan en posiciones fijas establecidas en el código.

La colocación aleatoria por nivel está planificada como mejora futura y queda pendiente para una versión ampliada del proyecto.

Explicaciones Adicionales

Movimiento y turnos

Cada jugador puede moverse una casilla por turno (o dos con cartas). Cuando un jugador termina su movimiento, el turno pasa al siguiente.

Combate

Ocurre cuando: - El jugador pisa la misma casilla que un enemigo. - Un enemigo detecta un jugador adyacente.

El combate es **1 vs 1**, por turnos: - Atacar - Usar carta - Huir (probabilidad del 30%)

Cartas

Cada carta tiene un efecto único, por ejemplo: - **Invisibilidad**: El enemigo no golpea el siguiente ataque. - **Esquivar daño**: Probabilidad de evitar el próximo golpe. - **Ataque doble**: Duplica daño. - **Escudo**: Da vida protegida. - **Curaciones**: Parcial o total. - **Teletransportación**: Mueve a coordenadas específicas.

Alianzas

Jugadores adyacentes pueden: - Proponer alianza (L) - Aceptar (Y) / Rechazar (N) - Transferir cartas - Ayudarse en combate

Las alianzas pueden romperse aleatoriamente.

Victoria y Derrota

- **Victoria**: No quedan enemigos vivos.
- **Derrota**: Todos los jugadores mueren.